HarvardX: PH125.9x: Liver Disease Classification

Teri Duffie

5/5/2020

1.Introduction

The purpose of this project is to build a classification model that predicts whether or not a patient has liver disease based on the demographic and chemical compound information provided within the Indian Liver Patient Records dataset aquired from kaggle. Most disease classification algorithms are primarily concerned with precision (when the model predicts disease, how often is it correct?) and sensitivity (out of all patients who truly have disease, how often does the model predict disease?). In the hypothetical scenario we describe later, the doctor has asked us to maximize precision. We will discuss this in detail prior to building the machine learning models.

Data

This dataset was downloaded to kaggle from the UCI Machine Learning Repository: Lichman, M. (2013) Irvine, CA: University of California, School of Information and Computer Science. https://www.kaggle.com/uciml/indian-liver-patient-records

The data contains 583 records for liver patients and non liver patients collected from North East of Andhra Pradesh, India. Age and sex information is provided for each record as well as measurements for 8 chemical compounds present in the body. Finally a "Dataset" column describes whether or not the patient actually has liver disease.

Key steps

- Data import, preprocessing, cleaning. The test set is split out initially and not used until model evaluation.
- Data exploration and visualization, noting insights.
- Feature selection and modeling approach.
- Model evalution and selection of final model. Once the final model is chosen we make predictions against the test dataset and review the model performance across multiple metrics.
- To conclude this report we summarize our findings and discuss limitations and potential improvements to the model.

2.Methods/Analysis

Load required packages and download data from GitHub

For the purpose of this project, the Indian Liver Patient Records dataset was uploaded to GitHub. The code provided here automatically downloads the dataset into the R environment.

```
# HarvardX: PH125.9x Data Science Capstone: Predicting Liver Disease - Final

#load necessary packages for analysis/modeling
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(reshape2)) install.packages("reshape2", repos = "http://cran.us.r-project.org")
if(!require(forecast)) install.packages("forecast", repos = "http://cran.us.r-project.org")
if(!require(janitor)) install.packages("janitor", repos = "http://cran.us.r-project.org")
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")
if(!require(gam)) install.packages("gam", repos = "http://cran.us.r-project.org")
if(!require(BiocManager)) install.packages("BiocManager", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(ggpubr)) install.packages("ggpubr", repos = "http://cran.us.r-project.org")
if(!require(WVPlots)) install.packages("WVPlots", repos = "http://cran.us.r-project.org")
if(!require(MLmetrics)) install.packages("MLmetrics", repos = "http://cran.us.r-project.org")


library(tidyverse)
library(caret)
library(data.table)
library(gridExtra)
library(knitr)
library(reshape2)
library(forecast)
library(janitor)
library(matrixStats)
library(gam)
library(randomForest)
library(ggpubr)
library(WVPlots)
library(MLmetrics)
library(BiocManager)


#download and read in indian_liver_patient from GitHub

url <- "https://raw.githubusercontent.com/tfitzg/CYO-Capstone-Indian-Liver-Disease/master/indian_liver_
liver_whole <- read_csv(url)
download.file(url, "indian_liver_patient.csv")

tempfile()
```

```
## [1] "C:\\Users\\terid\\AppData\\Local\\Temp\\RtmpoVTN1k\\file68fc74263057"
```

```r
tmp_filename <- tempfile()
download.file(url, tmp_filename)
liver_whole <- read_csv(tmp_filename)
file.remove(tmp_filename)
```

```
## [1] TRUE
```

Let's review the structure of the dataset to ensure successful download.

```
## tibble [583 x 11] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Age                       : num [1:583] 65 62 62 58 72 46 26 29 17 55 ...
##  $ Gender                    : chr [1:583] "Female" "Male" "Male" "Male" ...
##  $ Total_Bilirubin           : num [1:583] 0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.9 0.7 ...
##  $ Direct_Bilirubin          : num [1:583] 0.1 5.5 4.1 0.4 2 0.7 0.2 0.3 0.3 0.2 ...
##  $ Alkaline_Phosphotase      : num [1:583] 187 699 490 182 195 208 154 202 202 290 ...
##  $ Alamine_Aminotransferase  : num [1:583] 16 64 60 14 27 19 16 14 22 53 ...
##  $ Aspartate_Aminotransferase: num [1:583] 18 100 68 20 59 14 12 11 19 58 ...
##  $ Total_Protiens            : num [1:583] 6.8 7.5 7 6.8 7.3 7.6 7 6.7 7.4 6.8 ...
##  $ Albumin                   : num [1:583] 3.3 3.2 3.3 3.4 2.4 4.4 3.5 3.6 4.1 3.4 ...
##  $ Albumin_and_Globulin_Ratio: num [1:583] 0.9 0.74 0.89 1 0.4 1.3 1 1.1 1.2 1 ...
##  $ Dataset                   : num [1:583] 1 1 1 1 1 1 1 1 2 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Age = col_double(),
##   ..   Gender = col_character(),
##   ..   Total_Bilirubin = col_double(),
##   ..   Direct_Bilirubin = col_double(),
##   ..   Alkaline_Phosphotase = col_double(),
##   ..   Alamine_Aminotransferase = col_double(),
##   ..   Aspartate_Aminotransferase = col_double(),
##   ..   Total_Protiens = col_double(),
##   ..   Albumin = col_double(),
##   ..   Albumin_and_Globulin_Ratio = col_double(),
##   ..   Dataset = col_double()
##   .. )
```

The download was successful. We note that "Dataset" is a numeric variable and "Gender" is character type, and we know that for our purposes both of these should be converted to factors later on. The numeric variable type for the remainder of the columns seems correct.

Now we will break the data into two sets. One set will be used for exploration and model building. The second set ("test set") should be thought of as future data and will not be accessible to us until it is time to evaluate the final model. At that time, we will have to do the same preprocessing on the test set that we do on the train set (which for now is called "liver"). All data exploration and model training happens within the liver dataset (when we go to build models, this data is called the "train set").

We choose 20% for the proportion of data going to the test set. Since this dataset is relatively small, we want to ensure our final model is evaluated against a sample of reasonable size. The 80% of data remaining for training is later partioned again with the bootstrap approach so we don't want to push this lower than 80% initially.

```r
#covert to data frame from tibble for creating data partition
liver_whole <- data.frame(liver_whole)
```

```r
#for consistency in results, we set the seed.  We then partion the data such that 20% is in "test set"
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = liver_whole$Dataset, times = 1,
                                  p = 0.2, list = FALSE)
liver <- liver_whole[-test_index,]
test_set <- liver_whole[test_index,]
```

Visualization and Preprocessing

First we check out the first few rows of the dataset and see that it is already in tidy format, there is one record per row.

```
##   Age Gender Total_Bilirubin Direct_Bilirubin Alkaline_Phosphotase
## 1  65 Female             0.7              0.1                  187
## 2  62   Male            10.9              5.5                  699
## 3  62   Male             7.3              4.1                  490
## 4  58   Male             1.0              0.4                  182
## 5  72   Male             3.9              2.0                  195
## 6  46   Male             1.8              0.7                  208
##   Alamine_Aminotransferase Aspartate_Aminotransferase Total_Protiens Albumin
## 1                       16                         18            6.8     3.3
## 2                       64                        100            7.5     3.2
## 3                       60                         68            7.0     3.3
## 4                       14                         20            6.8     3.4
## 5                       27                         59            7.3     2.4
## 6                       19                         14            7.6     4.4
##   Albumin_and_Globulin_Ratio Dataset
## 1                       0.90       1
## 2                       0.74       1
## 3                       0.89       1
## 4                       1.00       1
## 5                       0.40       1
## 6                       1.30       1
```

Now let's determine if there are missing values.

```r
any(is.na(liver))
```

```
## [1] TRUE
```

There are in fact missing values. But where?

```
##     Age Gender Total_Bilirubin Direct_Bilirubin Alkaline_Phosphotase
## 210  45 Female             0.9              0.3                  189
## 242  51   Male             0.8              0.2                  230
## 254  35 Female             0.6              0.2                  180
## 313  27   Male             1.3              0.6                  106
##     Alamine_Aminotransferase Aspartate_Aminotransferase Total_Protiens Albumin
## 210                       23                         33            6.6     3.9
## 242                       24                         46            6.5     3.1
## 254                        12                               15             5.2     2.7
## 313                        25                               54             8.5     4.8
##     Albumin_and_Globulin_Ratio Dataset
## 210                         NA       1
## 242                         NA       1
## 254                         NA       2
## 313                         NA       2
```

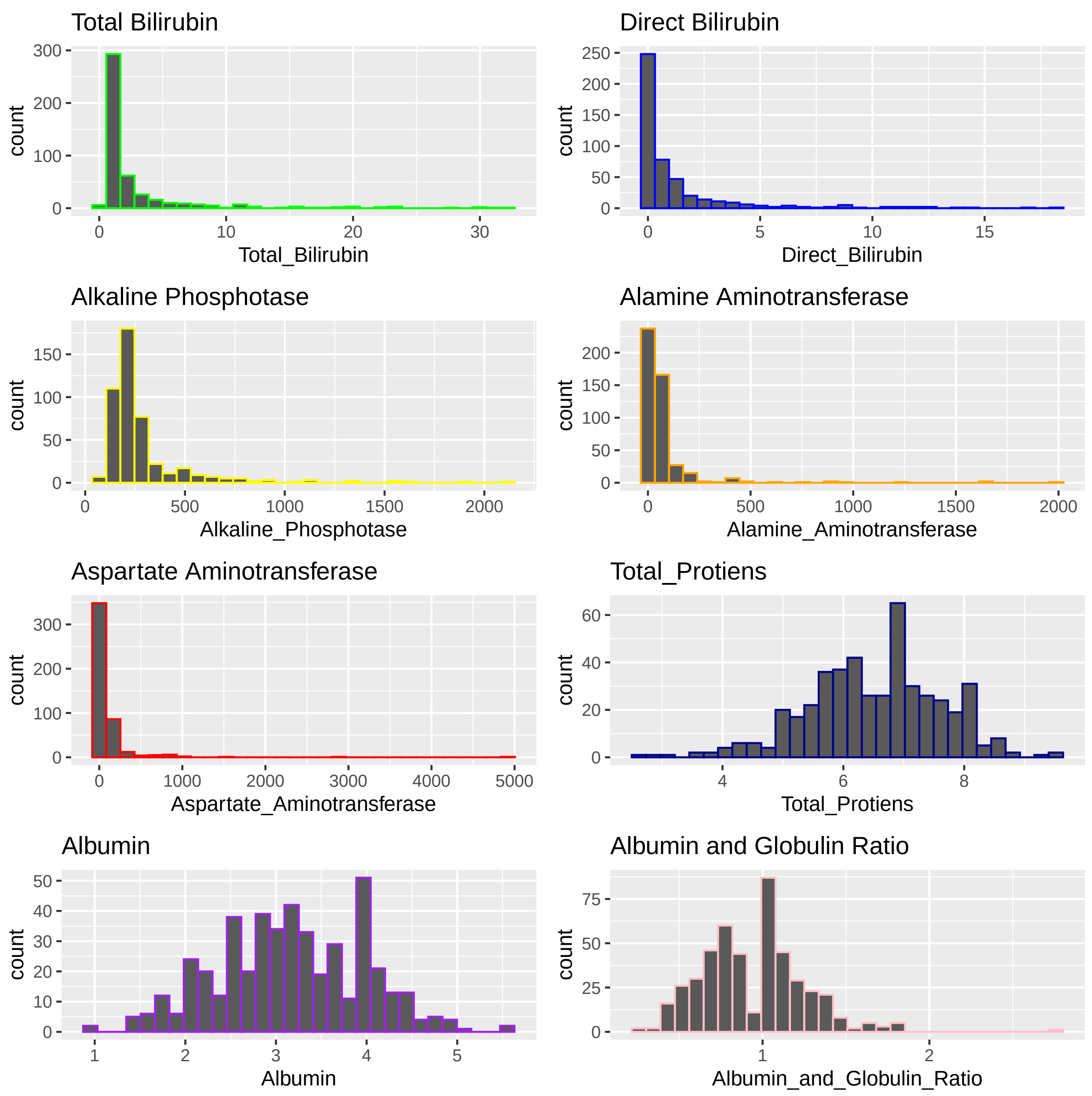
We see that there are 4 missing values and all are in the Albumin and Globulin Ratio column. Running a quick summary of this column (below), we see that the 1st through 3rd quartile ranges from 0.7 to 1.1 which is quite tight. We make the decision to replace the missing values with the mean value (0.96) and move on.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##  0.3000  0.7000  0.9600  0.9436  1.1000  2.8000       4
```

At this point let's change the "Dataset" column to a "Disease" column which indicates disease status and change Disease and Gender to factors.

```
#add factor Disease for clarity, remove "Dataset and change Gender to factor"
liver <- liver %>%
  mutate(Disease = as.factor(ifelse(Dataset == "1", "Disease", "NoDisease")),
         Gender = as.factor(Gender)) %>%
  select(-Dataset)
```

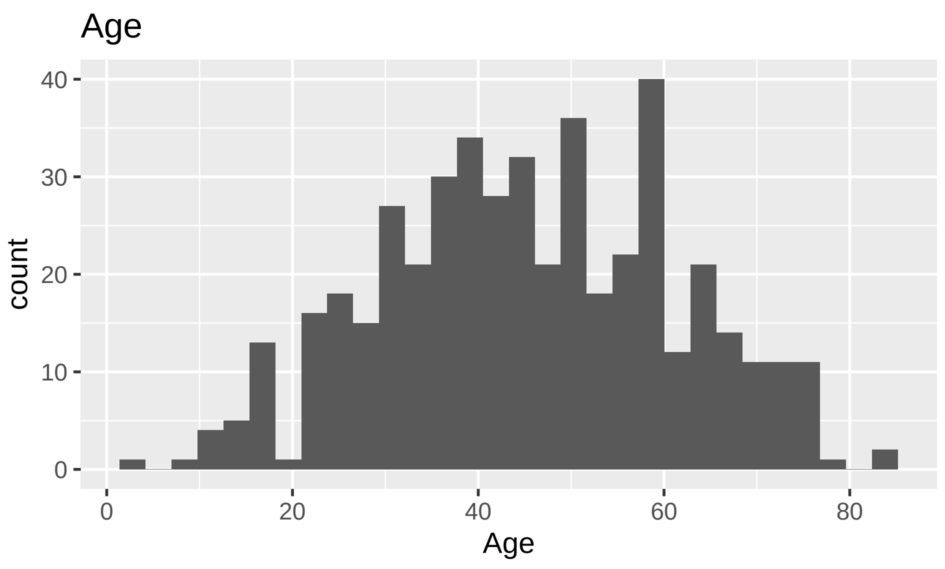
Now we're going to review histograms for the numeric variables. In doing this, we can accomplish a few things: we can visualize pontential outliers, we can view the shape of the distributions, and we can get a sense of the variance for each potential feature. Let's start with the chemical compounds.



We observe that Total Bilirubin, Direct Bilirubin, Alkaline Phosphotase, Alamine Aminotransferase and Aspartate Aminotransferage are positively skewed and possibly contain outliers (as the x-axis includes values much higher than where the majority of the distribution falls). Total Protiens, Albumin and Albumin-Globulin Ratio have distributions that appear closer to normal and no points we feel confident in excluding as outliers. We can see the general range and variance for all compounds.

Let's also understand the ages represented within the dataset and test for normality.

```
liver %>% ggplot(aes(Age)) +
  geom_histogram(bins = 30) +
  ggtitle("Age")
```



```r
shapiro.test(liver$Age)
```

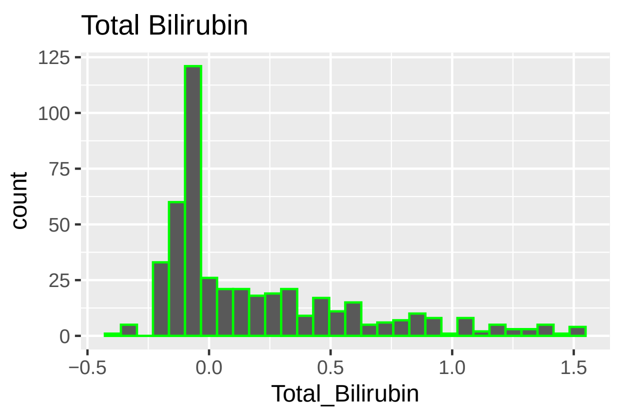
```
##
##  Shapiro-Wilk normality test
##
## data:  liver$Age
## W = 0.9908, p-value = 0.005289
```

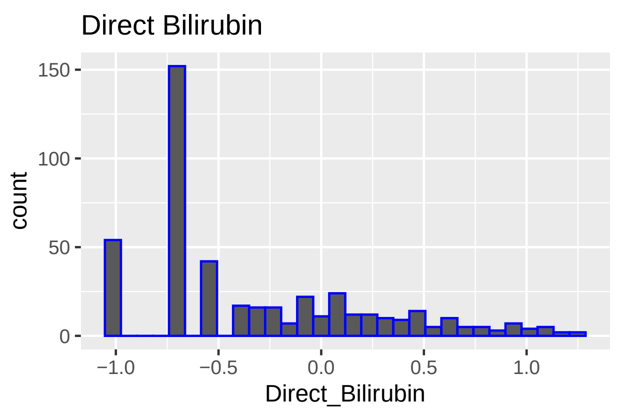
The ages are all reasonable (no bad data/outliers) and the distribution is reasonably close to normal, although the shapiro test for normality (p value < 0.05) shows that it is not in fact normal.

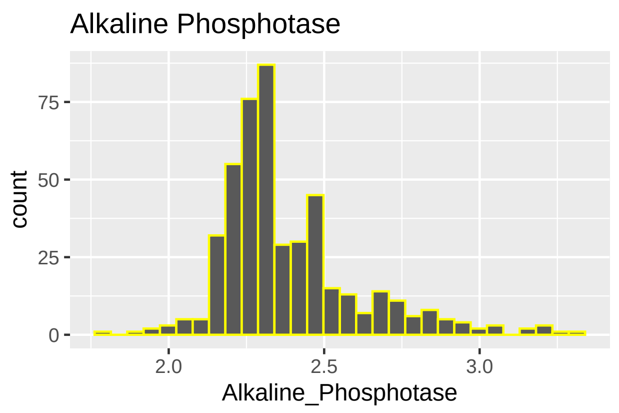
We make the decision to leave the normal-ish data as is, but since there is an assumption of normality in some models we may want to build and the possibily that normal data would help build better models, we are going to transform the variables with very skewed distributions. The log10 transformation is chosen from the few we investigated as it pushes most of the variable distributions closer to normal.

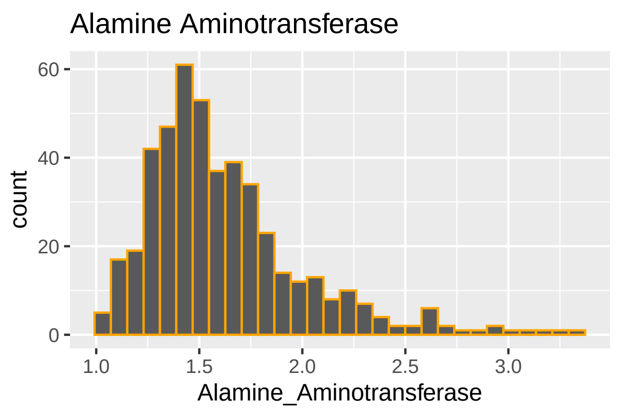
```r
#formula loop for tranformations
columns <- liver[, 3:7]
trans_cols <- sapply(columns, function(t) {
  transf <- log10(t)
})

trans_cols <- as.data.frame(trans_cols)
```

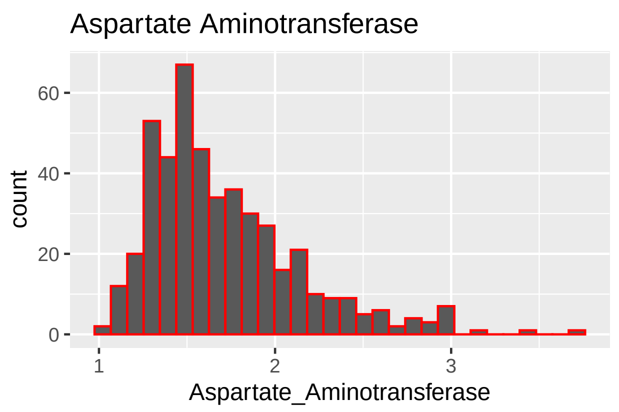
Let's review histograms of the transformed variables.



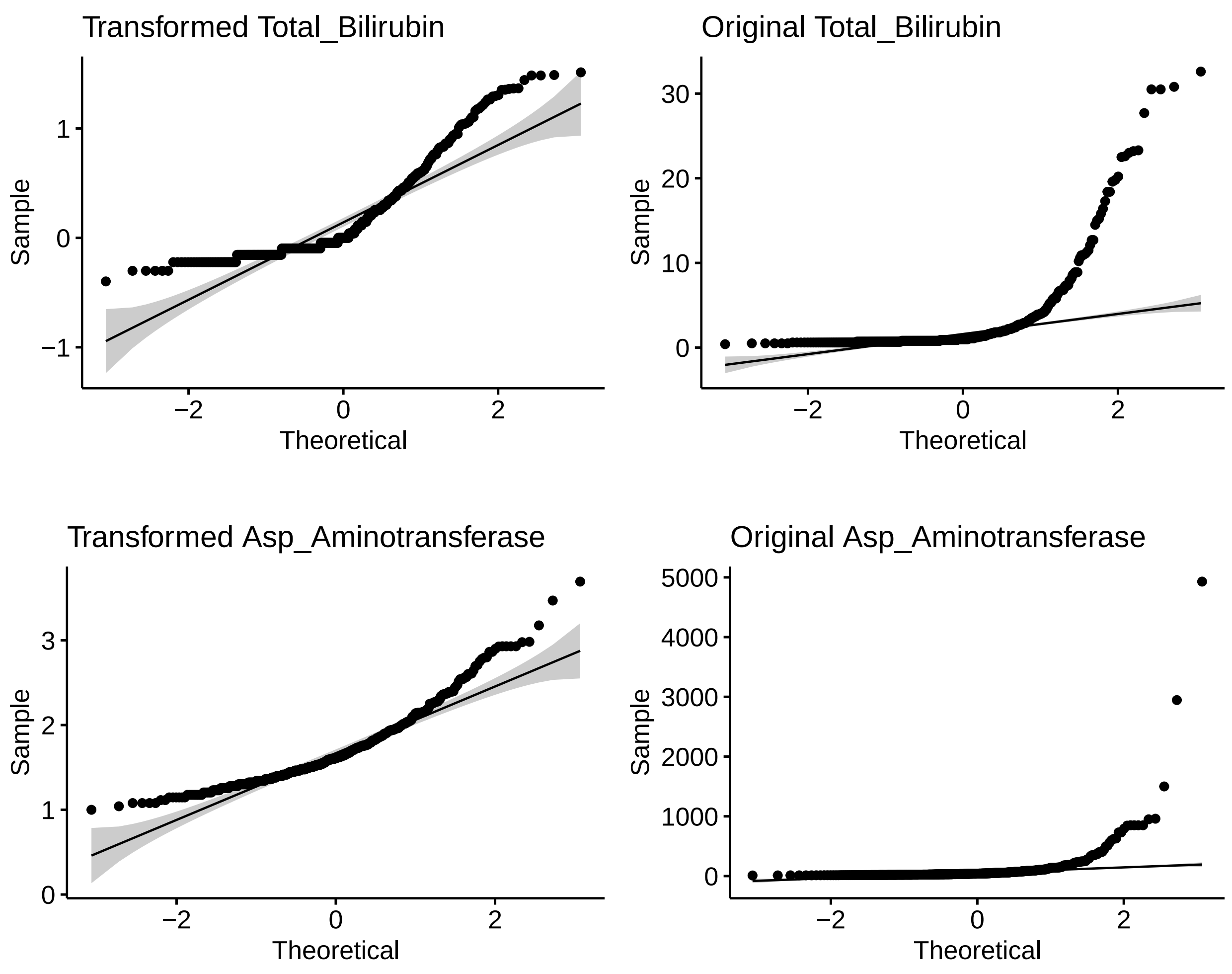








We can see that we weren't totally successful in achieving normality with the transformation, and it's further confirmed by checking the qqplots. Here are a couple of examples.

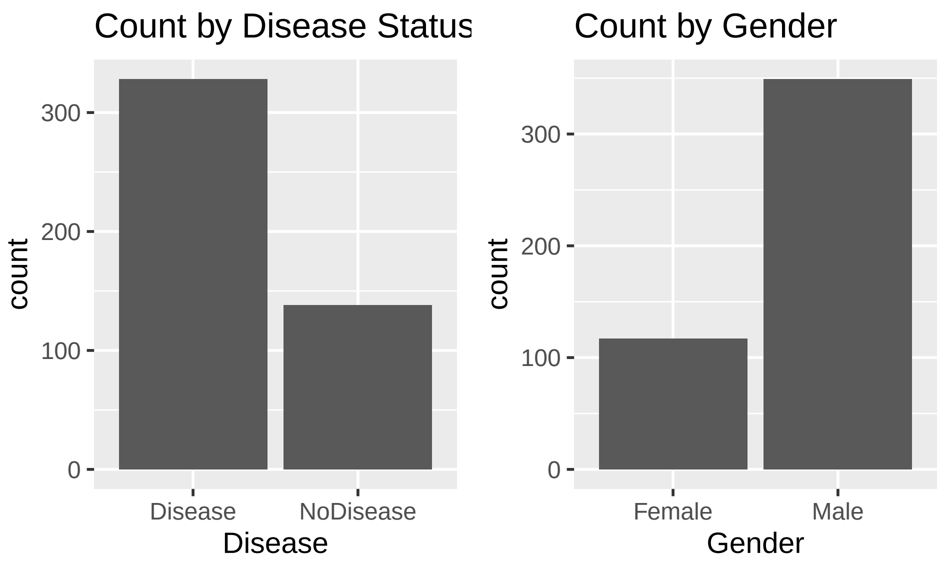


However, the distributions are certainly closer to normal than they were previously, and we no longer see obvious outliers. Since we don't totally understand the details of the data and can't confirm incorrect inputs, we really did not want to exclude any data. We'll count the tranformation as a win, and move on by updating the table with the transformed data. The updated data frame is called "new_liver".

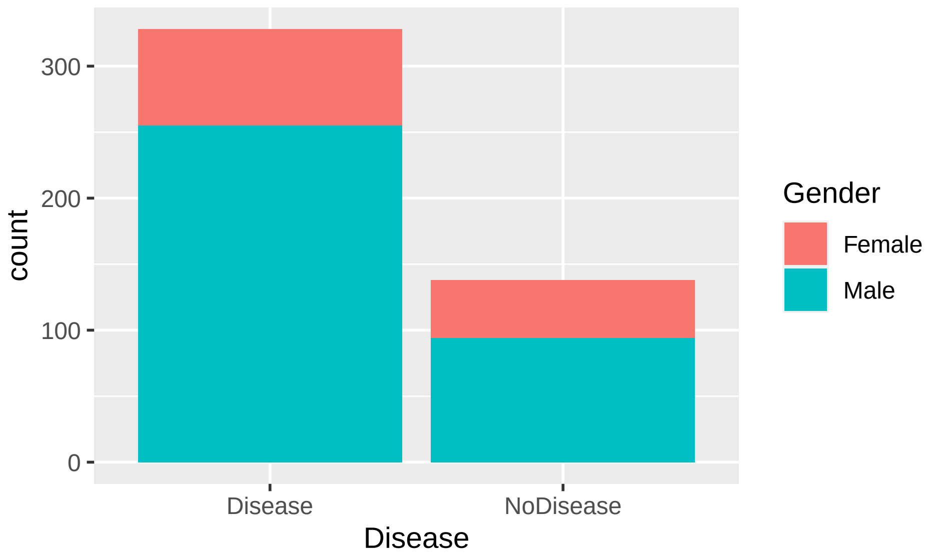
```
#new_liver is the liver dataframe updated with transformed columns
new_liver <- liver %>% mutate(Total_Bilirubin = trans_cols$Total_Bilirubin,
                              Direct_Bilirubin = trans_cols$Direct_Bilirubin,
                              Alkaline_Phosphotase = trans_cols$Alkaline_Phosphotase,
                              Alamine_Aminotransferase = trans_cols$Alamine_Aminotransferase,
                              Aspartate_Aminotransferase = trans_cols$Aspartate_Aminotransferase)
```

Now that we're happy with the state of the data, we can do some additional visualization.

Let's begin by reviewing the breakdown on patient records by disease status as well as the number of records based on Gender. We can see that there are many more patients with liver disease in the dataset than without, and note that prevalence will be a consideration when building/evaluating our models. We also see that there are far more males in the dataset than females.

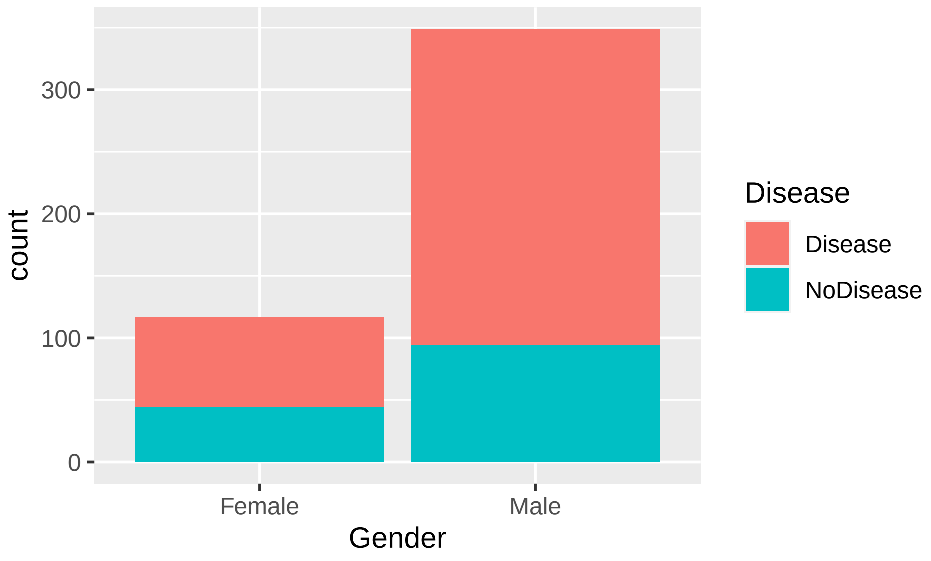


If we were to look at only the records with disease, we might be inclinded to think that the disease occurs far more often in males:



Disease	Female	Male
Has Disease	0.223	0.777

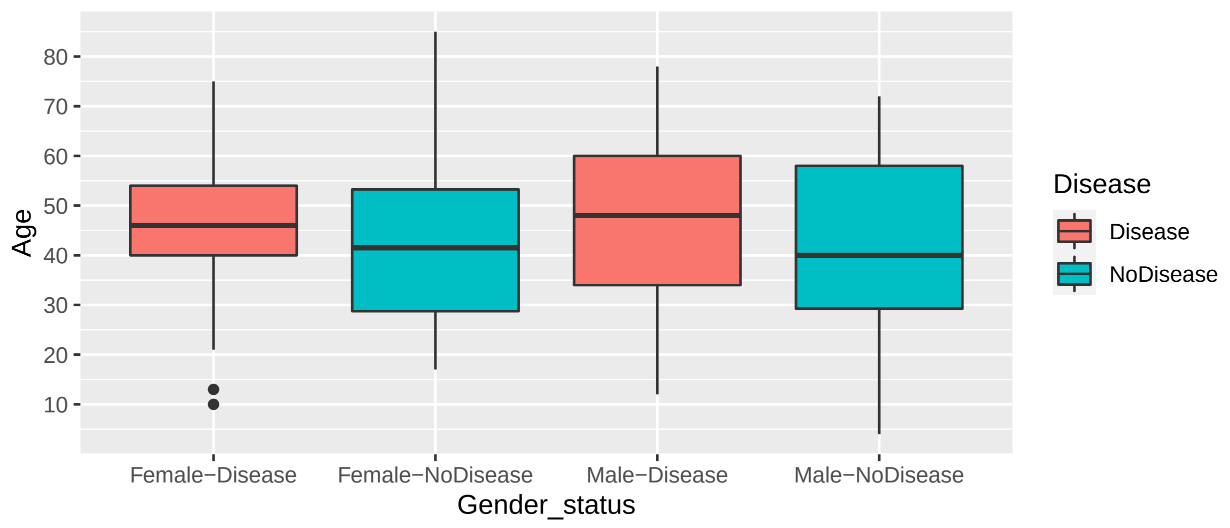
But actually, females in this dataset are also more likely to have the disease than not. We can see that the male percentage with disease is ~10% higher than female and we'll look into whether this is statistically significant later on.



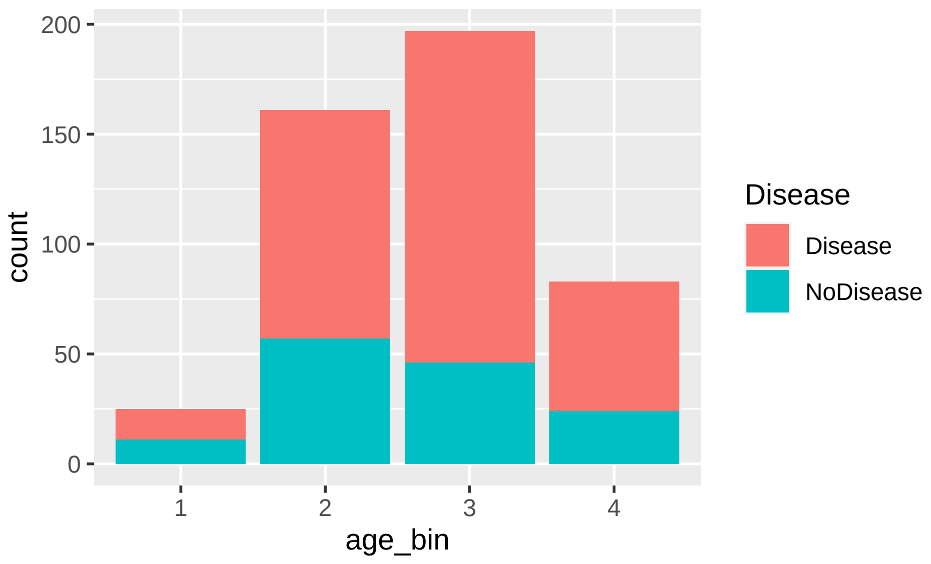
Gender	Disease	Frequency
Female	Disease	73
Female	NoDisease	44
Male	Disease	255
Male	NoDisease	94

Gender	Proportion_with_Disease
Female	0.624
Male	0.731

If we break this down a bit further by including age, we can see that the patients with liver disease tend to be slightly older (at the midpoint) than those without liver disease, although there is a lot of overlap in the boxplots.



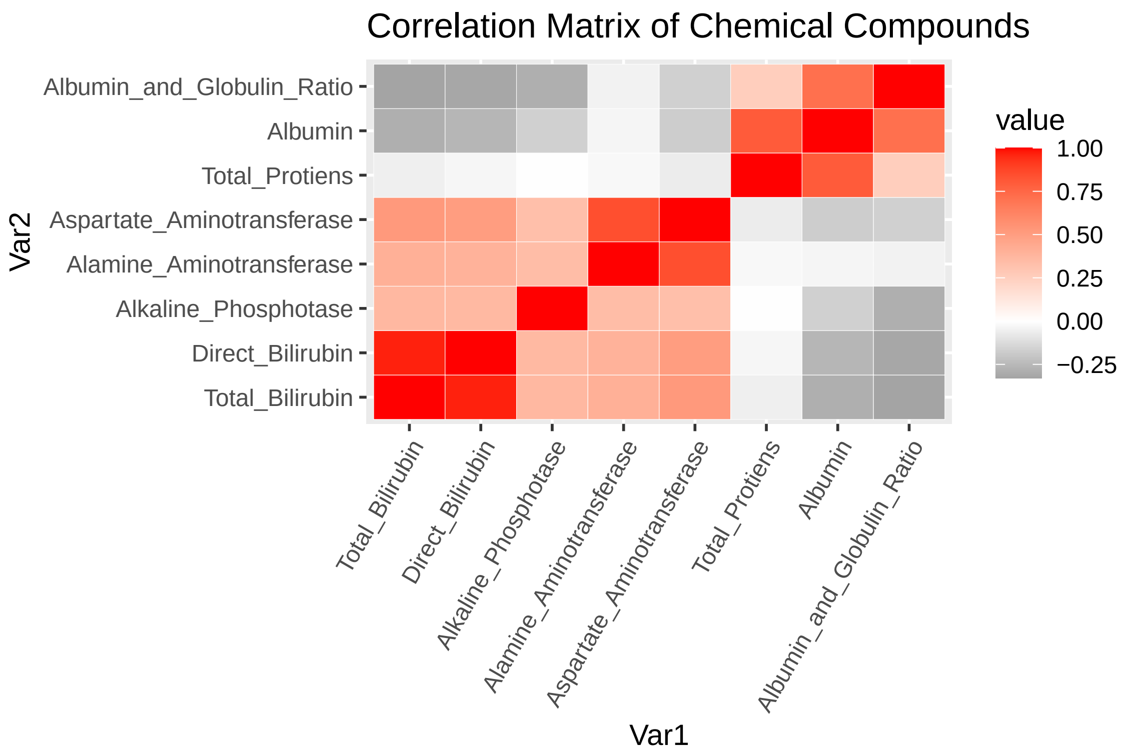
Another way to investigate age as a predictor for disease involves binning by age group. Here we bin by 20 year increments with age_bin 1 including those younger than 20 years and age_bin 4 including people 60 years and older. We can see from the bar chart and the table below that the youngest two bins (1&2) have a lower tendency to have liver disease than the older two groups. But again, within this dataset, all age bins are more likely to have the disease than not.



age_bin	Disease	NoDisease	Disease_proportion
1	14	11	0.5600000
2	104	57	0.6459627
3	151	46	0.7664975
4	59	24	0.7108434

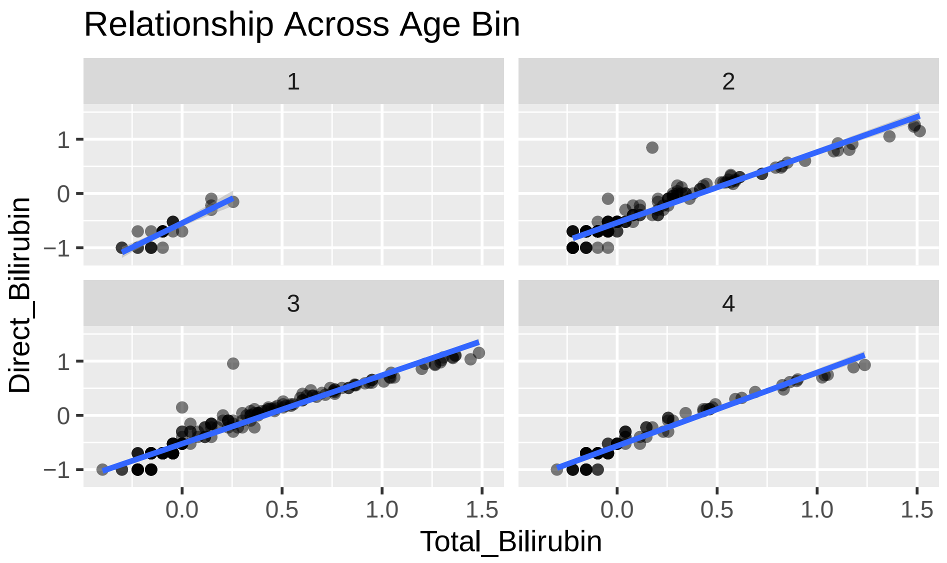
Now let's explore the chemical compounds in more detail. First of all, we know that we want to avoid multicollinearity in machine learning models and we noticed that several of the compounds had similar names (e.g. Total Bilirubin and Direct Bilirubin) so we want to understand if some of the features could be correlated.

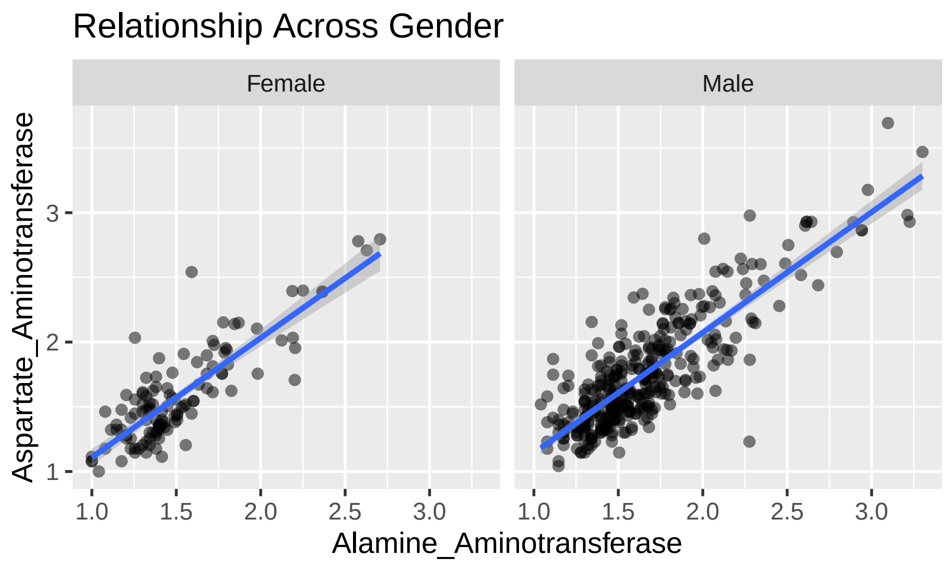
```
##                            Total_Bilirubin Direct_Bilirubin
## Total_Bilirubin                      1.000            0.966
## Direct_Bilirubin                     0.966            1.000
## Alkaline_Phosphotase                 0.370            0.365
## Alamine_Aminotransferase             0.412            0.399
## Aspartate_Aminotransferase           0.522            0.506
## Total_Protiens                      -0.054           -0.031
## Albumin                             -0.287           -0.260
## Albumin_and_Globulin_Ratio          -0.327           -0.314
##                            Alkaline_Phosphotase Alamine_Aminotransferase
## Total_Bilirubin                           0.370                    0.412
## Direct_Bilirubin                          0.365                    0.399
## Alkaline_Phosphotase                      1.000                    0.345
## Alamine_Aminotransferase                  0.345                    1.000
## Aspartate_Aminotransferase                0.334                    0.846
## Total_Protiens                           -0.004                   -0.024
## Albumin                                  -0.166                   -0.034
## Albumin_and_Globulin_Ratio               -0.286                   -0.044
##                            Aspartate_Aminotransferase Total_Protiens Albumin
## Total_Bilirubin                                 0.522         -0.054  -0.287
## Direct_Bilirubin                                0.506         -0.031  -0.260
## Alkaline_Phosphotase                            0.334         -0.004  -0.166
## Alamine_Aminotransferase                        0.846         -0.024  -0.034
## Aspartate_Aminotransferase                      1.000         -0.067  -0.175
## Total_Protiens                                 -0.067          1.000   0.800
## Albumin                                        -0.175          0.800   1.000
## Albumin_and_Globulin_Ratio                     -0.164          0.255   0.715
##                            Albumin_and_Globulin_Ratio
## Total_Bilirubin                                -0.327
## Direct_Bilirubin                               -0.314
## Alkaline_Phosphotase                           -0.286
## Alamine_Aminotransferase                       -0.044
## Aspartate_Aminotransferase                     -0.164
## Total_Protiens                                  0.255
## Albumin                                         0.715
## Albumin_and_Globulin_Ratio                      1.000
```



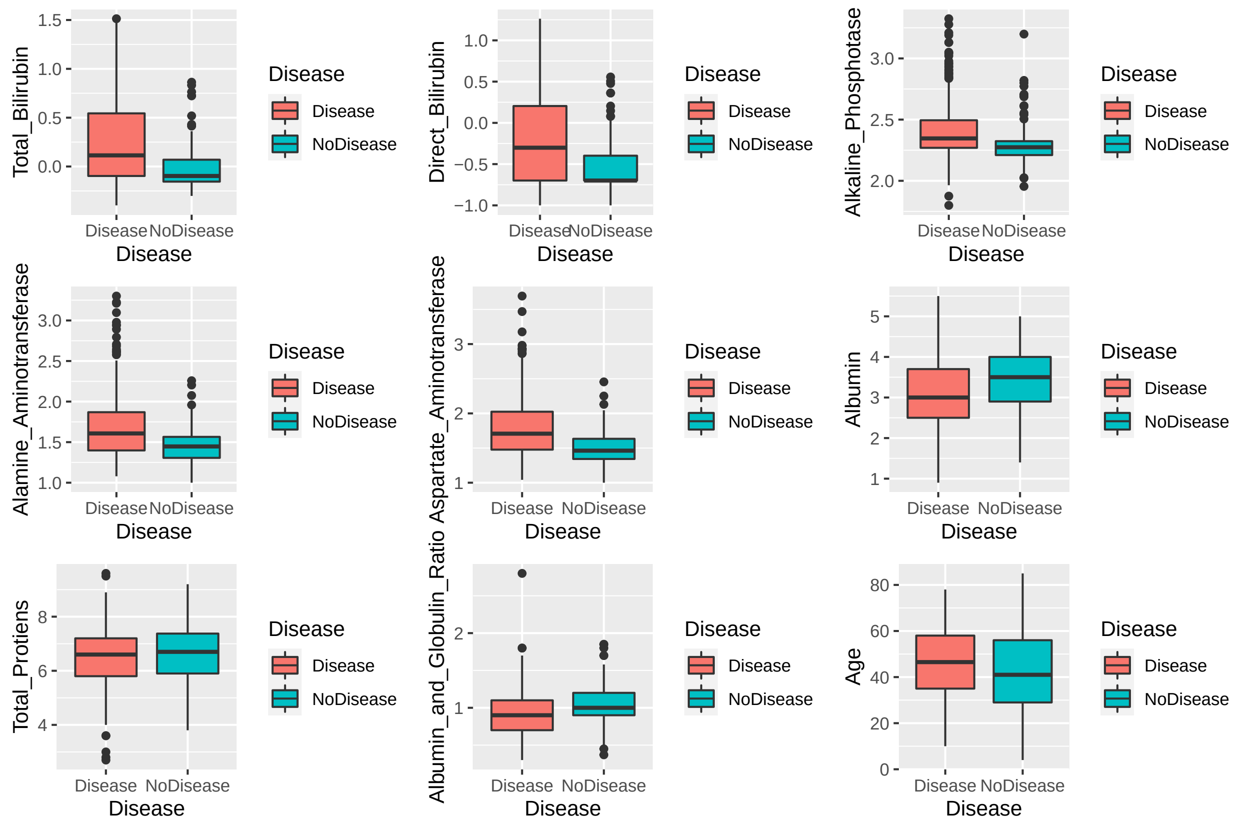
Just as we suspected, a number of compounds are highly correlated. When we start building models, we'll only select one feature for those pairs with correlation coefficient > 0.7.

To be diligent, we reviewed each highly correlated pair in closer detail, including stratifying by Gender and age_bin, to see if the relationship holds. In all cases, the linear correlation was strong and thus we stand by the decision to reduce to uncorrelated features. Below are a couple of examples of the stratification exercise.





To further aid in feature selection, let's try to visulize the potential impact each feature has on the presence of liver disease.



Here are the takeaways from the box plots: Total Bilirubin, Direct Bilirubin, Alkaline Phosphotase, and both Alamine and Aspartate Aminotranserases tend to be higher in patients with liver disease than those without. We already mentioned that age tended to be slightly higher when disease is present and we see that again here. Albumin and Albumin:Globulin Ratio tend to be lower in the patients with disease while Total Protiens seems to have no impact (the boxes are almost completely overlapping).

We have a couple of other tools we can use in factor selection. Here we run a chi squared analysis on both Gender and age_bin, and in both cases we see statistical significance (p value < 0.05). Both should be included as factors in our models, though where age is concerned we should choose either age_bin or the numeric Age value (not both).

Disease	female	male
Disease	73	255
NoDisease	44	94

```
## p value = 0.03833442
```

Disease	age_1	age_2	age_3	age_4
Disease	14	104	151	59
NoDisease	14	57	46	24

```
## p value = 0.008284916
```

Modeling approach

We could also use statistical tests for selection of the numeric features (such as t tests, or non-parametric tests for non-normal distributions). However, we decide to include the following factors due to our boxplot visualization as well as the lack of correlation: Total_Bilirubin, Alkaline_Phosphotase, Aspartate_Aminotransferase, Albumin, Gender, Age. When evaluating models, we could continue to reduce features if they aren't found to be significant. For our purposes, we won't continue reducing features as we want to evaluate several models and this report could get extremely lengthy. We don't expect that the extra factors are hurting the model performance.

As we stated initially, we have (created) a hypothetical scenario in which a doctor is asking us to build a predictive model, maximizing precision. The doctor has reason to believe that all patients within this dataset are at high risk for liver disease, hence the high proportion within the dataset having liver disease. We know that we could simply guess liver disease is present every time and obtain an accuracy and precision of ~70.4% on the training set, but this does us little good in our predictive ability moving forward. The doctor plans to use our algorithm to help justify recommending a costly and invasive procedure, thus she wants to know with as much certainty as possible that this is the correct decision. As for the patients in which the model deems to not have disease, she will continue monitoring and likely recommend further testing. So while ideally both precision and sensitivty would be very high we are prioritizing precision and tuning/selecting a model with this in mind.

We are going to evaluate a number of supervised machine learning models: logistic regression, random forest, k nearest neighbors, loess, and one unsupervised model: k means clustering.

We start by liming both the training set and the test set to the features we selected and applying the same transformation that we performed on the liver (train set) to the test set data. We should also make sure there is no data missing from the test set. Note that while we are preprocessing the test set data at this point, we are not using the test data for building/tuning the models or for evaluating model performance until the final model is chosen.

```
#check test set for NAs
any(is.na(test_set))
```

```
## [1] FALSE
```

```
#choose features and assign to train_set
train_set <- new_liver %>%
  select(Total_Bilirubin, Alkaline_Phosphotase, Aspartate_Aminotransferase, Albumin,
         Gender, Age, Disease)

#select same features for test_set
test_set <- test_set %>%
  mutate(Disease = as.factor(ifelse(Dataset == "1", "Disease", "NoDisease")),
         Gender = as.factor(Gender)) %>%
  select(Total_Bilirubin, Alkaline_Phosphotase, Aspartate_Aminotransferase, Albumin,
         Gender, Age, Disease, -Dataset)


#we need to apply log10 transformation to the highly skewed features in test_set which we did for train
test_set <- test_set %>% mutate(Total_Bilirubin = log10(Total_Bilirubin),
                                Alkaline_Phosphotase = log10(Alkaline_Phosphotase),
                                Aspartate_Aminotransferase = log10(Aspartate_Aminotransferase))
```

Because we intend to use models which utilize euclidian distance (knn, k means clustering), we need to center and scale the data. Doing this should not negatively impact the other models which don't require scaling.

```
#convert features to matrix and Disease response (y) to factor vector
train_x <- train_set %>%
  select(-Disease) %>%
  data.matrix()
train_y <- train_set$Disease

#center and scale the matrix train_x
train_x_centered <- sweep(train_x, 2, colMeans(train_x))
train_x_scaled <- sweep(train_x_centered, 2, colSds(train_x), FUN = "/")


#apply scaling to test_set, using data from train_set since test is "unknown"
#first convert test set to matrix and response vector
test_x <- test_set %>%
  select(-Disease) %>%
  data.matrix()
test_y <- test_set$Disease

#obtain train set column means and standard deviations
means <- colMeans(train_x)
std_devs <- colSds(train_x)

test_x_centered <- sweep(test_x, 2, means)
test_x_scaled <- sweep(test_x_centered, 2, std_devs, FUN = "/")

#Our final training and test feature matrices
```

```
train_x <- train_x_scaled
test_x <- test_x_scaled
#train_y and test_y are just the Disease response vectors previously defined
```

Now we are ready to build and evaluate machine learning models.

3. Modeling Results

For all of the supervised models we are planning to evaluate using the train function, we use the default cross validation: 25 bootstrap samples comprised of 25% of train set. We can evaluate performance of the model by reviewing the confusion matrix comprised only of train set data.

Let's start simple with logistic regression.

```
#set seed for conistency in results
set.seed(1, sample.kind = "Rounding")

#train logistic regression model
train_glm <- train(train_x, train_y, method = "glm", family = "binomial")

#confusion matrix for train cross validation
cm_glm <- confusionMatrix(train_glm, "none")
print(cm_glm)
```

```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##            Reference
## Prediction  Disease NoDisease
##   Disease      2590       947
##   NoDisease     416       370
##
##  Accuracy (average) : 0.6847
```

```
#precision of the glm model
Precision_glm <- cm_glm$table[1,1]/(cm_glm$table[1,1] + cm_glm$table[1,2])
cat("Precision =", Precision_glm)
```

```
## Precision = 0.732259
```

The glm model obtained a precision of 73.2% on the train set data. While this is improved from the 70.4% we would obtain from a totally biased model (guess Disease every time), we are hoping to achieve much better precision for the doctor's use case.

As mentioned previously, one thing we would probably do in a real-life scenario is reduce the model to significant features. We can see that in the case of the glm, we could simplify the model by eliminating Albumin, Gender and Alkaline Phosphotase from the feature list as they all have p values > 0.05. Another way to view this is by variable importance and we can see that the three features not found to be significant have lower importance in the model.

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5614  -0.8769  -0.4703   1.0989   2.7207
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -1.14625    0.13118  -8.738  < 2e-16 ***
## Total_Bilirubin              -0.43946    0.18146  -2.422  0.01544 *
## Alkaline_Phosphotase         -0.27897    0.14768  -1.889  0.05889 .
## Aspartate_Aminotransferase -0.67411    0.17523  -3.847  0.00012 ***
## Albumin                       0.09021    0.12084   0.746  0.45537
## Gender                       -0.05155    0.10897  -0.473  0.63618
## Age                          -0.25833    0.11574  -2.232  0.02562 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 566.24  on 465  degrees of freedom
## Residual deviance: 484.03  on 459  degrees of freedom
## AIC: 498.03
##
## Number of Fisher Scoring iterations: 5
```

```
## glm variable importance
##
##                              Overall
## Aspartate_Aminotransferase  3.8470
## Total_Bilirubin              2.4218
## Age                          2.2320
## Alkaline_Phosphotase         1.8890
## Albumin                      0.7465
## Gender                       0.4731
```

Because our purpose right now is to evaluate several models, so we aren't going to be going through the excerise of reducing/simplifying each model.

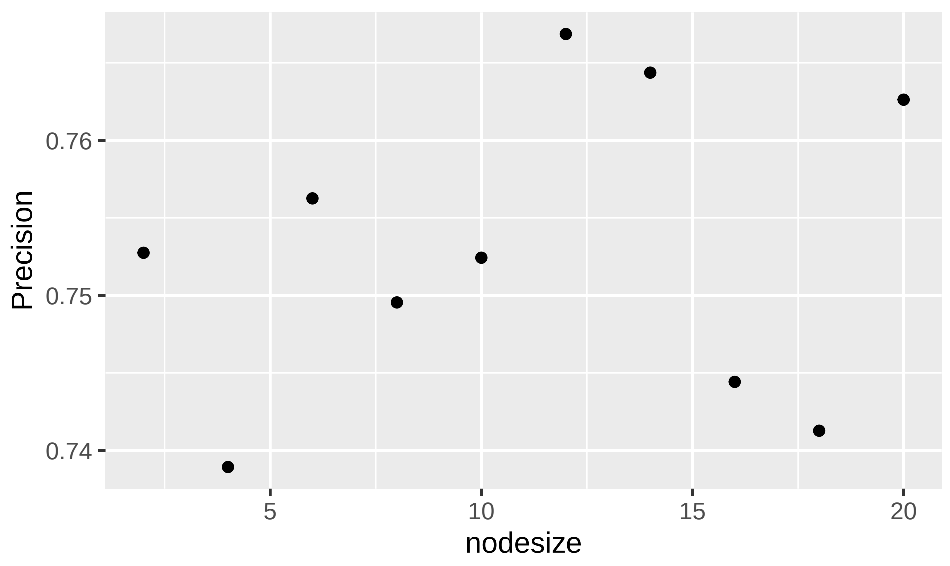
The next model we want to try is a random forest. A random forest is essentially a collection of uncorrelated decision trees and the final model is built by committee, thus obtaining a more accurate result than that of any individual tree. First let's tune the mtry parameter, which is the number of predictors that will be randomly sampled at each split. For train control, we use prSummary to allow us to train the model using precision as the metric to maximize.

```r
trctrl    = trainControl(summaryFunction = prSummary, classProbs = TRUE)

set.seed(1, sample.kind = "Rounding")
train_rf_1 <- train(train_x, train_y,
                    method = "rf",
                    trControl=trctrl,
                    metric = "Precision",
                    tuneGrid = data.frame(mtry = seq(1, 5, 1)))
print(train_rf_1)
```

```
## Random Forest
##
## 466 samples
##   6 predictor
##   2 classes: 'Disease', 'NoDisease'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 466, 466, 466, 466, 466, 466, ...
## Resampling results across tuning parameters:
##
##   mtry  AUC        Precision  Recall     F
##   1     0.8541850  0.7285026  0.9247837  0.8129714
##   2     0.8514978  0.7522639  0.8426334  0.7932573
##   3     0.8425176  0.7527172  0.8335047  0.7895504
##   4     0.8347697  0.7542339  0.8260425  0.7870653
##   5     0.8264485  0.7523278  0.8196964  0.7831128
##
## Precision was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

An mtry value of 4 resulted in the highest precision. Next we can tune node size. This needs to be done manually as it is not a tuneable parameter using the train function with the rf method in R. Nodesize is the minimum number of samples that can be split into a node. Ideally, we could have tuned mtry and nodesize simulateously rather than successively, and we're aware we might not be finding the ideal parameter values by this approach.

```
#now manually select nodesize, setting mtry to 4
set.seed(1, sample.kind = "Rounding")
nodesize <- seq(2, 20, 2)
Precision <- sapply(nodesize, function(ns){
  train(train_x, train_y, method = "rf",
        trControl=trctrl,
        metric = "Precision",
        tuneGrid = data.frame(mtry = 4),
        nodesize = ns)$results$Precision
})
qplot(nodesize, Precision)
```



A nodesize of 12 resulted in the highest precision. Finally, we build the random forest model using mtry = 4 and nodesize = 12 and obtain a precision of 0.746.

```r
#build model (mtry = 4, nodesize = 12)
set.seed(1, sample.kind = "Rounding")
train_rf <- train(train_x, train_y,
                  method = "rf",
                  trControl=trctrl,
                  metric = "Precision",
                  tuneGrid = data.frame(mtry = 4),
                  nodesize = 12)

#confusion matrix for train cross validation
cm_rf <- confusionMatrix(train_rf, "none")
print(cm_rf)
```

```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##            Reference
## Prediction  Disease NoDisease
##   Disease      2523       861
##   NoDisease     483       456
##
##  Accuracy (average) : 0.6891
```

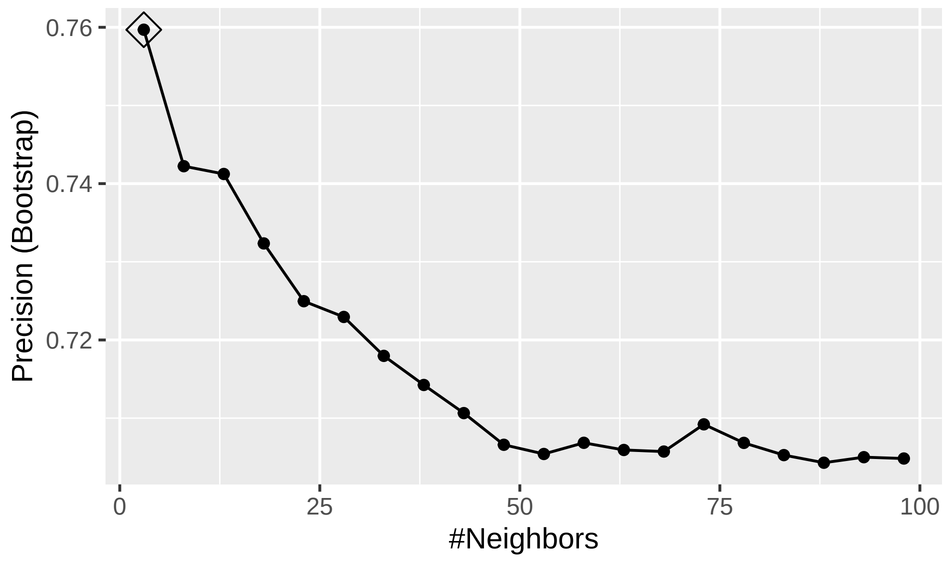
```r
#calculate precision
Precision_rf <- cm_rf$table[1,1]/(cm_rf$table[1,1] + cm_rf$table[1,2])
cat("Precision =", Precision_rf)
```

```
## Precision = 0.7455674
```

We have slightly better precision using random forest than glm, but we still want to see if another approach would be better.

K-nearest neighbors is a non-parametric method, and in the case of classification, the record will be classified by a plurality vote of it's closest neighbors. "k" is the number of neighbors which we consider for making the decision. We will now determine the k which maximizes precision in the train dataset.

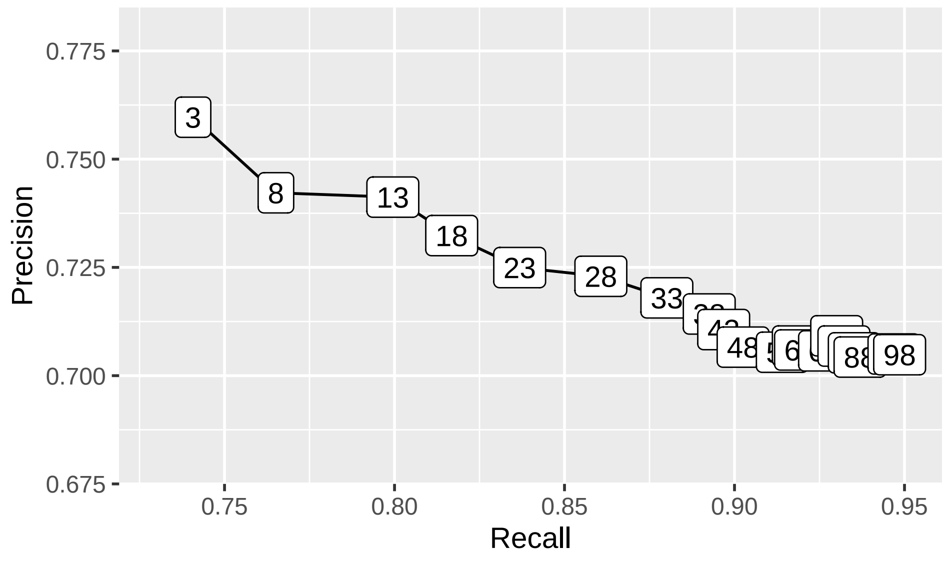
```
set.seed(1, sample.kind = "Rounding")
train_knn_1 <- train(train_x, train_y,
                     method = "knn",
                     trControl = trctrl,
                     metric = "Precision",
                     tuneGrid = data.frame(k = seq(3, 100, 5)))
ggplot(train_knn_1, highlight = TRUE)
```



```
train_knn_1$bestTune
```

```
##   k
## 1 3
```

We can visualize the tradeoff we are making between precision and sensitivity (recall). You can see below that with large values of k, sensitivity gets very high... because with large k we would tend to predict Disease due to the prevalence of Disease in the dataset.



We set k=3, train the model and evaluate the bootstrap cross validation. We can see that we obtained the highest precision yet, 0.759. . . . but we still want to see if we can do better.

```
set.seed(1, sample.kind = "Rounding")
train_knn <- train(train_x, train_y,
                     method = "knn",
                     tuneGrid = data.frame(k = 3))

#confusion matrix for knn
cm_knn <- confusionMatrix(train_knn, "none")
print(cm_knn)
```

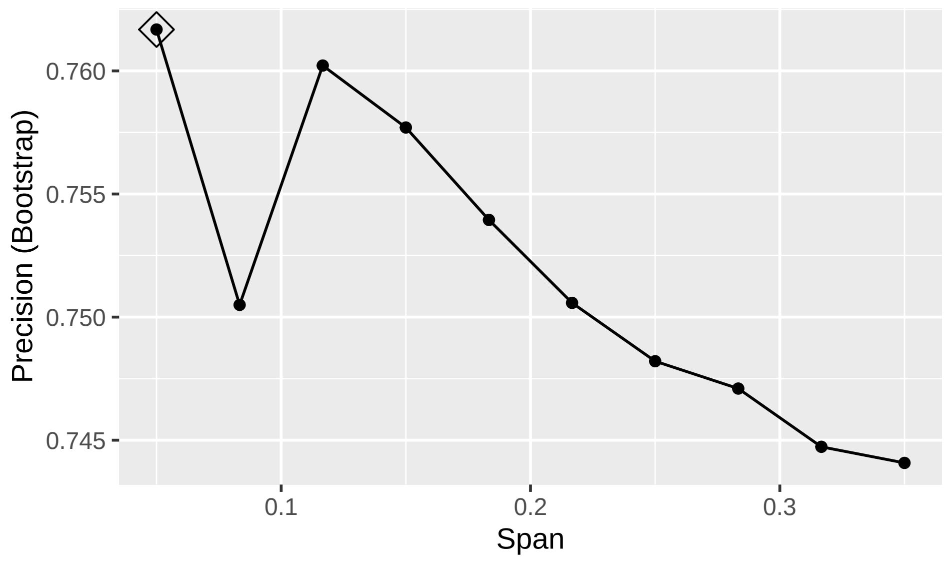
```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##              Reference
## Prediction  Disease NoDisease
##   Disease      2220       706
##   NoDisease     786       611
##
##  Accuracy (average) : 0.6549
```

```
#precision for knn
Precision_knn <- cm_knn$table[1,1]/(cm_knn$table[1,1] + cm_knn$table[1,2])
cat("Precision =", Precision_knn)
```

```
## Precision = 0.758715
```

Perhaps loess, which uses a kernel to make a more smooth prediction will be the answer. We will tune the span, evaluating 5%-35%, but we choose to leave degree = 1. (The code crashed when I tried degree = 1 and 2 and I read about a bug online that was related to this)

```r
set.seed(1, sample.kind = "Rounding")
grid <- expand.grid(span = seq(0.05, 0.35, len = 10), degree = 1)
train_loess_1 <- train(train_x, train_y,
                       method = "gamLoess",
                       tuneGrid = grid,
                       trControl = trctrl,
                       metric = "Precision")
ggplot(train_loess_1, highlight = TRUE)
```



```r
#span = 0.05 had highest precision
train_loess <- train(train_x, train_y,
                     method = "gamLoess",
                     span = 0.05)

#confusion matrix
cm_loess <- confusionMatrix(train_loess, "none")
print(cm_loess)
```

```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##            Reference
## Prediction  Disease NoDisease
##   Disease      2636       950
##   NoDisease     358       373
##
##  Accuracy (average) : 0.697
```

```r
#Precision
Precision_loess <- cm_loess$table[1,1]/(cm_loess$table[1,1] + cm_loess$table[1,2])
cat("Precision =", Precision_loess)
```

```
## Precision = 0.7350809
```

The final loess model using the best tune (span = 0.05) resulted in a precision of only 0.735, slightly better than glm, but not as good as knn or random forest.

At this point we want to shift gears and try an unsupervised model approach. With K-means clustering, we can use the training feature space to establish the two optimal centroids. Then we can see how well the actual disease status in the training set is explained by these two groups (2-means).

```
set.seed(1, sample.kind = "Rounding")
k <- kmeans(train_x, centers = 2, n = 25)
print(k)
```

```
## K-means clustering with 2 clusters of sizes 300, 166
##
## Cluster means:
##   Total_Bilirubin Alkaline_Phosphotase Aspartate_Aminotransferase    Albumin
## 1      -0.5341703           -0.4276368                 -0.4826146  0.3053841
## 2       0.9653679            0.7728377                  0.8721951 -0.5518991
##       Gender        Age
## 1 -0.0973525 -0.1060879
## 2  0.1759383  0.1917252
##
## Clustering vector:
##   [1] 1 2 2 1 2 1 1 1 1 2 1 1 1 2 1 2 2 2 2 2 2 1 2 2 2 1 2 1 1 2 2 1 2 2 1 1 2
##  [38] 1 2 1 2 1 2 1 1 1 2 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 1 2 2 1 1 2 1 1 1
##  [75] 1 1 2 2 2 2 2 2 1 2 2 1 2 1 1 2 2 1 1 1 1 2 1 2 2 2 2 2 1 1 1 1 2 2 2 2 1
## [112] 1 1 2 2 2 2 1 1 1 2 2 2 1 2 1 1 1 1 1 2 2 1 1 1 2 2 1 1 2 2 2 2 1 1 2 2 2
## [149] 2 2 2 1 1 2 2 2 1 2 2 2 1 1 2 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 2 1 1 1 1 1
## [186] 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 2 1 1 1 1 2 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1
## [297] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 2 2 2 1 1 2 2 1
## [334] 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 1 2 1 2 1 1 1 2 1 1 1 1 1
## [371] 1 1 1 1 1 1 1 1 2 1 2 2 1 2 1 2 1 2 1 1 1 1 2 1 1 1 1 1 1 2 2 2 1 2 2 2 1
## [408] 2 1 1 1 1 2 1 1 1 2 2 1 1 1 1 2 2 1 2 1 2 1 2 2 1 1 1 1 1 1 1 2 2 2 2 2 2
## [445] 1 1 2 1 1 1 2 2 2 2 1 2 1 2 2 2 2 2 2 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 1171.8884  931.6433
##  (between_SS / total_SS =  24.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

We can see by the centroids associated with each cluster, that theoretically cluster 2 should indicate disease (higher Total_Bilirubin, Alkaline Phosphotase, Aspartate Aminotransferase, Age and Gender = M). Let's create a table and determine how well, within the train set, that each cluster predicts the disease status.

```
train_x_df <- data.frame(cbind(k$cluster, train_x))
cluster_table <- cbind(Disease = train_y, train_x_df)
```

```r
cluster_table$cluster <- as.character(k$cluster)

cluster_table <- cluster_table %>%
  mutate(cluster_result = case_when(Disease == "Disease" & cluster == 2 ~ "True Positive",
                                    Disease == "Disease" & cluster == 1 ~ "False Negative",
                                    Disease == "NoDisease" & cluster == 1 ~ "True Negative",
                                    Disease == "NoDisease" & cluster == 2 ~ "False Positive"))
cluster_precision <- (sum(cluster_table$cluster_result == "True Positive"))/
  (sum((cluster_table$cluster_result == "True Positive")
       + (cluster_table$cluster_result == "False Positive")))

cluster_accuracy <- (sum((cluster_table$cluster_result == "True Positive") +
                           (cluster_table$cluster_result == "True Negative")))/
  (sum((cluster_table$cluster_result == "True Positive") +
         (cluster_table$cluster_result == "False Positive")
       + (cluster_table$cluster_result == "True Negative") +
         (cluster_table$cluster_result == "False Negative")))

cat("Precision =", cluster_precision)
```

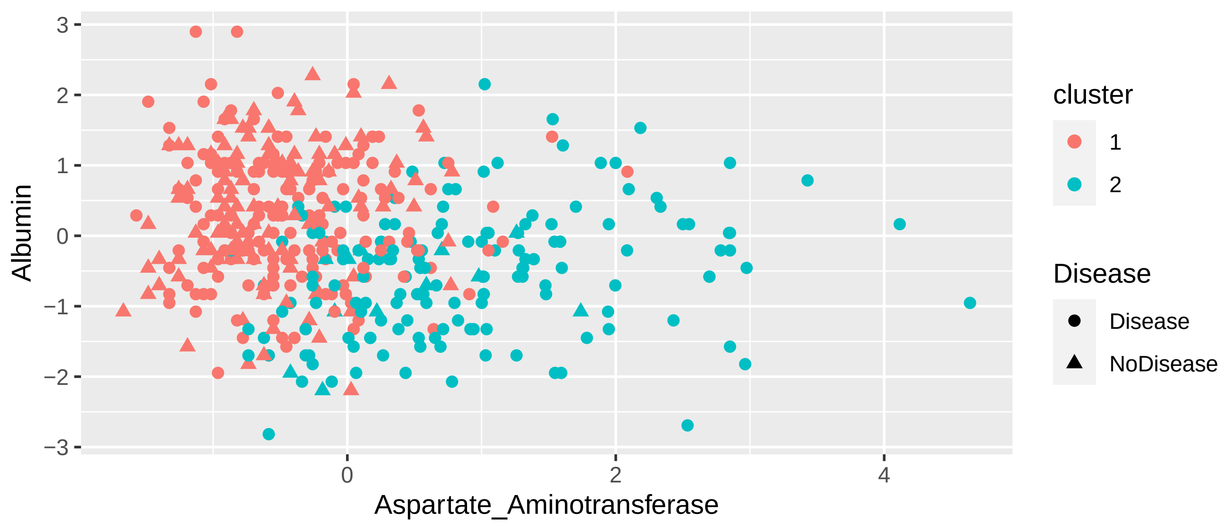
```
## Precision = 0.9156627
```

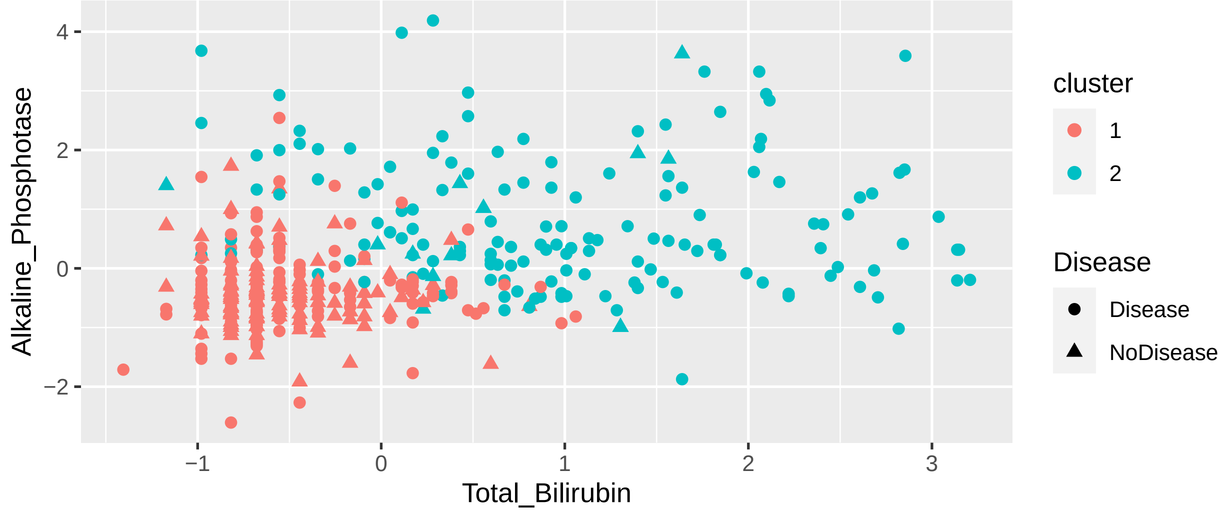
```r
cat("Accuracy =", cluster_accuracy)
```

```
## Accuracy = 0.5922747
```

Wow! Precision using k-means on the training set is all the way up to ~91.6%! This is by far the highest precision we've seen. The accuracy dropped to ~59% which indicates some other issues (in our case the biggest concern being that sensitivy must be lower), but given the goal to optimize precision, it appears k-means is the best option.

We can try to visualize the results to understand what is happening. We can see some of the results of clustering below. Cluster 1 should be associated with NoDisease. Thus the red triangles are true negatives while the red circles are false negatives. Cluster 2 should be associated with disease…so the blue circles are true positives while the blue triangles are false positives. You can see that there are very few false positives (blue triangles) which makes sense given the high precision. But you can also understand why there are so many false negatives in the performance of the algorithm as the red circles and red triangles do appear to be clustered close together and difficult or impossible to be distinguished by distance.





After reviewing these results, we believe that the k-means clustering algorithm is the best model we can build to predict disease, given that precision is the most important metric to the doctor. We can see that using this model, we predict NoDisease more often than we predict Disease, but we're very happy that when we predict Disease we're right more than 90% of the time.

Final model and evaluation against test set

Finally, we use the cluster centers from the training set and predict the disease status for each record in the test set by determing which cluster center has the shortest distance to the record. The final model results for the test set predictions are shown in the confusion matrix below.

```r
predict_kmeans <- function(x, k) {
  centers <- k$centers    # extract cluster centers
  #calculate distance to cluster centers
  distances <- sapply(1:nrow(x), function(i){
    apply(centers, 1, function(y) dist(rbind(x[i,], y)))
  })
  max.col(-t(distances))  # select cluster with min distance to center
}

kmeans_preds <- ifelse(predict_kmeans(test_x, k) == 2, "Disease", "NoDisease")
confusionMatrix(data = factor(kmeans_preds), reference = test_y, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Disease NoDisease
##   Disease        46         2
##   NoDisease      42        27
##
##                Accuracy : 0.6239
##                  95% CI : (0.5296, 0.7118)
##     No Information Rate : 0.7521
##     P-Value [Acc > NIR] : 0.9993
##
##                   Kappa : 0.3103
##
##  Mcnemar's Test P-Value : 4.116e-09
##
##             Sensitivity : 0.5227
##             Specificity : 0.9310
##          Pos Pred Value : 0.9583
##          Neg Pred Value : 0.3913
##               Precision : 0.9583
##                  Recall : 0.5227
##                      F1 : 0.6765
##              Prevalence : 0.7521
##          Detection Rate : 0.3932
##    Detection Prevalence : 0.4103
##       Balanced Accuracy : 0.7269
##
##        'Positive' Class : Disease
##
```

As you can see, we were very successful in optimizing precision… nearly 96%! Even though we don't love the overall accuracy at ~62% or the sensitivity of ~52%, the model is quite good for the doctor's use case and we were aware there would be a trade-off when deciding to optmize precision.

4. Conclusion

This report contains the performance evaluation of several machine learning algorithms using the Indian Liver Patient dataset, as well as the cleaning, visualizations, analysis, and preprocessing required to build better performing models. The goal of this project was to determine which algorithm among logistic regression, knn, loess, random forest and k-means clustering would achieve the highest precision. Ultimately, k-means clustering was chosen to predict the disease status within the test set and a precision of nearly 96% was achieved.

Had the goal of the project been to maximize accuracy or sensitivity, a model other than k-means would have been better suited. We could have taken a similar approach, but tuned the pararmeters by optimizing accuracy or sensitiviy within the training set cross validation (train function). In the case of sensitivity, we would have needed to be cautious of the Disease prevalence as a sensitivty of 100% (and accuracy of 70%) could have been achieved by guessing Disease in every case for the training set data, but this might not have gone well when evaluated against the "future" test set. Given this issue, another metric that would have made sense to optimize is the F1 score, which is the harmonic mean between precision and sensitivity. By tuning the supervised models against the F1 score, we would have ended up with lower precision but better sensitivity (fewer false negatives).

While working on this project, I found that it is tempting to try to keep improving the models and achieving better metrics across the board ("a perfect model"). But after several iterations and many long days, I found that there are always going to be tradeoffs… hence the saying, "all models are wrong, but some are useful". In the case of this dataset, we saw when viewing the boxplots (by disease status) that portions of all of the features are overlapping, and this was also evident by visualizing the clustering analysis on the train set. There wasn't a clean split for any feature where Disease is on one side and No Disease is on the other. So while we can build models using the tendency of features to be higher (or lower) when disease is present, we can't explain all of the results by using only the data provided.

Overall I thought this was a great exercise and I'm very happy with the result of 96% precision.