[redacted] Statistical Learning Assessment Two Part A

Student Number: [redacted]

Problem 1: Olympic games

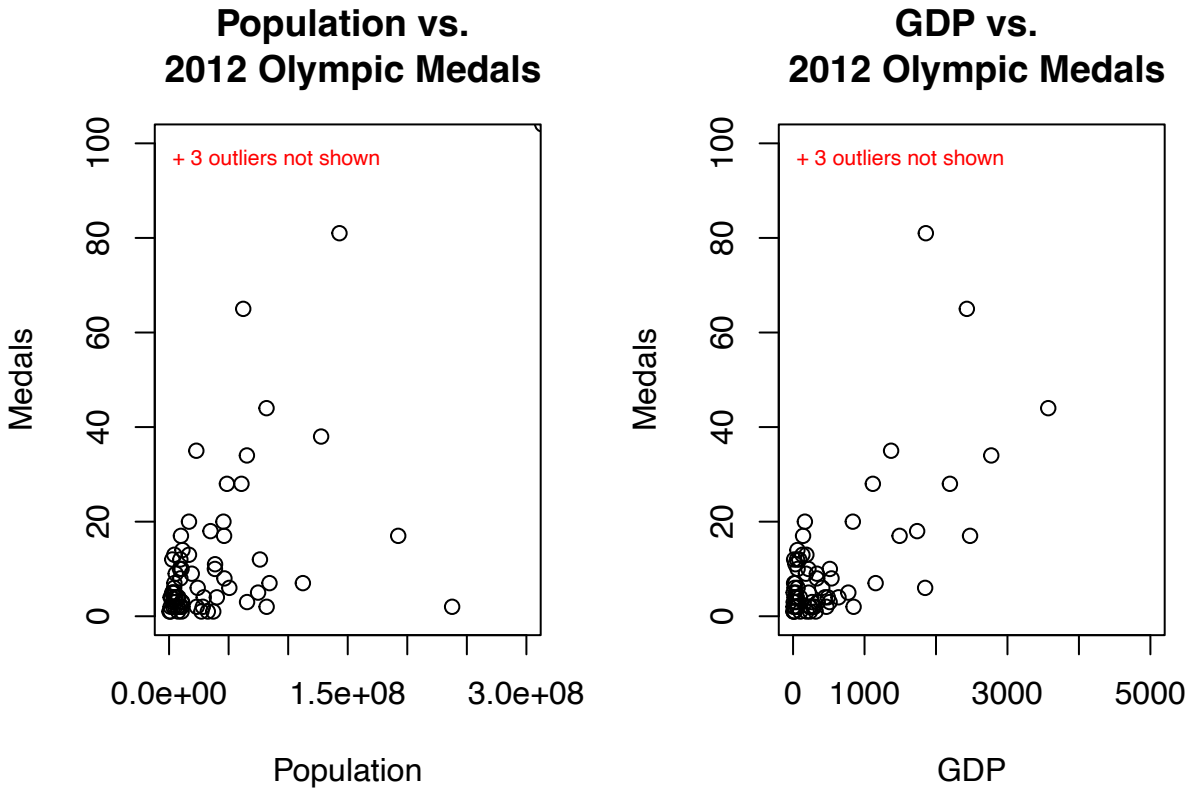
To fit a linear regression model for 2012 Olympic medal counts against both population and GDP, we must consider the following equation for multiple linear regression (MLR):

$$medals_X = \beta_0 + \beta_1 * p_X + \beta_2 * g_X + \epsilon_X$$

where X represents each country in the dataset, β_0 is the intercept, p is Population, g is GDP, β_1, β_2 are the coefficients for Population and GDP respectively, and ϵ is the residual error.

Preliminary findings

```
# Loading Olympic Games dataset
medals = read.csv('medal_pop_gdp.csv',header=T)

# Plotting MLR inputs individually
par(mfrow=c(1,2))

plot(medals$Population,medals$Medal2012,
     main="Population vs. \n 2012 Olympic Medals",
     xlab='Population',ylab='Medals',
     mtext(side=3, adj=0.1, line=-1.25, col = "red", cex = 0.7,
           '+ 3 outliers not shown'),
     xlim=c(0,300000000),ylim=c(0,100)) # Limiting range/domain to show trend

plot(medals$GDP,medals$Medal2012,
     main="GDP vs. \n 2012 Olympic Medals",
     xlab='GDP',ylab='Medals',
     mtext(side=3, adj=0.1, line=-1.25, col = "red", cex = 0.7,
           '+ 3 outliers not shown'),
     xlim=c(0,5000),ylim=c(0,100)) # Limiting range/domain to show trend
```




```
# Calculating outliers outside of axes limits
pop_out = (medals$Population > 300000000 | medals$Medal2012 > 100)
gdp_out = (medals$GDP > 5000 | medals$Medal2012 > 100)
pop_out_sum = sum(pop_out) # 3 outliers
gdp_out_sum = sum(gdp_out) # 3 outliers
```

Visualising the model inputs individually, we see that Population and GDP both show positive linear trends against the number of medals won in the 2012 Olympic Games. This is a good indication that both variables are likely to contribute significantly to the MLR model for medals. However, we cannot confirm this observation until other factors are considered (e.g. possible multicollinearity).

Building the MLR model

```
medals_mlr = glm(Medal2012 ~ Population + GDP, data=medals)
summary(medals_mlr)
```

```
##
## Call:
## glm(formula = Medal2012 ~ Population + GDP, data = medals)
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.076e+00  1.500e+00   4.051 0.000133 ***
## Population  5.247e-09  7.193e-09   0.729 0.468225
## GDP         7.564e-03  7.325e-04  10.326 1.45e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 132.1562)
```

```
##
##     Null deviance: 28402.8  on 70  degrees of freedom
## Residual deviance:  8986.6  on 68  degrees of freedom
## AIC: 553.19
##
## Number of Fisher Scoring iterations: 2
```

Analysing the MLR model summary, we see that both Population and GDP have (small) positive coefficients. However, looking at the **p-values**, $Pr(>|t|)$, we see that Population is not statistically significant, $Pr(>|t|) > 0.05$. In contrast, GDP has a highly significant positive effect considering its p-value. Given the initial observation that both Population and GDP show positive linear trends against medal count, it is possible there exists some relationship between the two variables.

Assessing multicollinearity

There are two ways to diagnose multicollinearity in the model. Firstly, using `cor()` we can calculate the Pearson correlation between Population and GDP, given by

$$r = \frac{\Sigma(x_1 - \bar{x}_1)(x_2 - \bar{x}_2)}{\sqrt{\Sigma(x_1 - \bar{x}_1)^2 \Sigma(x_2 - \bar{x}_2)^2}}$$

where x_1, x_2 are the variables and $\bar{x}_1, \bar{x}_2$ are their respective means. This will surface any relationship between any two independent variables. Alternatively, and most commonly used for models with more than two inputs, we can calculate the Variance Inflation Factor (VIF) given by

$$VIF = \frac{1}{1 - R^2}.$$

```
# Diagnostic 1: Correlation
pop_gdp_cor = cor(medals$Population,medals$GDP,
                  method='pearson',use='complete.obs')

# Diagnostic 2: Variance Inflation Factor (VIF)
# glm() does not output R-Squared in summary, using lm()
pop_gdp_model = lm(Population ~ GDP, data=medals)
pop_gdp_vif = 1 / (1- (summary(pop_gdp_model)$r.squared))

paste('The Pearson correlation between Population and GDP is',
      round(pop_gdp_cor,4))
```

```
## [1] "The Pearson correlation between Population and GDP is 0.4715"
```

```
paste('The VIF is',round(pop_gdp_vif,4))
```

```
## [1] "The VIF is 1.2859"
```

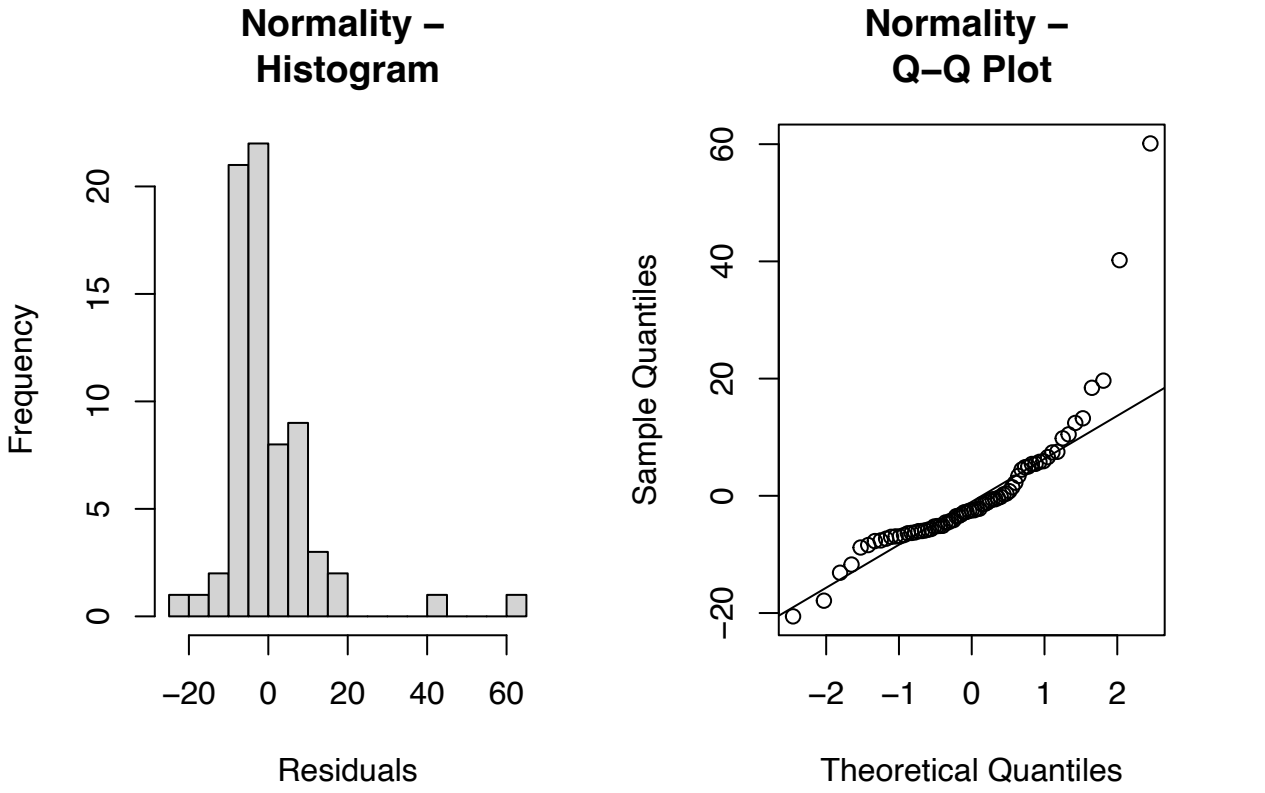
The Pearson correlation value suggests there is a moderate positive relationship between Population and GDP. However is it clear that when considering Population and GDP as inputs, multicollinearity is not an issue to the medals model. This is due to the low VIF value (anything below 5 is considered acceptable to reject assumptions of multicollinearity).

Residual analysis

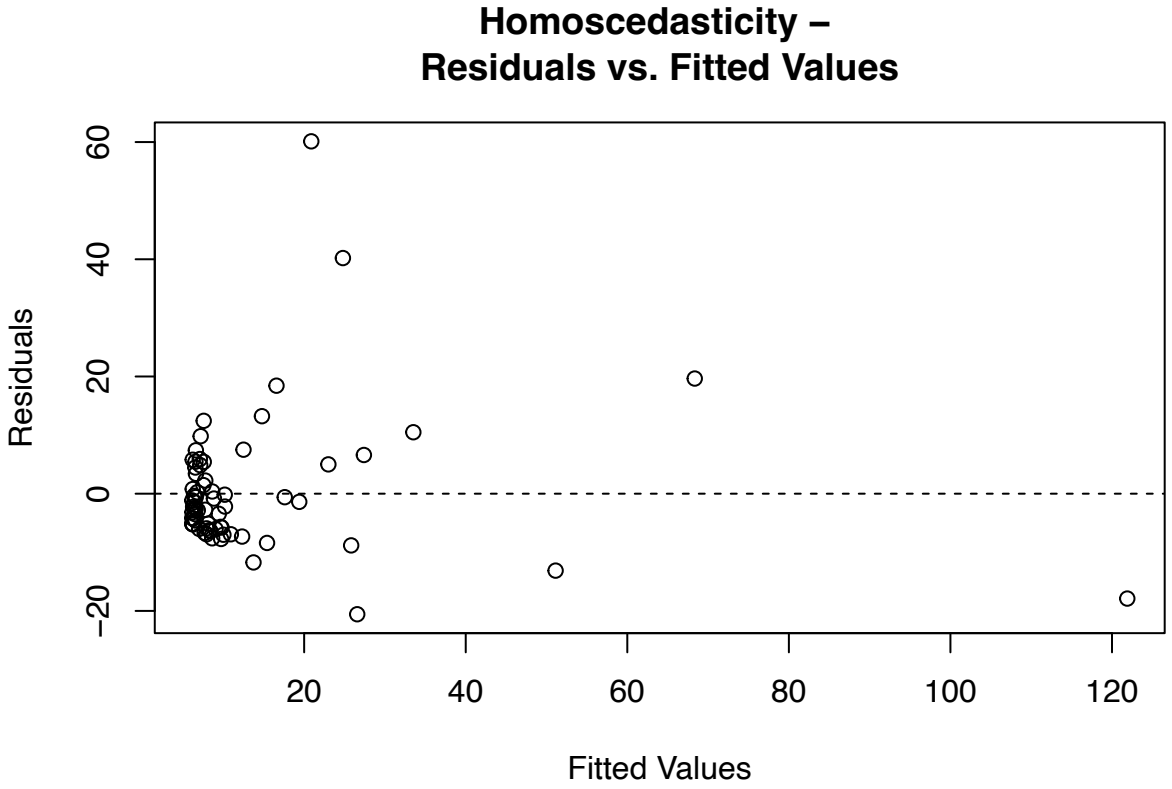
Now that multicollinearity has been assessed (and disproved), we should assess residuals. Linear regression follows the assumption that errors/residuals approximate a normal distribution with zero mean and unknown variance, $\epsilon \sim \mathcal{N}(0, \sigma^2)$. It also assumes homoscedasticity; the variance of residuals across the independent

variables (Population and GDP) are constant. Violating these assumptions is a good indication that the current medals MLR is not appropriate for the data.

```
# Assessing normality
par(mfrow=c(1,2))
hist(medals_mlr$residuals,breaks=20,xlab='Residuals',main='Normality - \nHistogram')
qqnorm(medals_mlr$residuals,main='Normality - \nQ-Q Plot')
qqline(medals_mlr$residuals)
```




```
# Assessing homoscedasticity
plot(medals_mlr$fitted.values,medals_mlr$residuals,,xlab='Fitted Values',
     ylab='Residuals',main='Homoscedasticity - \nResiduals vs. Fitted Values')
abline(h=0,lty='dashed')
```



Analysing normality in the residuals, we see a clear positive skew in the histogram which is not indicative of a normal distribution. Furthermore, the Q-Q plot shows significant deviation from the normality line, particularly at the tails, which confirms the lack of normality of residuals in this model. Secondly, looking at the residuals vs model fitted values plot we can assess homoscedasticity; the absence of clear pattern and 'randomness' of data about the line $residuals = 0$ implies the model is homoscedastic. Given the presence of both clustering and anomalous data points it suggests the model violates the homoscedasticity assumption.

It is clear that the magnitude of fitted values (outputs) has an effect on the variance of the residuals. Contextually the model outputs have relatively small residuals where the medal counts are low, but the variance of residuals begins to increase as medal count increases. This will impact model accuracy; especially if predictions extrapolate beyond the observed data.

Log transformations & cross-validation

Log-transforming any combination of inputs/outputs has the potential to surface a model more appropriate than initially fitted, by reducing data skewness, variance and normality. Given the initial medals model exhibits issues with each of these attributes, eight possible combinations arise:

Log-transformed (Y/N)	2012 Medal Counts	Population	GDP
Model 1	N	N	N
Model 2	N	Y	N
Model 3	N	N	Y
Model 4	N	Y	Y
Model 5	Y	N	N
Model 6	Y	Y	N
Model 7	Y	N	Y
Model 8	Y	Y	Y

We also begin the process of cross-validation:

1. Split the `medals` data into a 80/20 train-test-split.
2. Generate MLR models for each log transformation based on the train data.
3. Compare both the predictive log-likelihood & AIC for each model

It is important to note the log-likelihood for each model must be comparable. For models without log-transformed outputs, log-likelihood is calculated as

$$\log \mathcal{L} = \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}\hat{\sigma}} \exp\left(-\frac{(x_i - \hat{x_i})^2}{2\hat{\sigma}^2}\right)$$

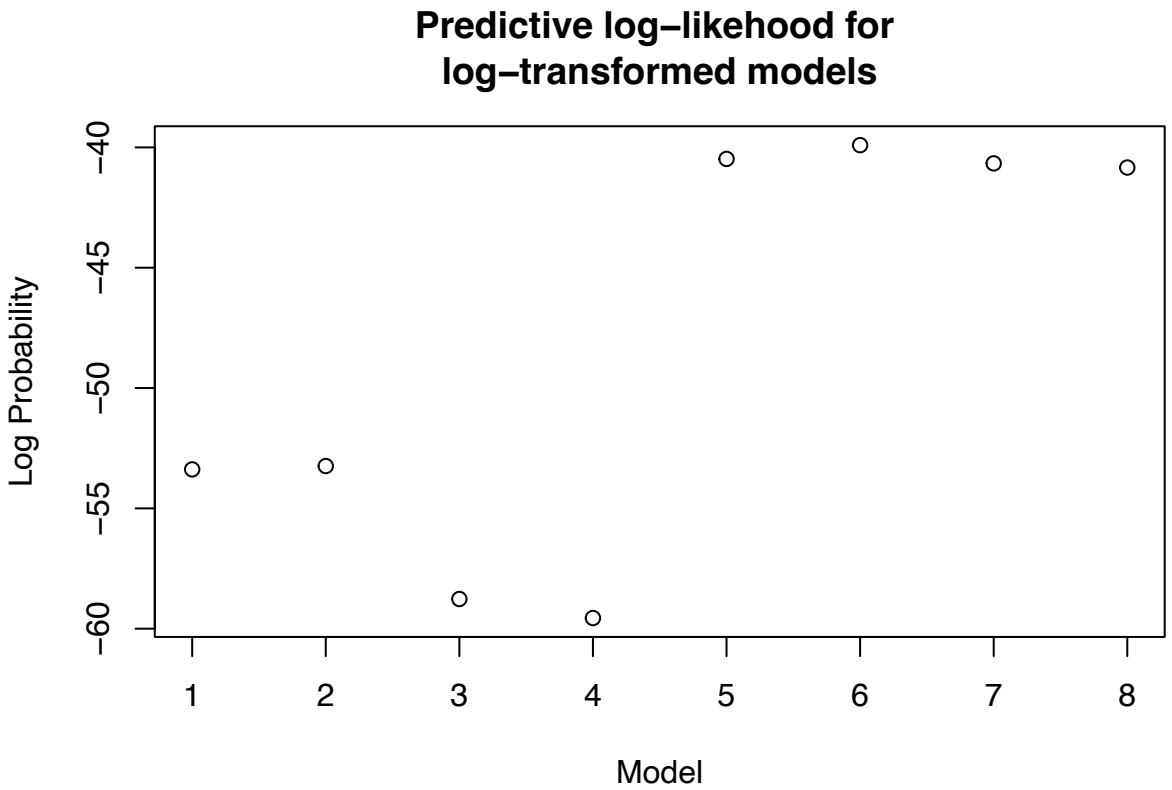
where x_i is the actual medal count in the training data, and estimated parameters $\hat{x_i}, \hat{\sigma}$ are the predicted model counts and residual standard deviation from the regression model respectively. In R, this can be simply implemented using the `dnorm()` function. However, models with log-transformed outputs will have a different log-likelihood calculation

$$\log \mathcal{L} = \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}\hat{\sigma}} \exp\left(-\frac{(\log(x_i) - \hat{x_i})^2}{2\hat{\sigma}^2}\right)$$

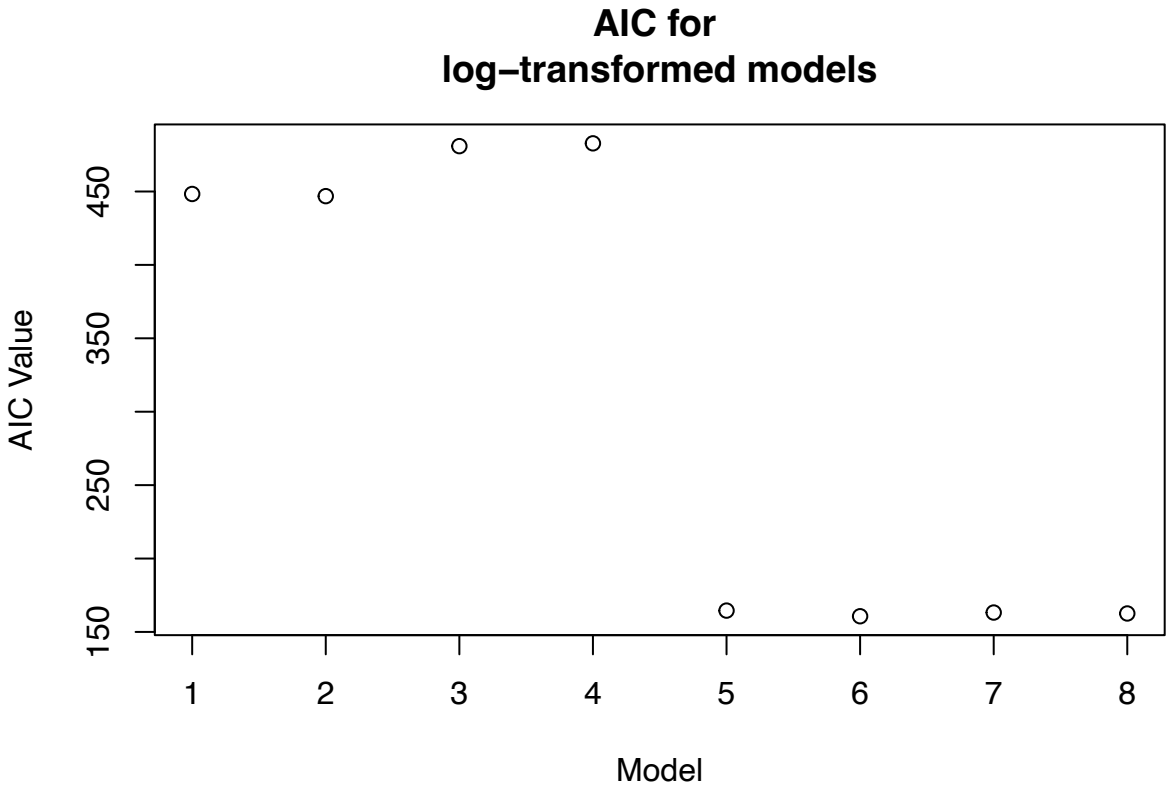
using the log-normal probability density function. In R, this is accounted for by using the `dlnorm()` function. Applying the above ensures all outputs are comparable.

```
# Set seed for reproducibility
set.seed(14)

# Conduct train-test-split
# N.B. Sample based on probability so may not generate an exact 80/20 split
sample_medals = sample(c(T,F),nrow(medals),replace=T,prob=c(0.8,0.2))
train_medals = medals[sample_medals,] # 21.1% of medals dataset
test_medals = medals[!sample_medals,] # 78.9% of medals dataset

# List model options
ltrans = c('Medal2012 ~ Population + GDP',
           'Medal2012 ~ log(Population) + GDP',
           'Medal2012 ~ Population + log(GDP)',
           'Medal2012 ~ log(Population) + log(GDP)',
           'log(Medal2012) ~ Population + GDP',
           'log(Medal2012) ~ log(Population) + GDP',
           'log(Medal2012) ~ Population + log(GDP)',
           'log(Medal2012) ~ log(Population) + log(GDP)')

# Create values for the loop below
pred_ll = numeric(length(ltrans))
aic_value = numeric(length(ltrans))

# Create loop to evaluate LL & AIC for each model
for (i in 1:length(ltrans)) {
  # Fit MLR with training data
  lmodel = glm(formula=ltrans[i],data=train_medals)

  # Differentiate between count and log outputs
  if (startsWith(ltrans[i],"log(Medal2012)")) {
    # Extract parameter estimates for mean (predictions) & sd
    x_hat_predict = predict(lmodel,test_medals)
    sigma_hat = sqrt(summary(lmodel)$dispersion)
```

```
    # Calculate predictive log-likelihood
    pred_ll[i] = sum(dlnorm(test_medals$Medal2012,
                            meanlog=x_hat_predict,
                            sdlog=sigma_hat,log=T))
  }
  else {
    x_hat_predict = predict(lmodel,test_medals)
    sigma_hat = sqrt(summary(lmodel)$dispersion)

    pred_ll[i] = sum(dnorm(test_medals$Medal2012,
                           mean=x_hat_predict,
                           sd=sigma_hat,log=T))
  }
  # Calculate AIC
  aic_value[i] = AIC(lmodel)
}

plot(1:length(ltrans),pred_ll,xlab='Model',ylab='Log Probability',
     main='Predictive log-likehood for \nlog-transformed models')
```




```
plot(1:length(ltrans),aic_value,xlab='Model',ylab='AIC Value',
     main='AIC for \nlog-transformed models')
```



Conclusion

Log-transformed (Y/N)	2012 Medal Counts	Population	GDP	Predictive Log-Likelihood	AIC
Model 1	N	N	N	Lower	Highest
Model 2	N	Y	N	Lower	Highest
Model 3	N	N	Y	Lowest	Highest
Model 4	N	Y	Y	Lowest	Highest
Model 5	Y	N	N	Highest	Lowest
Model 6	Y	Y	N	Highest	Lowest
Model 7	Y	N	Y	Highest	Lowest
Model 8	Y	Y	Y	Highest	Lowest

Using predictive log-likelihood and AIC as optimal model fit criterion, we can identify a number of models that are a more appropriate fit compared to the initial model `Medal2012 ~ Population + GDP` . Goodness of fit for models favours the highest predictive log-likelihood, implying more accurate predictions of the test data based on the given training set. Lower AIC values are also favoured, implying a more harmonious balance between model accuracy and complexity. This can be derived from the definition of AIC, given as

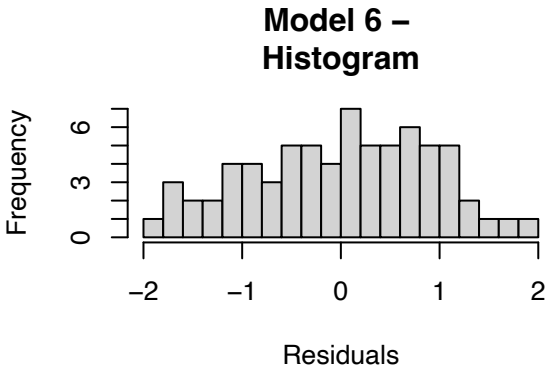
$$AIC = 2k - 2\log\mathcal{L}$$

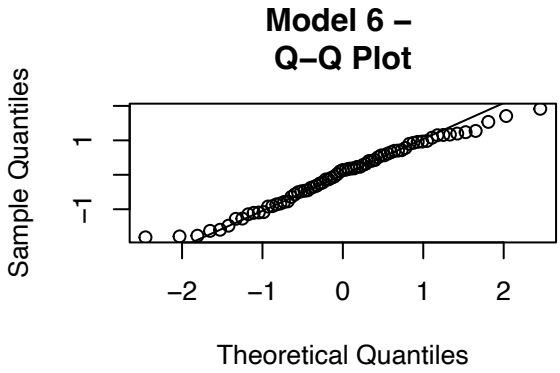
where $\log\mathcal{L}$ represents the log-likelihood (accuracy), and k represents the number of parameters in the model (complexity).

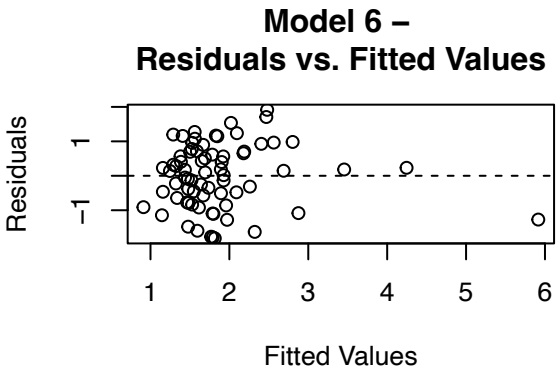
Given the above, any one of Models 5-8 could be considered. However **Model 6**, `log(Medal2012) ~ log(Population) + GDP`, has a slightly higher log-likelihood and slightly lower AIC value than all other models, proving to be the most appropriate model fit.

```r
# Model 6 is the best fit based on predictive log-likelihood and AIC
# Testing Normality & Homoscedasticity for comparison against model 1
model6 = glm(log(Medal2012) ~ log(Population) + GDP, data=medals)

par(mfrow=c(2,2))
hist(model6$residuals,breaks=20,xlab='Residuals',main='Model 6 - \nHistogram')
qqnorm(model6$residuals,main='Model 6 - \nQ-Q Plot')
qqline(model6$residuals)
plot(model6$fitted.values,model6$residuals,,xlab='Fitted Values',
     ylab='Residuals',main='Model 6 - \nResiduals vs. Fitted Values')
abline(h=0,lty='dashed')

summary(model6)
```

```
##
## Call:
## glm(formula = log(Medal2012) ~ log(Population) + GDP, data = medals)
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -1.449e+00  1.381e+00  -1.049 0.297922
## log(Population)  1.846e-01  8.406e-02   2.196 0.031518 *
## GDP              2.487e-04  6.216e-05   4.001 0.000158 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.8759891)
##
##     Null deviance: 96.505  on 70  degrees of freedom
## Residual deviance: 59.567  on 68  degrees of freedom
## AIC: 197.02
##
## Number of Fisher Scoring iterations: 2
```







For comparison with the initial model `Medal2012 ~ Population + GDP` , the normality and homoscedasticity assumption checks show better results. Whilst still not perfectly normally distributed, the residuals exhibit a more normal shape based on the histogram and Q-Q plots. More randomness and less clustering is seen in the residuals vs. fitted values plot.

The most appropriate model for 2012 Olympic medal counts based on population and GDP is:

$$\log(medals_X) = -1.449 + 0.1846 * \log(p_X) + 0.0002 * g_X + \epsilon_X.$$

Problem 2: UK Brexit vote

Logistic regression is a form of probabilistic modelling, predicting the probability of a binary output given any number of inputs. Using the `brexit` dataset, we want to predict the probability of an electoral ward voting to leave the European Union (`voteBrexit`) based on a subset of socioeconomic and demographic measures (`abc1`,`notBornUK`,`medianIncome`,`medianAge`,`withHigherEd`). This creates the multiple logistic regression equation

$$P(voteBrexit = TRUE \,|\, \mathbf{X}) = \phi(\beta_0 + \beta_1 x_1 + \ ... \ + \beta_5 x_5)\,,\ \phi(x) = \frac{1}{1+e^{-x}}$$

where $\mathbf{X} = (abc1, notBornUK, medianIncome, medianAge, withHigherEd)$ is the model inputs and $\phi(x)$ is the logistic function, characterised by its sigmoid curve.

Building the logistic regression model

```
brexit = read.csv('brexit.csv',header=T)

# is.logical(brexit$voteBrexit) = TRUE
# voteBrexit contains boolean data (TRUE/FALSE) unsuitable for modelling
# This needs to be converted to numerical data type (1/0)
brexit$voteBrexitNum = as.numeric(brexit$voteBrexit)
brexit_logreg = glm(voteBrexitNum ~ abc1 + notBornUK + medianIncome +
                    medianAge + withHigherEd,
                    data=brexit, family=binomial)
summary(brexit_logreg)
```

```
##
## Call:
## glm(formula = voteBrexitNum ~ abc1 + notBornUK + medianIncome +
##     medianAge + withHigherEd, family = binomial, data = brexit)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.1386     0.8477  -0.164 0.870122
## abc1          17.5780     2.9114   6.038 1.56e-09 ***
## notBornUK      5.6861     1.8033   3.153 0.001615 **
## medianIncome  -6.3857     1.9217  -3.323 0.000891 ***
## medianAge      5.9209     1.4066   4.209 2.56e-05 ***
## withHigherEd -26.7443     3.5762  -7.478 7.52e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 426.52  on 343  degrees of freedom
## Residual deviance: 247.39  on 338  degrees of freedom
## AIC: 259.39
##
## Number of Fisher Scoring iterations: 6
```

Interpretation of model inputs

What inputs are relevant for explaining model output?

Model inputs `abc1`, `medianIncome`, `medianAge` and `withHigherEd` are highly statistically significant with $Pr(>|z|) < 0.001$. This indicates a level of statistical certainty that the true coefficient values of these model inputs are not zero; there is a 0.1% probability of observing coefficients that are zero. Likewise, model input `NotBornUK` is statistically significant, but at a lower confidence level (99%). Before considering the effect each input has on the model, we can conclude that each of the five model inputs appear to be relevant influences on the UK population's decision to vote Brexit.

What inputs have strong effects?

Model Input	Rank	Absolute Coefficient	Model Influence	Context
withHigherEd	1	26.74	Negative	Strongly reduces $P(voteBrexit = TRUE)$
abc1	2	17.58	Positive	Strongly increases $P(voteBrexit = TRUE)$
medianIncome	3	6.39	Negative	Slightly reduces $P(voteBrexit = TRUE)$
medianAge	4	5.92	Positive	Slightly increases $P(voteBrexit = TRUE)$
notBornUK	5	5.69	Positive	Slightly increases $P(voteBrexit = TRUE)$

The magnitude/strength of model inputs should only consider the absolute coefficient value. Two inputs have a disproportionately larger effect on the model than the rest; `withHigherEd` (26.74) and `abc1` (17.58). Examining this in context, for example, `withHigherEd` has a magnitude approximately 4.5x larger than `MedianAge`. Therefore, the proportion of residents in an electoral ward with university-level education has a much stronger influence on the probability of a 'Leave' voting outcome, compared to the median age of the electoral ward.

Separate to magnitude, the direction of each coefficient is equally relevant. Both `withHigherEd` and `medianIncome` have negative direction, increasing the probability of a 'Remain' vote. In contrast, `abc1` , `medianAge` and `notBornUK` increase the probability of an electoral ward voting 'Leave'.

What factors could have an effect on the interpretability of any regression coefficients?

```
# Assessing multicollinearity using VIF
brexit_inputs = c('abc1','notBornUK','medianIncome','medianAge','withHigherEd')

# Create an empty vector with columns by each input (and model combination)
brexit_vif = setNames(rep(NA,length(brexit_inputs)),brexit_inputs)

for (i in 1:length(brexit_inputs)) {
  input = brexit_inputs[i]
  other_inputs = brexit_inputs[-i] # To create A ~ B + C + D + ...

  # Use as.formula() to not pass the expression as a string
  expression = as.formula(paste(input,"~",paste(other_inputs,collapse="+")))

  # Using lm() as glm() doesn't output R-Squared for VIF calculation
  model = lm(expression,data=brexit)
```

```
  brexit_vif[i] = round(1 / (1 - (summary(model)$r.squared)),4)
}
print(brexit_vif)
```

```
##         abc1    notBornUK medianIncome    medianAge withHigherEd
##       6.6895       3.6596       3.2137       2.3956       6.4169
```

```
# Using a correlation matrix to discover specific pairwise relationships
cor(brexit[, c("abc1", "withHigherEd", "medianIncome", "medianAge", "notBornUK")],
    use="complete.obs",method='pearson')
```

```
##                    abc1 withHigherEd medianIncome  medianAge  notBornUK
## abc1          1.0000000    0.8863129    0.7835105 -0.1662686  0.3947856
## withHigherEd  0.8863129    1.0000000    0.7412892 -0.2356651  0.5501448
## medianIncome  0.7835105    0.7412892    1.0000000 -0.3467963  0.5584333
## medianAge    -0.1662686   -0.2356651   -0.3467963  1.0000000 -0.7314555
## notBornUK     0.3947856    0.5501448    0.5584333 -0.7314555  1.0000000
```

Hidden relationships between variables can expose predictive inaccuracies and contribute to model overfitting. Both `abc1` and `withHigherEd` have VIF values greater than 5, suggesting there is multicollinearity present in the model. This explains why the coefficient magnitude for these inputs are so inflated compared to others, thus hindering the reliability of the model. Furthermore, we can see 4 pairwise combinations of inputs where $|p| > 0.7$, indicating strong correlation that should be addressed. The strongest relationship is `abc1` vs. `withHigherEd`; due to this being so strong (>0.88) the recommendation would be to remove one of these inputs as the predictive accuracy is unlikely to be majorly affected.

Additionally, there are factors relating specifically to the data itself which can affect the interpretability of the model:

- There is potential sampling bias in the `brexit` dataset. It contains normalised data based on the results from 344 electoral wards across the UK. However, the Office for National Statistics states that as of 2022 there are 8,483 electoral wards in the UK. Considering this is a subset of the entire population, there is the potential for bias based on attributes such as population density and location.
- The `brexit` dataset contains only five inputs available for the logistic regression model. There is no guarantee that the population measures featured are entirely representative of an optimal predictive model.

Conclusion

Without reassessing a number of different concerns, such as dimensionality reduction or further data exploration, it is not possible to reliably conclude both the relevance of each input and the optimal logistic model to use. This is because:

- There is multicollinearity in the current model, with a number of different inputs showing significant correlation.
- There is a lack of contextual information surrounding the reliability of the dataset used for modelling.

The ranking given above still holds relevance due to the statistical significance of each input. Accuracy is often a subjective measure; factors like multicollinearity do not prevent the model from making predictions. However, without address there are limitations to the reliability of each input's influence. Additionally, there are philosophical overlaps between inputs which create uncertainty. For example, we identified that `abc1`has an influence on model output much greater than `medianIncome`, but to what extent is income a decisive factor on social grade? The `abc1` social grade includes all members of the population that fall under the 'upper class' class hierarchy, which is indicative of income. We see this relationship through both input's strong correlation value, but this does not mean one input should be removed when both hold relevance.

Problem 3: Heart disease

The `heartdisease` dataset contains a number of clinical measures aiming to classify the presence of heart disease in an individual. We can build decision and random forest trees to explore this classification, comparing their predictive accuracy to select an optimal model.

Whilst there are different methods, the most common decision tree is the recursive method. This involves successively partitioning a dataset by minimising impurity, defined as

$$G = \sum_{i=1}^{n} p_i \left(1 - p_i\right)$$

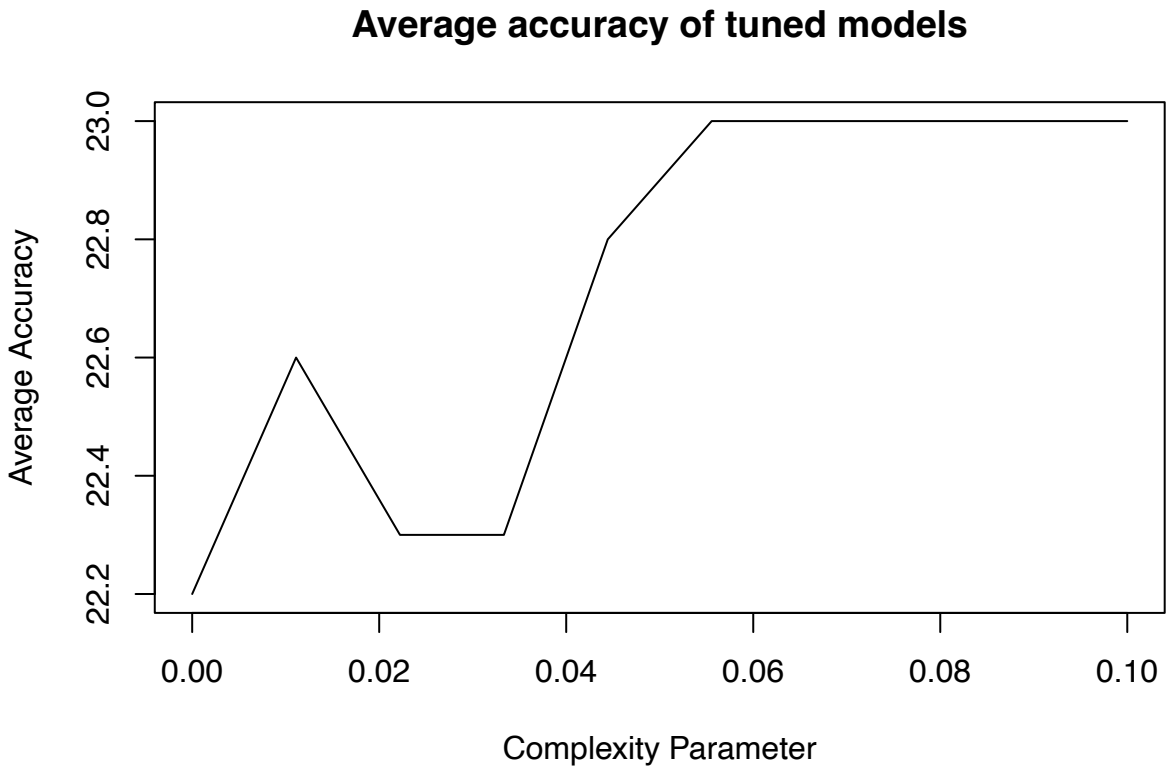
where G is the Gini Impurity, n is the number of classifications of the output variable, and p is the proportion of observations falling under given class. Contextually, $G = 0$ indicates the partition perfectly separates those with/without heart disease for a given input variable.

Random forest trees expand on this, building an ensemble of decision trees by randomly sampling both the dataset and model features; a process known as bootstrapping.

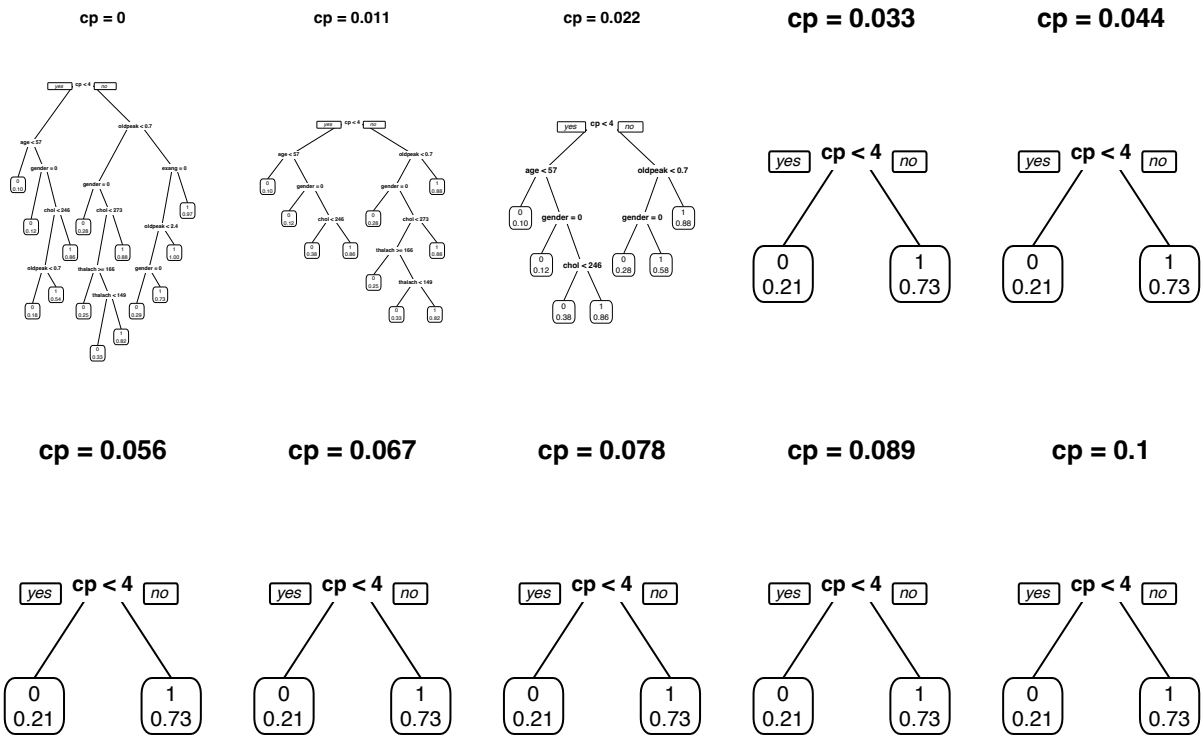
Building and tuning the recursive decision tree

```
heartdis = read.csv('heartdisease.csv',header=T)
heartdis$target = as.factor(heartdis$target) # This is important for random forest

# Loading rpart (Assumes rpart & rpart.plot are pre-installed)
library(rpart)
library(rpart.plot)

# Set seed for reproducibility
set.seed(19)

# Starting k-fold cross validation
complexity = seq(0,0.1,length.out=10)
k = 10 # Number of random train-test-split iterations (k-folds)
heartdis_size = nrow(heartdis)

folds = cut(seq(1,heartdis_size),breaks=k,labels=F)
accuracy = matrix(NA,nrow=k,ncol=length(complexity)) # Create accuracy matrix

# k-fold loop
for (i in 1:k) {
  recursive_test = which(folds==i)
  recursive_train = setdiff(1:heartdis_size,recursive_test)

  for (j in 1:length(complexity)) {
    recursive_model = rpart(target ~ age + gender + cp + trestbps + chol +
                              fbs + restecg + thalach + exang + oldpeak + slope,
                            data=heartdis[recursive_train,], # Only training set
                            method='class', # Classification
                            cp = complexity[j])

    # Use try() method to avoid failures with unseen data from training set
    recursive_pred = try(predict(recursive_model,
                                 heartdis[recursive_test,],
                                 type='class'),silent=T)
```

```
    # Calculate correct predictions
    if (!is(recursive_pred,'try-error')) {
      accuracy[i,j] = sum(recursive_pred == heartdis[recursive_test,'target'])
    }
  }
}

avg_accuracy = colMeans(accuracy,na.rm=T)
plot(complexity,avg_accuracy,type='l',
     ylab='Average Accuracy',xlab='Complexity Parameter',
     main='Average accuracy of tuned models')
```




```
par(mfrow = c(2, 5))
for (i in complexity) {
  recursive_tree = rpart(target ~ age + gender + cp + trestbps + chol + fbs +
                           restecg + thalach + exang + oldpeak + slope,
                         data=heartdis,
                         method='class',
                         cp = i)
  rpart.plot::prp(recursive_tree, extra=6,compress=T,
                  main=paste('cp =',round(i,3)))
}
```

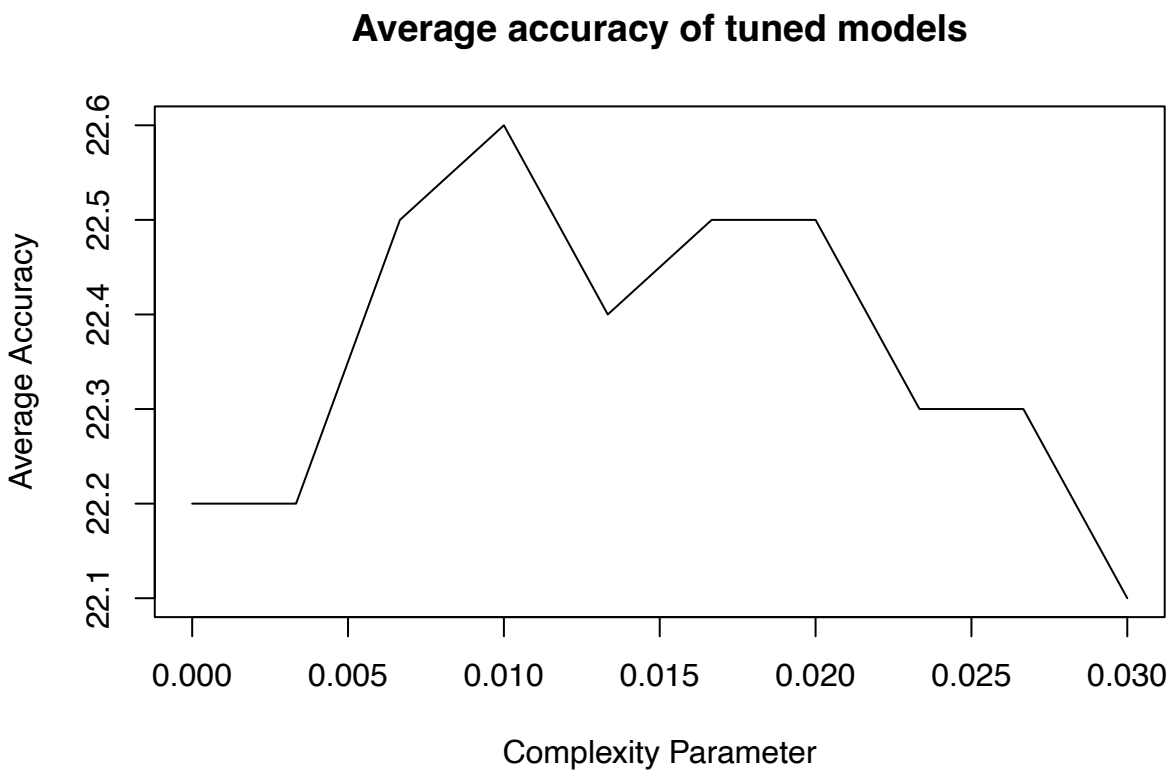


To evaluate the accuracy of a single recursive decision tree, we use k-fold cross validation to split the dataset into multiple, equal-sized training sets. We also use a tuning method (complexity parameter) to evaluate which tree is optimal. Accuracy is then calculated by the average number of correct classifications across each tuned model.

The complexity parameter (`cp`) is a punitive measure, requiring a split following each node to improve by a given factor. The initial training model was built using `cp` values between 0 and 0.1, with the model accuracy indicating that any `cp` value greater than 0.06 generates the highest accuracy. However, upon closer inspection we see that using a `cp` value greater than 0.03 causes overfitting so severe there is only one split.

```
complexity = seq(0,0.03,length.out=10)
k = 10
heartdis_size = nrow(heartdis)

folds = cut(seq(1,heartdis_size),breaks=k,labels=F)
accuracy = matrix(NA,nrow=k,ncol=length(complexity))

set.seed(23)

for (i in 1:k) {
  recursive_test = which(folds==i)
  recursive_train = setdiff(1:heartdis_size,recursive_test)

  for (j in 1:length(complexity)) {
    recursive_model = rpart(target ~ age + gender + cp + trestbps + chol +
                              fbs + restecg + thalach + exang + oldpeak + slope,
                            data=heartdis[recursive_train,],
                            method='class',
                            cp = complexity[j])

    recursive_pred = try(predict(recursive_model,
```

```
                                heartdis[recursive_test,],
                                type='class'),silent=T)

    if (!is(recursive_pred,'try-error')) {
      accuracy[i,j] = sum(recursive_pred == heartdis[recursive_test,'target'])
    }
  }
}

avg_accuracy = colMeans(accuracy,na.rm=T)
plotvalues = data.frame(round(complexity,4),avg_accuracy)
plot(complexity,avg_accuracy,type='l',
     ylab='Average Accuracy',xlab='Complexity Parameter',
     main='Average accuracy of tuned models')
```




```
recursive_final = rpart(target ~ age + gender + cp + trestbps + chol + fbs +
                          restecg + thalach + exang + oldpeak + slope,
                        data=heartdis,method='class',cp = 0.01)

final_predscore = plotvalues$avg_accuracy[plotvalues$round.complexity..4.==0.010]
training_size = round(nrow(heartdis)/k)

paste('Optimal cp-value: 0.01')
```

```
## [1] "Optimal cp-value: 0.01"
```

```
paste('Predictive performance:',final_predscore)
```

```
## [1] "Predictive performance: 22.6"
```

```r
paste('Accuracy (%):',round(final_predscore/training_size,4))
```

```
## [1] "Accuracy (%): 0.7533"
```

Re-running the same process with `cp` values between 0 and 0.03, we see more optimal results. Accuracy is highest at a `cp` value of 0.01, correctly identifying an average of 22.6 individuals. Considering we conducted 10-fold cross validation, and the `heartdisease` dataset has a total of 303 rows, there are 30 rows in each trained model, meaning average accuracy is $22.6/30 = 75.3\%$.

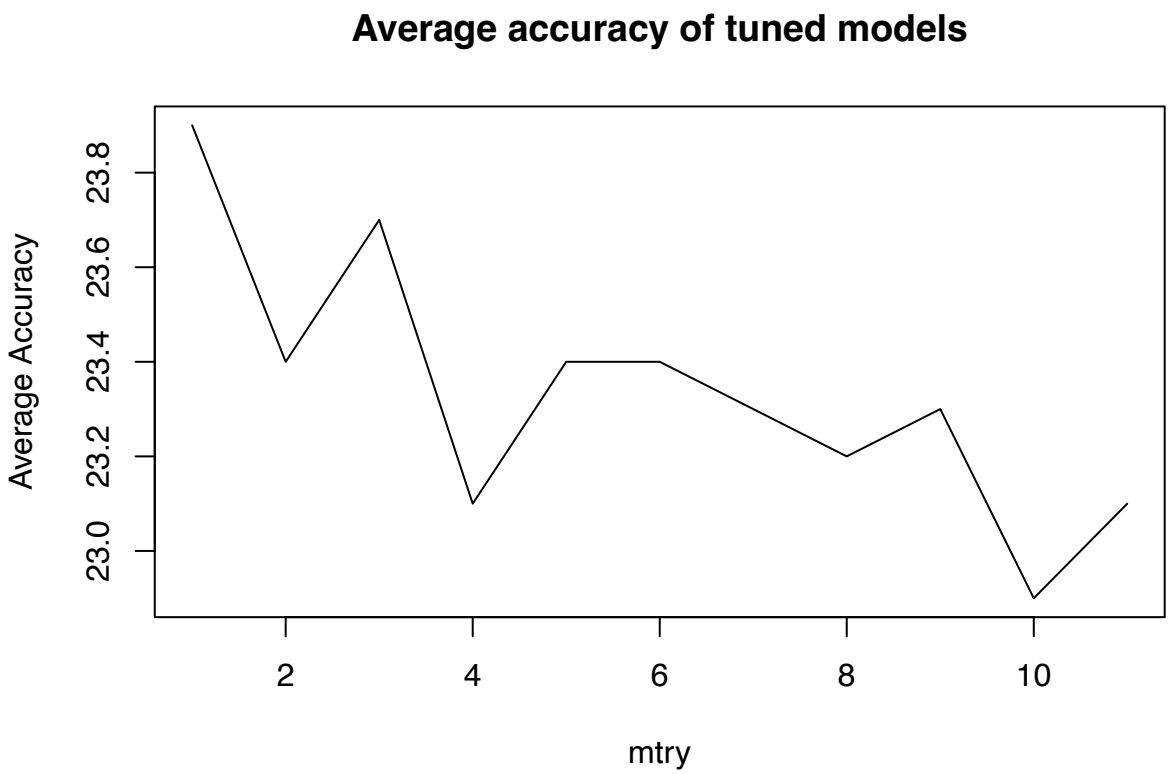
Building and tuning the random forest tree

```r
# Loading randomForest (assumes package is pre-installed)
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
# Set seed for reproducibility
set.seed(29)

# Using mtry for tuning considering all variables
tuning_var = 1:length(c('age','gender','cp','trestbps','chol','fbs','restecg',
                        'thalach','exang','oldpeak','slope'))
rf_accuracy = matrix(NA,nrow=k,ncol=length(tuning_var)) # Create accuracy matrix

# k-fold loop
for (i in 1:k) {
  rf_test = which(folds==i)
  rf_train = setdiff(1:heartdis_size,rf_test)

  for (j in 1:length(tuning_var)) {
    rf_model = randomForest(target ~ age + gender + cp + trestbps + chol +
                              fbs + restecg + thalach + exang + oldpeak + slope,
                            data=heartdis[rf_train,], # Only training set
                            mtry=tuning_var[j])

    # Use try() method to avoid failures with unseen data from training set
    rf_pred = try(predict(rf_model,
                          heartdis[rf_test,],
                          type='class'),silent=T)

    # Calculate correct predictions
    if (!is(rf_pred,'try-error')) {
      rf_accuracy[i,j] = sum(rf_pred == heartdis[rf_test,'target'])
    }
  }
}

avg_rf_acc = colMeans(rf_accuracy,na.rm=T)
plotvalues2 = data.frame(tuning_var,avg_rf_acc)
plot(tuning_var,avg_rf_acc,type='l',
     ylab='Average Accuracy',xlab='mtry',
     main='Average accuracy of tuned models')
```




```
final_predscore_rf = plotvalues2$avg_rf_acc[plotvalues2$tuning_var==1]

paste('Optimal mtry-value: 1')
```

```
## [1] "Optimal mtry-value: 1"
```

```
paste('Predictive performance:',final_predscore_rf)
```

```
## [1] "Predictive performance: 23.9"
```

```
paste('Accuracy (%):',round(final_predscore_rf/training_size,4))
```

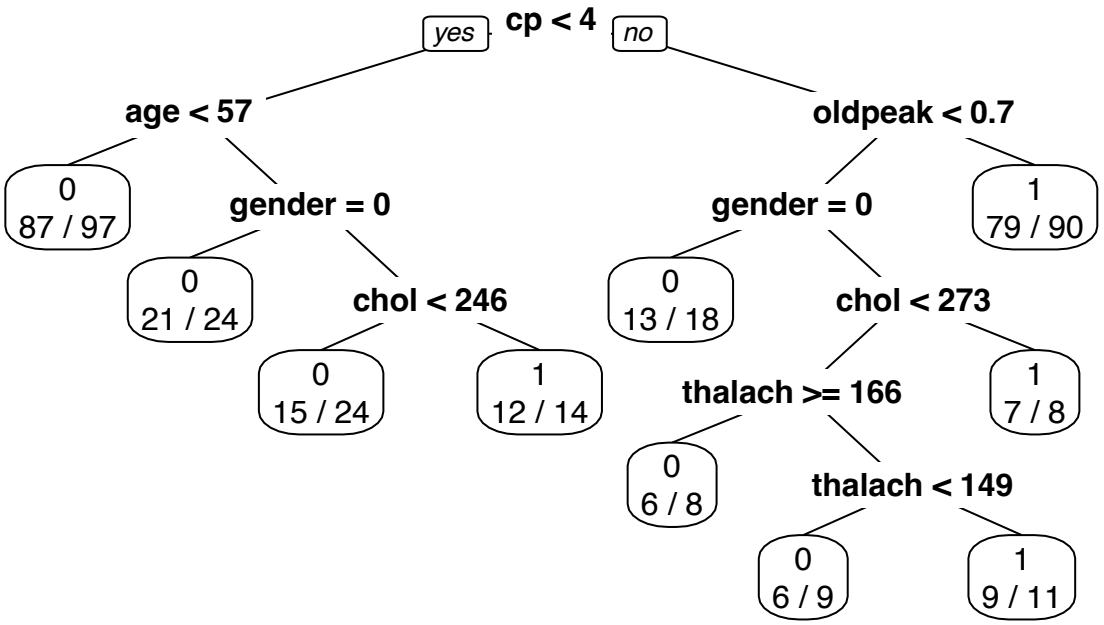
```
## [1] "Accuracy (%): 0.7967"
```

We adopt a similar approach for training the random forest tree, again using 10-fold cross validation. However, we use `mtry` as a tuning method. This defines how many model inputs are randomly sampled during bootstrapping. This is a key difference between the two model types: decision trees consider all inputs at each split, whereas random forest randomly assigns an input in order to prevent correlation and/or overfitting. Cross validation indicates that using an `mtry` value of 1 yields the highest accuracy based on training data, correctly identifying an average of 23.9 individuals. Interestingly, the low `mtry` value is an indication than one or more features predict heart disease very well, suggesting other model inputs may have little relevance to heart disease classification.

Conclusion

```
correct_preds = sum(predict(recursive_final,type='class')==heartdis$target)
acc_dt = round(100*(correct_preds/heartdis_size),2)

rf_final = randomForest(target ~ age + gender + cp + trestbps + chol +
                          fbs + restecg + thalach + exang + oldpeak + slope,
                        data=heartdis, mtry=1)
```

```
correct_preds_rf = sum(predict(rf_final,type='class')==heartdis$target)
acc_rf = round(100*(correct_preds_rf/heartdis_size),2)

rpart.plot::prp(recursive_final, extra=2,compress=T,
                main=paste('Final recursive decision tree:\nHighest optimal performance'))
```




```
conclusion = data.frame(
  model = c('decision tree','random forest'),
  correct_predictions = c(correct_preds,correct_preds_rf),
  model_accuracy = c(acc_dt,acc_rf)
)

print(conclusion)
```

```
##           model correct_predictions model_accuracy
## 1 decision tree                 255          84.16
## 2 random forest                 240          79.21
```

Fitting each model to the full `heartdisease` dataset, we see a change in behaviour. Out of 303 total observations, the final tuned decision tree shown above correctly classified 255 individuals, giving a final model accuracy of 84.16%. The random forest tree correctly identified only 78.55% of individuals, despite being more accurate than the decision tree using training data.

This is an expected outcome as recursive decision trees are prone to overfitting. Whilst this particular model achieved a high degree of accuracy, it is unlikely this model would fit well to new data. On the other hand, the random forest tree is more likely to adapt well to new data; at the expense of a small decrease in accuracy. Regardless, given that the random forest tree achieved the highest accuracy during training, we can conclude this is the optimal model.