ML Practical Notebook

Load Necessary Packages

```
# Utility
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# better plots
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```
#logistic regretion libraries
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.6.2
```

```
# Support Vector Libraries
library(nnet)
```

```
## Warning: package 'nnet' was built under R version 3.6.2
```

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

```r
# random forest
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

Load Data Into Dataframes

loading the larger dataset as training data that will be used to train the ML models prior to being tested using the smaller data set.

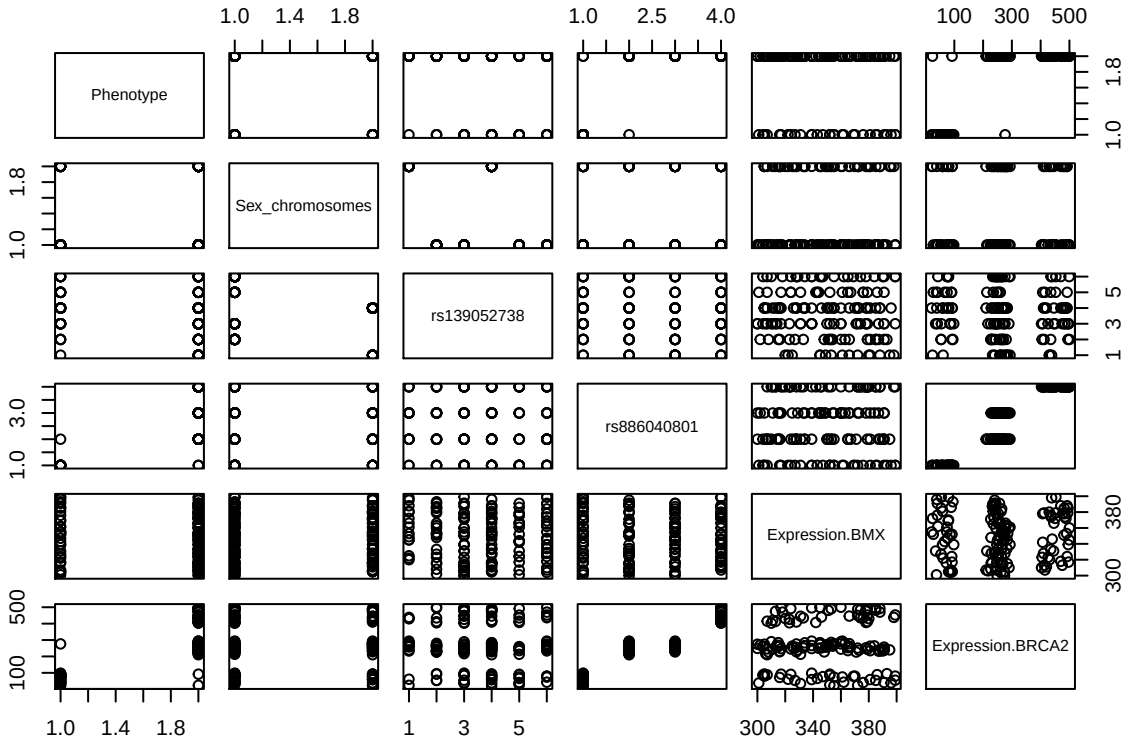
```r
# set seed for reproducibility
set.seed(123)

train.data <- read.delim('~/Downloads/assignment_data.tsv', sep = '\t', header = T)
predict.data <- read.delim('~/Downloads/predict_data.tsv', sep = '\t', header = T)
# the predddict dataset has no 'C' value in rs139052738 so we must make sure the levels are the same in
predict.data$rs139052738 <- factor(predict.data$rs139052738, levels = levels(train.data$rs139052738))
```

Visualise the Data

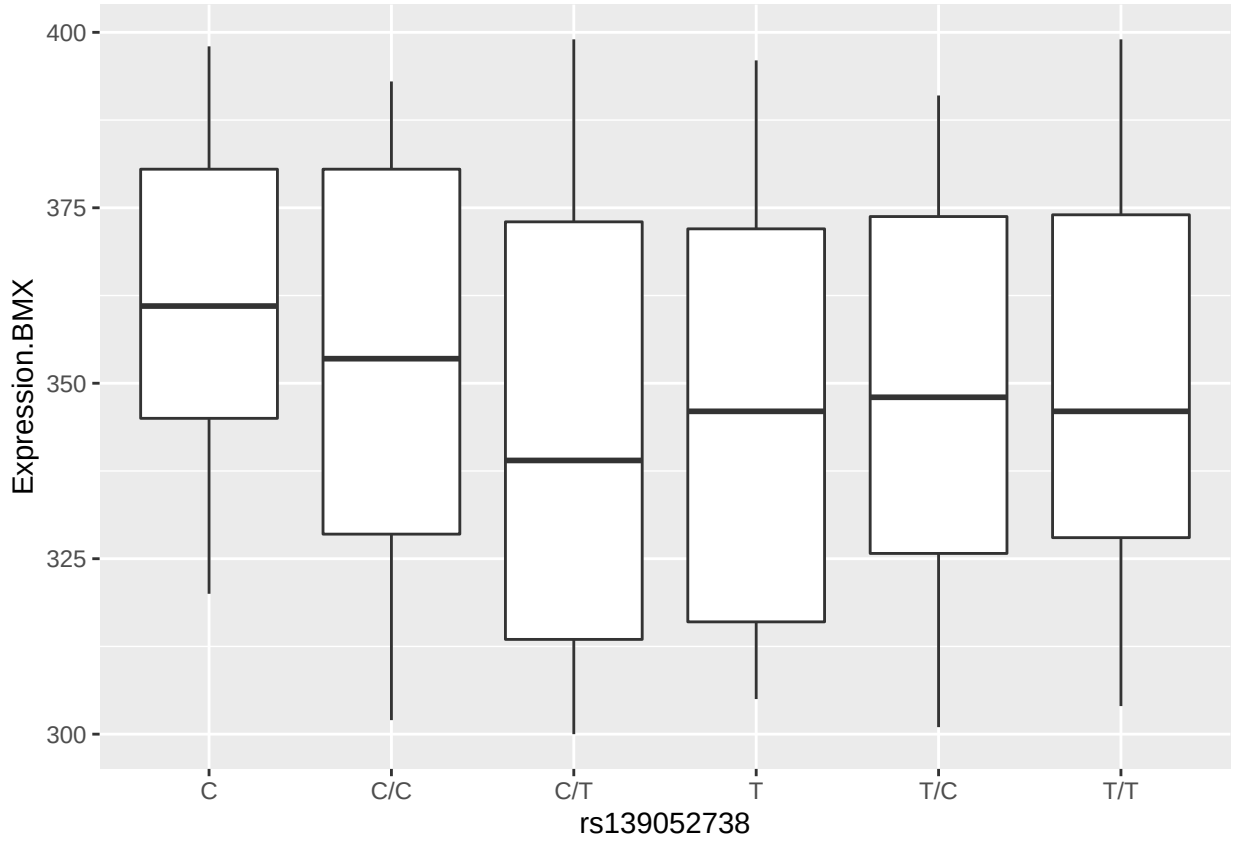
A few ways I visualize the data to get a grasp of the data included in the files starting by getting a broad view.

```r
pairs(train.data)
```

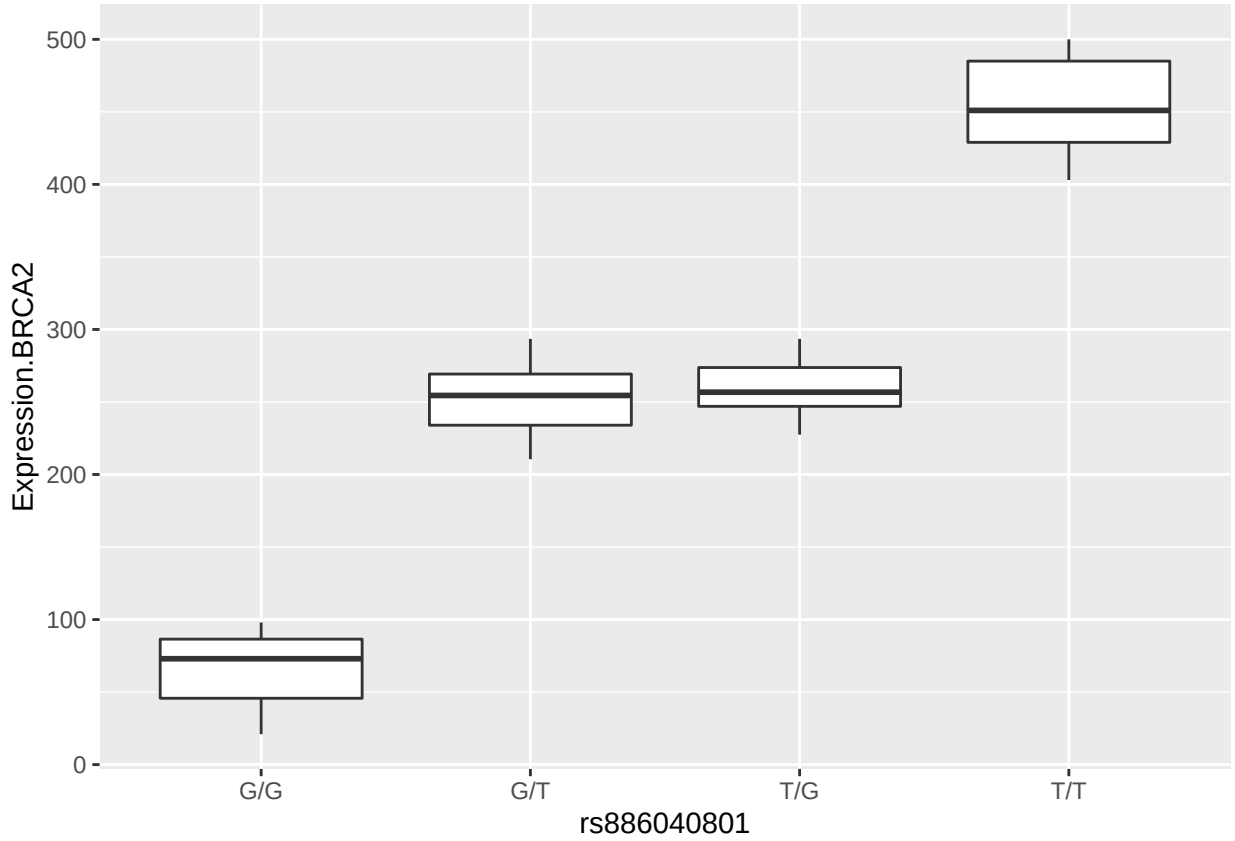


and now a few particular plots to explore some relationships between some of the features

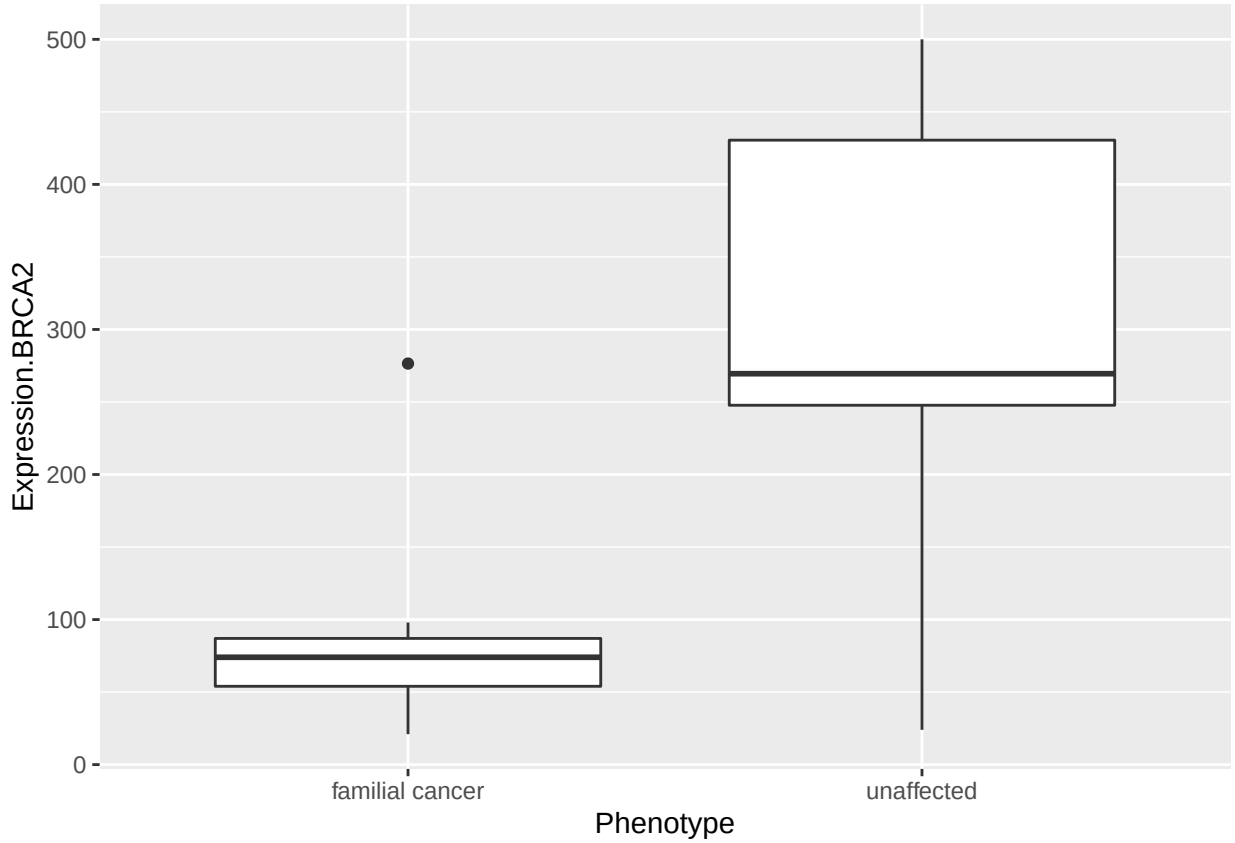
```
ggplot(train.data)+
  geom_boxplot(aes(rs139052738, Expression.BMX))
```



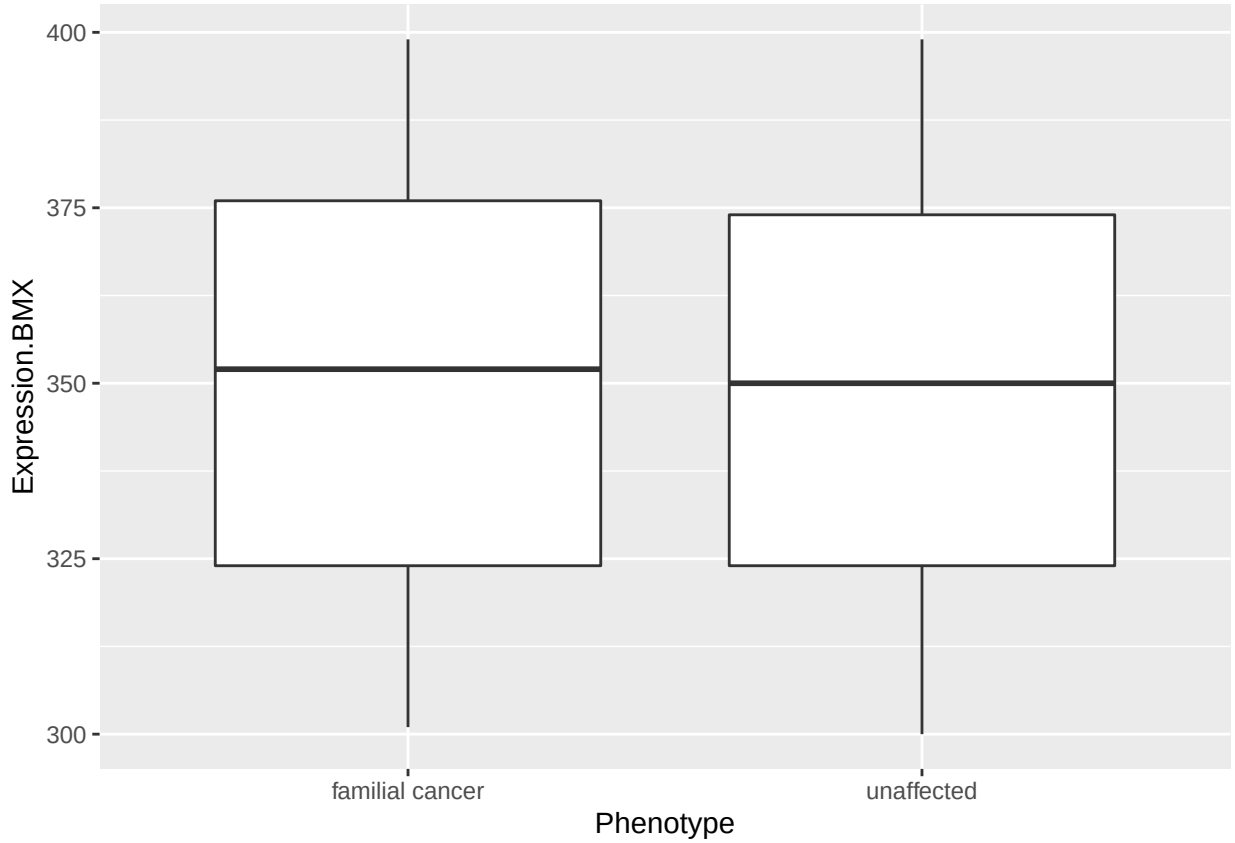

```
ggplot(train.data)+
  geom_boxplot(aes(rs886040801, Expression.BRCA2))
```



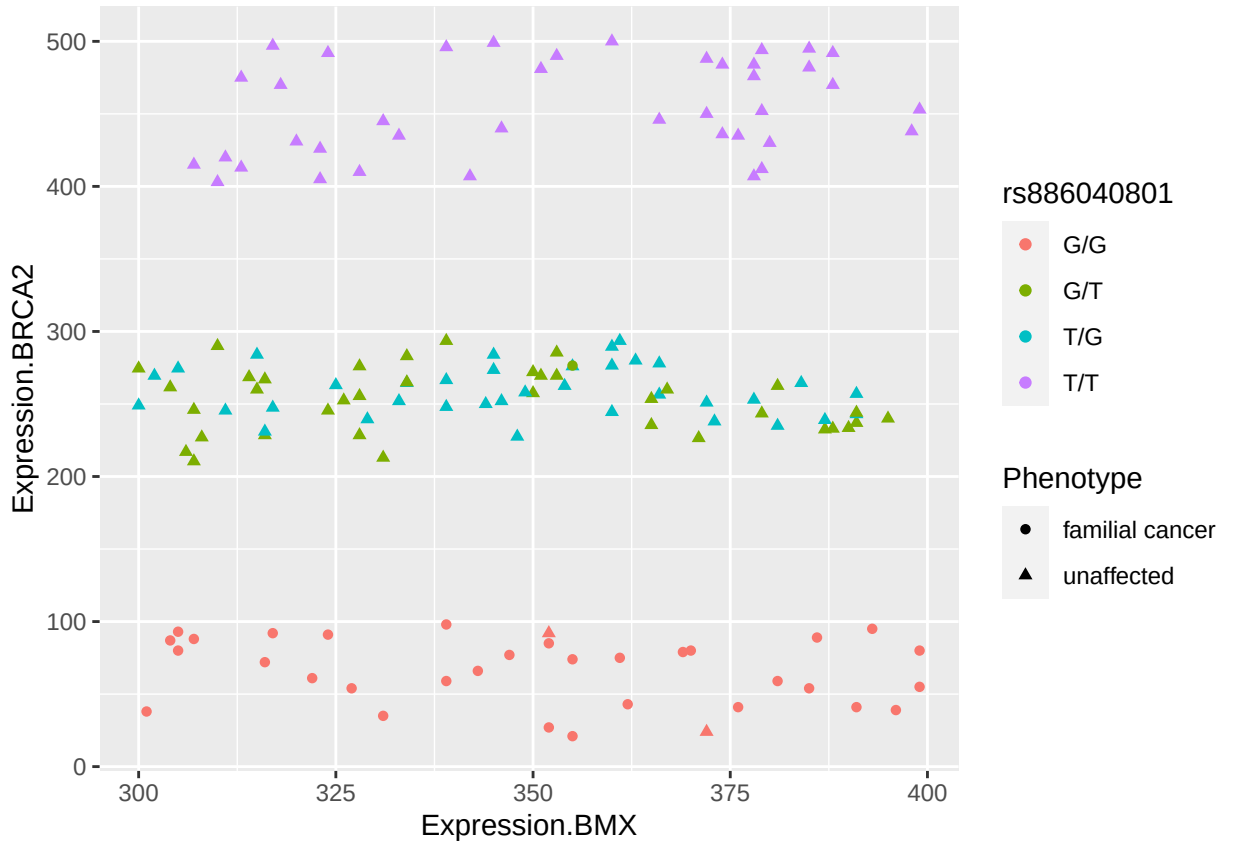

```
ggplot(train.data)+
  geom_boxplot(aes(Phenotype, Expression.BRCA2))
```




```
ggplot(train.data)+
  geom_boxplot(aes(Phenotype, Expression.BMX))
```




```
ggplot(train.data, aes(x=Expression.BMX, y=Expression.BRCA2, color=rs886040801, shape=Phenotype))+
  geom_point()
```



Support Vector Machine(SVM)

```
cancer.svm <- svm(Phenotype ~  ., data = train.data )
table(train.data$Phenotype, predict(cancer.svm, train.data, type='prob'))
```

```
##
##                   familial cancer unaffected
##   familial cancer              32          1
##   unaffected                    2        113
```

The above table shows that Using SVM will result in 3 miss-classifications in the training data, this is the worst result of all the methods tried here.

Neural Network

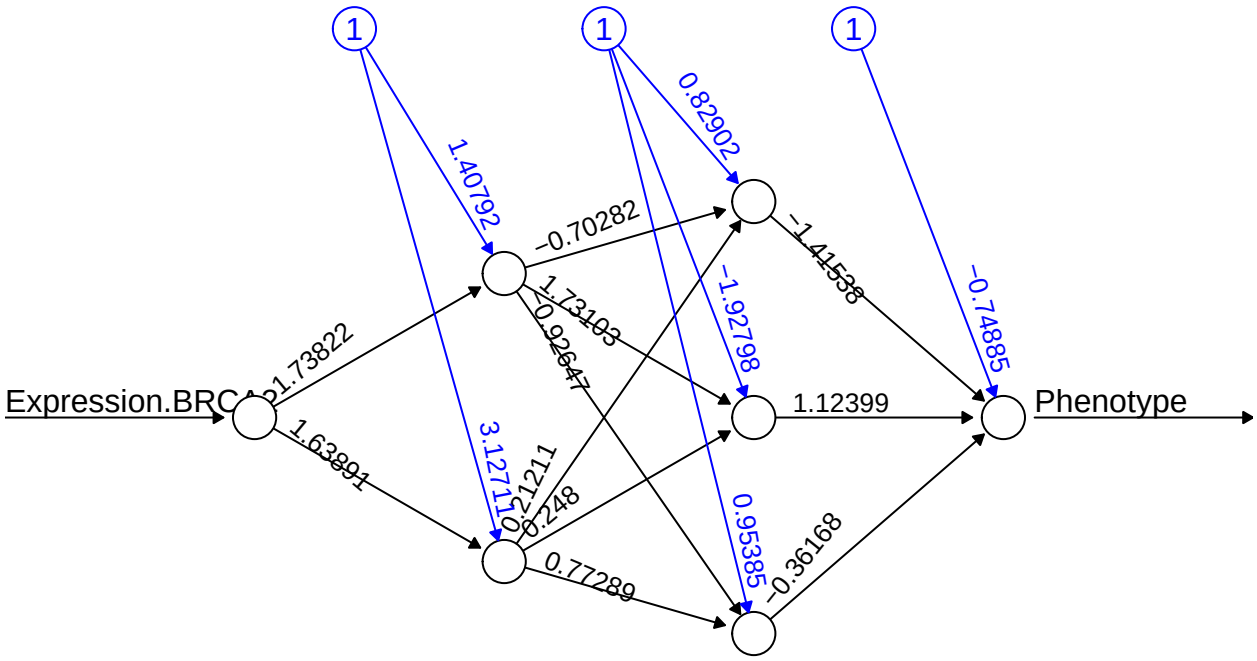
The neural network requires our response variable be numerical so I have converted the 'Phenotype' column to 1/0 instead of Familial Cancer/unaffected.

```
train.logical <-train.data %>% mutate(Phenotype = ifelse(Phenotype == 'familial cancer', 1,0))


net<-neuralnet(Phenotype~Expression.BRCA2, train.logical
```

```
                ,hidden=c(2,3), err.fct="ce", linear.output=FALSE, rep = 2)

plot(net, rep="best")
```



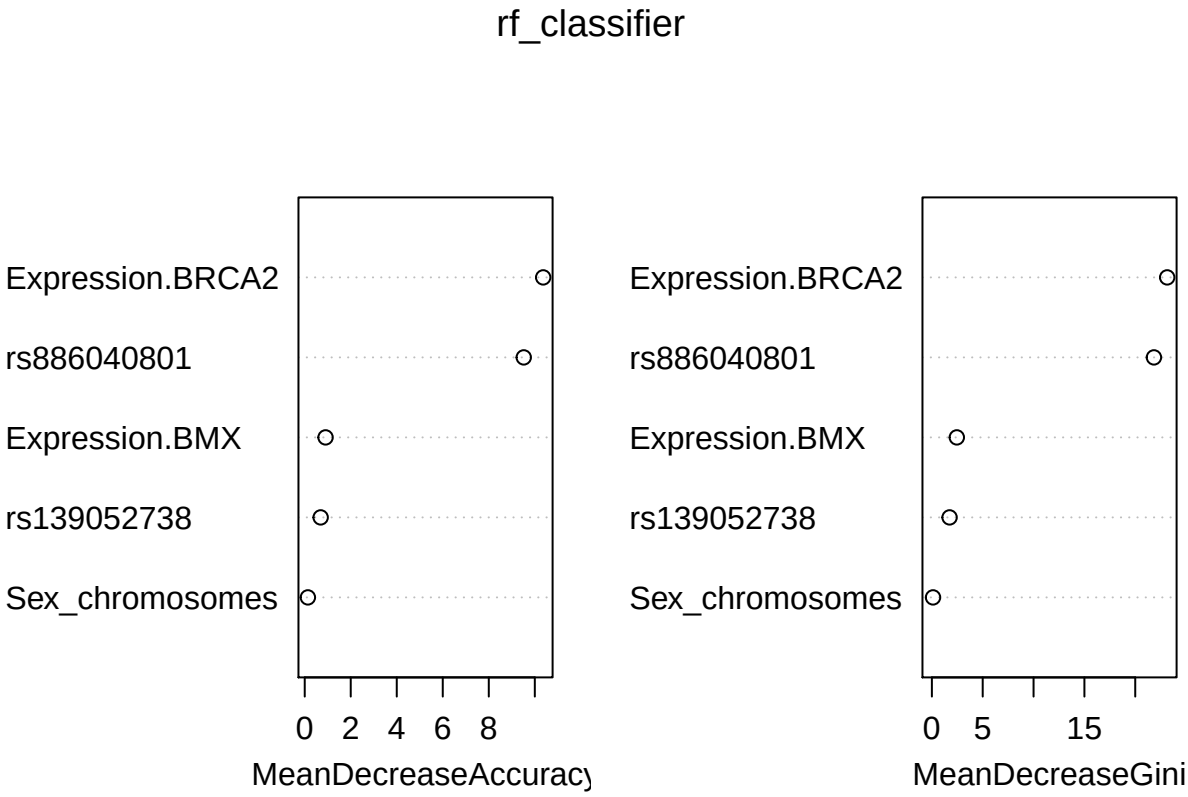

```
Predict <- neuralnet::compute(net, train.logical)
table(train.logical$Phenotype, round(Predict$net.result, 0))
```

```
##
##       0
##   0 115
##   1  33
```

The neural net most offten only gets 3 miss-classification with the training data. This is a good result however this method required a bit more tailoring by changing the number of nodes and hidden layers in the model to get these results.

random forest

```
rf_classifier = randomForest(Phenotype ~ Sex_chromosomes + rs139052738 + rs886040801 + Expression.BMX +
                             data=train.data, ntree=100, importance=TRUE)
varImpPlot(rf_classifier)
```




```
table(train.data$Phenotype, predict(rf_classifier, train.data, type='class'))
```

```
##
##                   familial cancer unaffected
##   familial cancer              32          1
##   unaffected                    1        114
```

The random forest method can get as low as 0 miss-classifications when trying to classify the training data, however, the benefit of using this method is that it takes comparatively less setup and can handle variables of various data types and weight their importance in classifying test data. This is why I have decided that this method has performed the best.

Making a Prediction

```
prediction_table <- predict(rf_classifier, predict.data, type = 'class')

data_frame(predict.data, prediction_table)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
## # A tibble: 8 x 6
```

```
##   Sex_chromosomes rs139052738 rs886040801 Expression.BMX Expression.BRCA2
##   <fct>           <fct>       <fct>                <int>            <int>
## 1 XX              C/C         G/T                    327              264
## 2 XX              T/T         T/G                    345              192
## 3 XX              T/C         G/G                    365               23
## 4 XX              T/C         T/T                    376              435
## 5 XX              C/T         G/T                    335              270
## 6 XY              T           T/G                    372              251
## 7 XX              T/C         G/G                    324               42
## 8 XX              C/T         G/G                    331               35
## # ... with 1 more variable: prediction_table <fct>
```

Question B

The first of the two steps in our typical NGS pipeline that could be improved by machine learning is variant annotation and prioritisation and classification. This could be done in order to provide predictions of the variant pathogenicity or clinical relevance even for novel mutations by looking at contextual information such as the location of the variant, is it a splice site, is it frequently associated with disease along with any other data that could be used to predict variant classification.

This has been done before in the form of LEAP (Learning from Evidence to Assess Pathogenicity) which provides a web application to aid scientists in a clinical reporting workflow(Carmen, et al., 2020)

The Second step in the NGS pipeline that could use machine learning is to Introduce a variant calling quality control step. Often sequencing errors can be incorrectly called as variants, particularly in sequencing samples with low depth. For example, If an error occurs in a read in a position with only 2x depth then it may be called as a heterozygous variant. Thorough 2x is an extreme example. Machine learning could be used to identify variants that are more likely to be sequencing errors and remove them before any clinical analysis. This could be done using ForestQC (Jiajin, et al., 2019). ForestQC uses a random forest model to identify possible variants that have been called during variant calling but may actually be sequencing errors.

References

Carmen, L., Anjali, D.Z., Robert, O., Serra, K., Ray, C., Jeroen, V.D.A, Alicia, Y.Z, Scott, T., and Gilad, M., 2020. LEAP: Using machine learning to support variant classification in a clinical setting. Human Mutation. [online] Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7317941/

Jiajin, L., Brandon, J., Lingyu, Z., Sungoo, H., Giovanni, C., Nelson B.F., Jae, H.S, 2019. ForestQC: Quality control on genetic variants from next-generation sequencing data using random forest. POLS Computational Biology [online] Available at: <https://journals.plos.org/ploscompbiol/article?id=10.1371/jour