Expert System To Assess Machinery Deals and Washout

Introduction

The company sells and services machinery, typically combines, telehandlers and tractors. The value of such machinery is high, so in accounting terms each deal is 'material'. Typically a combine sells for £300,000.

When a machine is sold the customer will usually want to trade-in an older machine. That older machine may easily be 50% of the value of the machine being sold, whereas the profit in the deal may have been 10%. So, the deal consumes cash, it does not generate cash for the business unless the trade-in is also sold. This involves another deal, where that customer may also want to trade-in a machine. And so on, until there is a deal without a trade-in and the chain of deals is said to be 'washed out'. This can take months or even years!

Because deals lock up cash, instead of generate cash, for a long period of time, the sales reps present their deals for approval by the finance director. This is done via an online sales order processing tool. The finance director must judge whether the profit ultimately achieved by the washed-out chain of deals will be worth the cash being locked up by the deal being presented.

This judgement is a real art. The finance director is being asked to make a forecast based on the customer, the rep's history, the machines being sold, the economic climate etc. He may have 20 such judgements to make in a day and many other tasks besides. Which deals should he query with the sales rep?

The task of this paper is to investigate possible tools to identify those deals likely to belong to loss making chains.

Executive Summary

3,500+ deals are extracted from the accounting system using SQL. The data wrangling is not straight forward and half of the paper is SQL.

Various statistical learning models and two deep learning models are then attempted on this classification problem. The random forest is found to be best, having the most appropriate 'F1'. It is further investigated with various probability thresholds in this classificaiton problem.

Extract relevant data from the accounting system

The accounting systems stores deals, these can be extracted via SQL, along with relevant data such as the customer's debtor status, machinery type, age, season of deal (its a seasonal industry) etc.

Since the data is already in a structured database (Microsoft SQL Server 2016), it's convenient to complete the data wrangling in Microsoft's Transact SQL rather than sqldf, dplyr or base R.

The task of extracting the relevant data is split into the following steps of sql:

1. The accounting system has no concept of a washout chain, so we must link deals into washout chains ourselves
2. The outcome we are seeking is whether a chain is profitable, so we must aggreagte the result of each deal into a sum for each chain
3. Candidate predictors are extracted; specification and age of the machine being sold, debtor history of the customer, the sales rep ID, etc.

The SQL code can be found in an appendix at the foot of this notebook.

Connect to the SQL Data

Let's take our first look at the data provided by SQL

```
library(RODBC)

##Create a connection to the database using ODBC
conn<-odbcConnect(dsn="RODBC",uid="sa",pwd="CauliGOOgle")

##Extract the required part_number data
chains_all <- sqlQuery(conn, "SELECT *
                              FROM [41_ChainAnalysisByAdvice]
                              ORDER BY CHAIN_NO, ADVICE",
                              stringsAsFactors=FALSE) #we'll set the factors by hand
#Close connection
odbcClose(conn)
print(str(chains_all))
```

There is a deep learning model used later in this notebook. If using Paperspace, or other service, for cloud computing then you need to export the data to file and then import in the cloud environment. Here is that export.

How skewed is this data?

In other words, how many examples do we have of deals belonging to loss making (worthwhile=0) chains, vs deals belonging to profitable chains (worthwhile=1)?

```
paste0('Total deals=',nrow(chains_all),
       '. Profitable=', sum(chains_all$Worthwhile==1),
       '. Unprofitable =', sum(chains_all$Worthwhile==0))
```

```
## [1] "Total deals=3571. Profitable=3222. Unprofitable =349"
```

So, this is a highly skewed dataset, only approx 10% are unprofitable. This will make a useful model much harder to produce. With such a skew even a model which simply guessed 'profitable' all the time would get 90% accuracy.

Class imbalance difficulties are commonly addressed by over-sampling the rare class and/or under-sampling the common class. This can easily be done using ovun.sample() from the ROSE package. But for now, we keep see what we cna achieve with the data 'as is'.

Prepare data for categorical regressions

As ever there are som gaps in teh data, we'll replace those with the relevant mean. Also, there are a number of categorical columns, for example, REP, which must be made into 'factors'.

```
#########################################################
#replace NA with mean
#########################################################

chains_all$Debtor_OutstandingMth0[which(is.na(chains_all$Debtor_OutstandingMth0) == T)] <-
    mean(chains_all$Debtor_OutstandingMth0, na.rm=TRUE)
chains_all$Debtor_OutstandingMth1[which(is.na(chains_all$Debtor_OutstandingMth1) == T)] <-
    mean(chains_all$Debtor_OutstandingMth1, na.rm=TRUE)
chains_all$Debtor_OutstandingMth2[which(is.na(chains_all$Debtor_OutstandingMth2) == T)] <-
```

```
    mean(chains_all$Debtor_OutstandingMth2, na.rm=TRUE)
chains_all$Debtor_OutstandingMth3Plus[which(is.na(chains_all$Debtor_OutstandingMth3Plus) == T)] <-
    mean(chains_all$Debtor_OutstandingMth3Plus, na.rm=TRUE)
chains_all$Debtor_OutstandingTotal[which(is.na(chains_all$Debtor_OutstandingTotal) == T)] <-
    mean(chains_all$Debtor_OutstandingTotal, na.rm=TRUE)
chains_all$WholegoodSold_YEARS_OLD[which(is.na(chains_all$WholegoodSold_YEARS_OLD) == T)] <-
    mean(chains_all$WholegoodSold_YEARS_OLD, na.rm=TRUE)
chains_all$TradeIn_YEARS_OLD[which(is.na(chains_all$TradeIn_YEARS_OLD) == T)]<-
    mean(chains_all$TradeIn_YEARS_OLD, na.rm=TRUE)

#######################################################
#change text and numerical categories into factors
#######################################################

chains_all$ADVICE <- as.factor(chains_all$ADVICE)
chains_all$INVOICE <- as.factor(chains_all$INVOICE)
chains_all$CHAIN_NO <- as.factor(chains_all$CHAIN_NO)
chains_all$CUSTOMER <- as.factor(chains_all$CUSTOMER)
chains_all$REP <- as.factor(chains_all$REP)
chains_all$WholegoodSold_NEW_USED <- as.factor(chains_all$WholegoodSold_NEW_USED)
chains_all$WholegoodSold_MachineTypeDesc <- as.factor(chains_all$WholegoodSold_MachineTypeDesc)
chains_all$WholegoodSold_MODEL_CODE <- as.factor(chains_all$WholegoodSold_MODEL_CODE)
chains_all$WholegoodSold_MAKE_CODE <- as.factor(chains_all$WholegoodSold_MAKE_CODE)
chains_all$TradeIn_NEW_USED <- as.factor(chains_all$TradeIn_NEW_USED)
chains_all$TradeIn_MachineTypeDesc <- as.factor(chains_all$TradeIn_MachineTypeDesc)
chains_all$TradeIn_MODEL_CODE <- as.factor(chains_all$TradeIn_MODEL_CODE)
chains_all$TradeIn_MAKE_CODE <- as.factor(chains_all$TradeIn_MAKE_CODE)
chains_all$Worthwhile <- as.factor(chains_all$Worthwhile)

#######################################################
#replace NA with 0 or 1
#######################################################

chains_all$slope[which(is.na(chains_all$slope) == T)] <- 0
chains_all$intercept[which(is.na(chains_all$intercept) == T)] <- 0
chains_all$YearsToWait[which(is.na(chains_all$YearsToWait) == T)] <- 0
chains_all$TradeIn_Cost_Adj[which(is.na(chains_all$TradeIn_Cost_Adj) == T)] <- 0
chains_all$ExpectedProfitFromTradeIn[which(is.na(chains_all$ExpectedProfitFromTradeIn) == T)] <- 0
chains_all$ExpectedProfitFromChain[which(is.na(chains_all$ExpectedProfitFromChain) == T)] <- 0
chains_all$RatioSalesThisYrVsPrevYr[which(is.na(chains_all$RatioSalesThisYrVsPrevYr) == T)] <- 0
```

Scale the data

Most models prefer scaled data, else predictors which are arbitrarily large (eg distances measured in mm) tend to dominate for no good reason. Of course, categorical fields, factored in the chunk above, cannot be scaled.

```
#identify the columns which are factors
ColIsFactor <- seq(1:ncol(chains_all))[sapply(chains_all, function(x) is.factor(x))]

#ensure the names are all 'R' compliant, else caret (used later) throws errors
for (i in ColIsFactor){
  levels(chains_all[,i]) <- make.names(levels(chains_all[,i]))
```

```
}

#scale the non-factor columns
chains_all_scaled <- sapply(chains_all[,-ColIsFactor],
                            function(x) scale(x, center = TRUE, scale = TRUE))
chains_all_scaled <- cbind(chains_all_scaled, chains_all[,ColIsFactor])

#Keep the mean and SD for CHAIN_PROFIT_ADJ so we can 'unscale' this later
CHAIN_PROFIT_ADJ_mean <- mean(chains_all$CHAIN_PROFIT_ADJ)
CHAIN_PROFIT_ADJ_sd   <- sd(chains_all$CHAIN_PROFIT_ADJ)

print(str(chains_all_scaled))
```

```
## 'data.frame':    3571 obs. of  44 variables:
##  $ DATE_SOLD                     : num  -1.74 -1.77 -1.74 -1.67 -1.7 ...
##  $ DATE_SOLD_Month               : num  -0.5442 -0.8624 -0.226 0.4104 0.0922 ...
##  $ DATE_SOLD_Year                : num  -1.69 -1.69 -1.69 -1.69 -1.69 ...
##  $ ADVICE_QTY_WHOLEGOODS_SOLD    : num  -0.264 -0.264 -0.264 -0.264 -0.264 ...
##  $ ADVICE_QTY_TRADEIN_BOUGHT     : num  -0.313 -0.313 -0.313 -0.313 -0.313 ...
##  $ ADVICE_SALES                  : num  -0.19 0.108 -0.023 -0.256 -0.386 ...
##  $ ADVICE_PROFIT                 : num  -0.66626 0.16431 0.17514 1.39433 -0.00781 ...
##  $ ADVICE_PROFIT_ADJ             : num  -0.745 0.29 0.367 -0.17 0.132 ...
##  $ CHAIN_SALES                   : num  -0.33394 -0.00241 -0.21283 -0.42378 -0.42552 ...
##  $ CHAIN_PROFIT                  : num  -0.5705 0.0738 -0.0754 0.8897 -0.2202 ...
##  $ CHAIN_MARGIN                  : num  -1.0868 0.0238 0.0738 4.642 0.2302 ...
##  $ CHAIN_PROFIT_ADJ              : num  -0.6093 0.2442 -0.0259 -0.2471 -0.1013 ...
##  $ CHAIN_QTY_WHOLEGOODS_SOLD     : num  -0.734 -0.734 -0.734 -0.734 -0.734 ...
##  $ ChainSeq_SeqNo                : num  -0.655 -0.655 -0.655 -0.655 -0.655 ...
##  $ ChainSeq_CumulativeProfit     : num  -0.563 -0.0361 -0.0274 0.9556 -0.1749 ...
##  $ ChainSeq_CumulativeProfitAdj  : num  -0.6172 0.0698 0.1324 -0.3046 -0.0589 ...
##  $ Debtor_OutstandingMth0        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Debtor_OutstandingMth1        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Debtor_OutstandingMth2        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Debtor_OutstandingMth3Plus    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Debtor_OutstandingTotal       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ WholegoodSold_YEARS_OLD       : num  0 0 0 0 0 ...
##  $ TradeIn_YEARS_OLD             : num  0 0 0 0 0 ...
##  $ TradeIn_Cost_Adj              : num  -0.09615 0.82269 0.27065 -0.33868 0.00208 ...
##  $ slope                         : num  0.6441 0.0896 0.6441 -0.6213 -0.0087 ...
##  $ intercept                     : num  -0.41 -0.188 -0.41 0.155 -0.892 ...
##  $ YearsToWait                   : num  0.512 0.267 0.512 -1.077 -0.496 ...
##  $ ExpectedProfitFromTradeIn     : num  -0.1486 0.2966 0.0996 -0.2019 -0.6901 ...
##  $ ExpectedProfitFromChain       : num  -0.5643 0.0928 -0.065 0.861 -0.2541 ...
##  $ RatioSalesThisYrVsPrevYr      : num  -0.18 -0.1473 -0.18 -0.2077 -0.0913 ...
##  $ ADVICE                        : Factor w/ 3571 levels "X100002","X100003",..: 26 6 30 102 89 179 45
##  $ INVOICE                       : Factor w/ 3488 levels "X150062","X150063",..: 75 6 79 265 196 765 1
##  $ CUSTOMER                      : Factor w/ 1392 levels "X1203","X1219",..: 533 30 65 570 667 268 140
##  $ REP                           : Factor w/ 12 levels "X..","CC","JB",..: 1 1 1 1 1 1 1 1 5 1 ...
##  $ CHAIN_NO                      : Factor w/ 2604 levels "X15","X28","X29",..: 1 2 3 4 5 6 7 8 8 9 ..
##  $ Worthwhile                    : Factor w/ 2 levels "X0","X1": 1 2 2 2 2 2 2 2 2 2 ...
##  $ WholegoodSold_NEW_USED        : Factor w/ 2 levels "N","U": 1 1 1 2 1 1 1 2 2 1 ...
##  $ WholegoodSold_MachineTypeDesc: Factor w/ 35 levels "Bale.Sled","Baler.Round",..: 30 3 30 7 2 18 31
##  $ WholegoodSold_MODEL_CODE      : Factor w/ 1348 levels "X1.6.MTR","X10.TON",..: 209 70 1105 532 1237
##  $ WholegoodSold_MAKE_CODE       : Factor w/ 78 levels "AT","AW","AZ",..: 9 9 59 9 9 9 9 46 46 4 ...
```

```
##  $ TradeIn_NEW_USED             : Factor w/ 2 levels "N","U": 2 2 2 2 2 2 2 2 2 2 ...
##  $ TradeIn_MachineTypeDesc      : Factor w/ 37 levels "Bale.Sled","Baler.Round",..: 32 4 32 8 2 18 33
##  $ TradeIn_MODEL_CODE           : Factor w/ 2187 levels "X.TR250PS","X.ZAM.3000",..: 1222 119 1979 84
##  $ TradeIn_MAKE_CODE            : Factor w/ 97 levels "X..","AC","AT",..: 11 11 88 11 61 11 9 58 58 5
## NULL
```

Quick and Dirty Regression Model

Let's create a quick and dirty model on all the data (no training/testing subsets) to discover which columns are relevant and which are a distraction. It is normal practise to should consider only those where the probability of null hypothesis is $< 5\%$, but we'll widen the bracket to 10%, for now.

Two regression models will be created: 1. a linear regression model to consider the profit, in money, of the chain to which each deal belongs. 3. a logisitic regression model

Factors with almost as many levels as there are records in the data set will not be useful. However they will slow down the discovery of the optimal regression model, so they should be excluded:

ADVICE (col 31) INVOICE (col 32) CUSTOMER (col 33) CHAIN_NO (col 35) WholegoodSold_MODEL_CODE (col 39) TradeIn_MODEL_CODE (col 43)

Must also exclude final chain summaries, as these are not known at the time of an individual deal

CHAIN_SALES (col 9) CHAIN_PROFIT (col 10) CHAIN_MARGIN (col 11) CHAIN_QTY_WHOLEGOODS_SOLD (col 14)

For the analog model, the objective will be to calculate CHAIN_PROFIT_ADJ (col 17), exclude Worthwhile (col 18) For logisitic regression, the objective will be to calculate Worthwhile (col 18), exclude CHAIN_PROFIT_ADJ (col 18)

```
# filter down to columns for logistic regression
chains_analog  <- chains_all_scaled[,c(-31,-32,-33,-35,-39,-43,-9,-10,-11,-13, -36)]
chains_logit   <- chains_all_scaled[,c(-31,-32,-33,-35,-39,-43,-9,-10,-11,-13, -12)]

# create regression models
model_analog  <- lm(CHAIN_PROFIT_ADJ ~., data=chains_analog)

# select only the columns where prob(null hypothesis) < 10%
model_analog_usefulcols <- summary(model_analog)$coefficients[
                                                  summary(model_analog)$coefficients[,4]<=0.1,4]

# report results:
cat("\n")
```

```
paste0("Useful columns for analog regression")
```

```
## [1] "Useful columns for analog regression"
```

```
print(model_analog_usefulcols)
```

```
##    ADVICE_QTY_TRADEIN_BOUGHT                 ADVICE_SALES
##                 1.010460e-03                 1.878974e-10
##    ChainSeq_CumulativeProfit ChainSeq_CumulativeProfitAdj
##                6.266129e-198                 0.000000e+00
##       Debtor_OutstandingMth1       Debtor_OutstandingMth2
##                 8.329434e-02                 5.475401e-02
##            TradeIn_YEARS_OLD             TradeIn_Cost_Adj
##                 2.970954e-06                 6.703994e-15
```

```
##      ExpectedProfitFromChain                        REPMW
##                 0.000000e+00                 3.431277e-02
##    WholegoodSold_MAKE_CODEJD          TradeIn_MAKE_CODEMG
##                 8.334249e-02                 9.425826e-02
```

Independent Predictors

Note, some of these are NOT independent predictors. For example, ExpectedProfitFromChain is calculated using a linear model taking TradeIn_Cost_Adj as one of the inputs. Not ideal, but we seek only a simple, quick model, to understand what can be achieved using a very simple approach to the problem.

Create Linear Regression Model

Let's see how effective a model using this shortlist of predictors can be.

At this stage we simply want to see whether a regresion model can offer any advantage over guessing. That's not easy because guesing 'Worthwhile=1' every time will get 90% accuracy, such is the skewed data. With this as the objective we won't separate the data into training/test sets (that will come later).

We will simply attempt to see if our model, with all available data, can at least match a guess! If it can't, then the investigation can stop here and waste no more time!

It is understood that such a model will likely be over-trained, but we will refine the model later, using precautions to avoid contaminating our test set.

```
library(MASS)
model_analog_simplified  <- lm(CHAIN_PROFIT_ADJ ~
                               ADVICE_QTY_TRADEIN_BOUGHT + ADVICE_SALES +
                               ChainSeq_CumulativeProfit + ChainSeq_CumulativeProfitAdj +
                               TradeIn_YEARS_OLD + TradeIn_Cost_Adj + ExpectedProfitFromChain,
                               data=chains_analog)

model_analog_simplified_stepAIC <- stepAIC(model_analog_simplified, direction=c("both"), trace=F)
```

Linear Model Results

```
paste0("Analog model goodness of fit = ", summary(model_analog_simplified_stepAIC)$r.squared )
```

```
## [1] "Analog model goodness of fit = 0.920214196926565"
```

```
library(knitr)
kable(summary(model_analog_simplified)$coefficients)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0000000	0.0047314	0.000000	1.0000000
ADVICE_QTY_TRADEIN_BOUGHT	-0.0272988	0.0054838	-4.978044	0.0000007
ADVICE_SALES	0.1067044	0.0097283	10.968457	0.0000000
ChainSeq_CumulativeProfit	-0.5670214	0.0135352	-41.892378	0.0000000
ChainSeq_CumulativeProfitAdj	0.9212143	0.0077795	118.416174	0.0000000
TradeIn_YEARS_OLD	-0.0164765	0.0049925	-3.300255	0.0009755
TradeIn_Cost_Adj	-0.1876587	0.0089327	-21.007973	0.0000000
ExpectedProfitFromChain	0.6621011	0.0123104	53.783766	0.0000000

What we really want to know is whether the deal belongs to a chain which is profitable or loss making. We can do a confusion matrix on the data, applying a threshold to make the data binary (1 or 0, profit or loss)

```
library(caret)
confusionMatrix(as.factor(ifelse(model_analog_simplified_stepAIC$fitted.values
                      #multiply by sd and add the mean to revert to original scale
                      * CHAIN_PROFIT_ADJ_sd + CHAIN_PROFIT_ADJ_mean > 0, 1 ,0)),
                as.factor(ifelse(chains_analog$CHAIN_PROFIT_ADJ
                      #multiply by sd and add the mean to revert to original scale
                      * CHAIN_PROFIT_ADJ_sd + CHAIN_PROFIT_ADJ_mean > 0, 1, 0)))
```

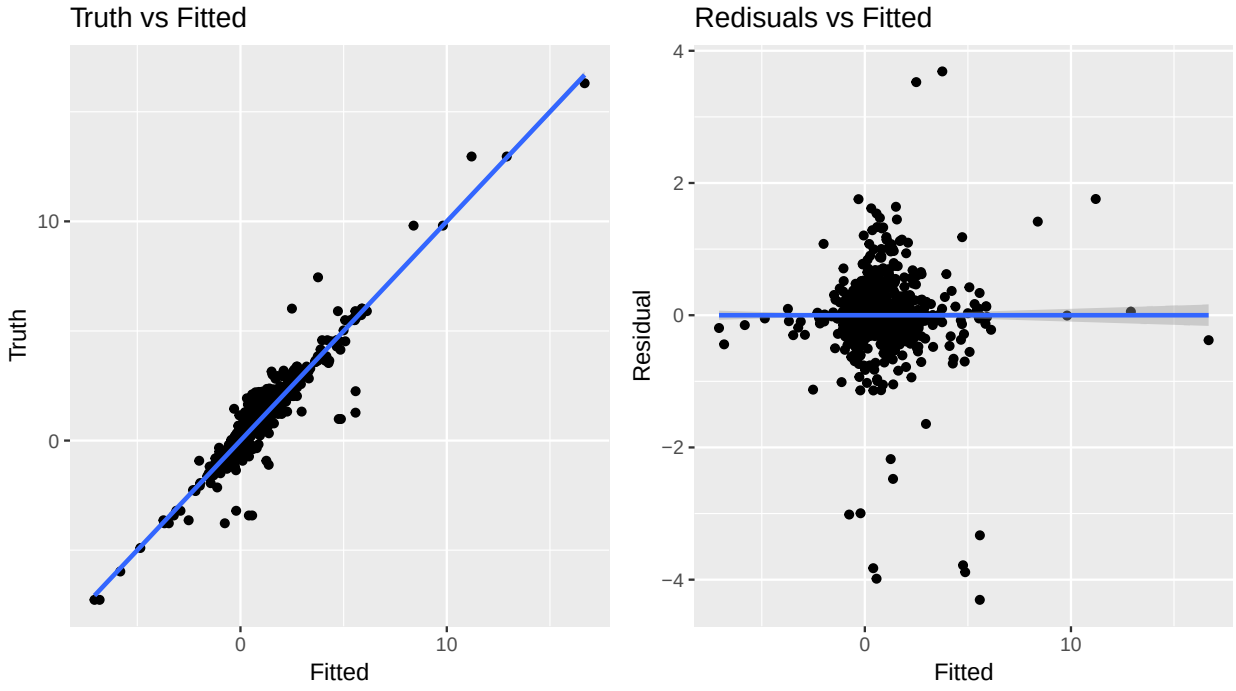
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  194   52
##          1  123 3202
##
##                Accuracy : 0.951
##                  95% CI : (0.9434, 0.9578)
##     No Information Rate : 0.9112
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.663
##  Mcnemar's Test P-Value : 1.213e-07
##
##             Sensitivity : 0.61199
##             Specificity : 0.98402
##          Pos Pred Value : 0.78862
##          Neg Pred Value : 0.96301
##              Prevalence : 0.08877
##          Detection Rate : 0.05433
##    Detection Prevalence : 0.06889
##       Balanced Accuracy : 0.79800
##
##        'Positive' Class : 0
##
```

Analyse Residuals for any Patterns

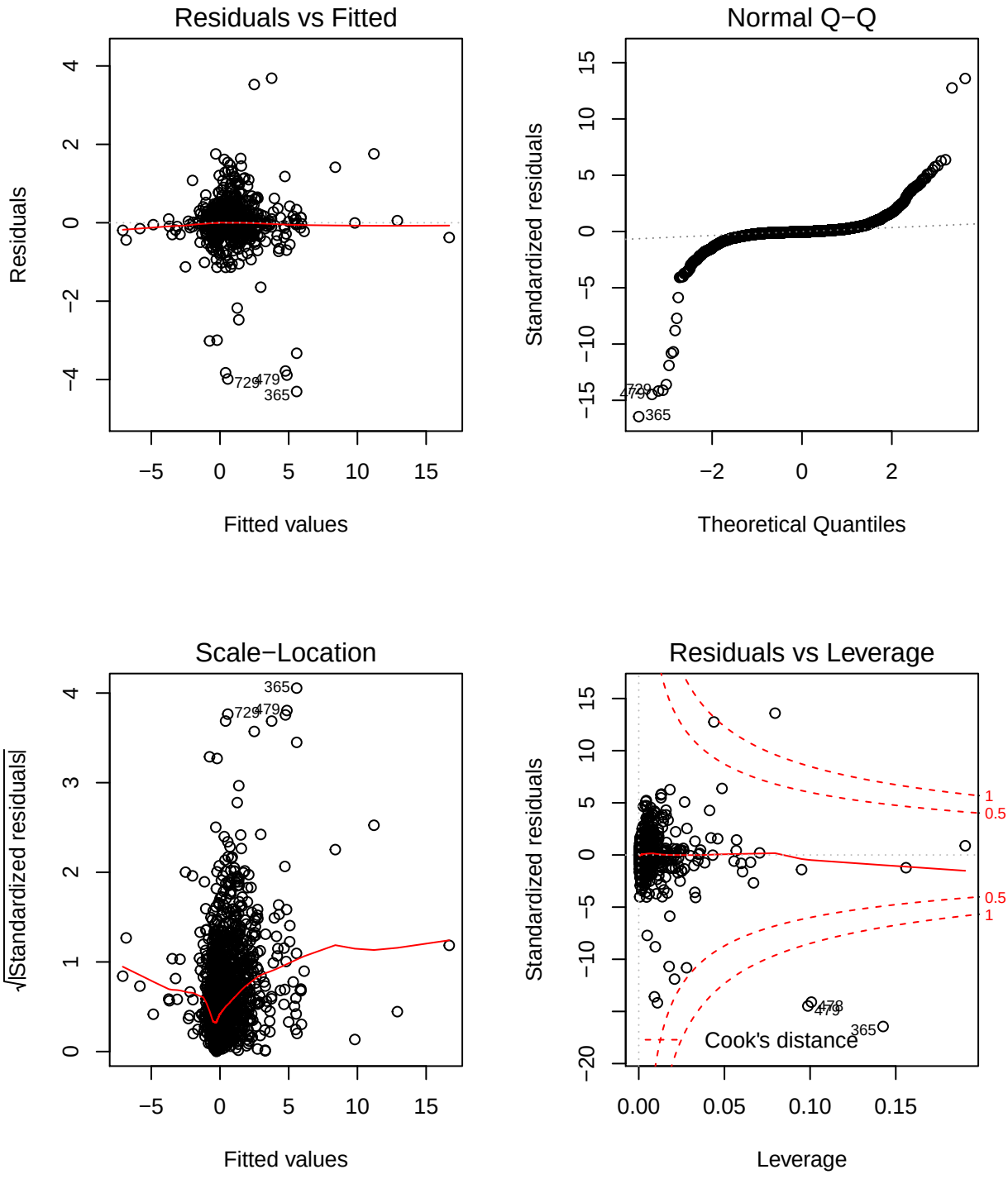
If there are patterns in the residuals then there may be important information our model is missing. The results show there is no obvious pattern to the residuals, but there may be some outliers to be analysed.

```
library(ggplot2)
library(gridExtra)

chartdata <- as.data.frame(cbind(model_analog_simplified_stepAIC$fitted.values,
                          chains_analog$CHAIN_PROFIT_ADJ,
                          model_analog_simplified_stepAIC$residuals))

g <- ggplot(chartdata, aes(x= chartdata[,1], y= chartdata[,2]))+
     geom_point()+
     geom_smooth(method="lm")+
     xlab("Fitted")+
```

```
     ylab("Truth")+
     ggtitle("Truth vs Fitted");

h <- ggplot(chartdata, aes(x= chartdata[,1], y= chartdata[,3]))+
     geom_point()+
     geom_smooth(method="lm")+
     xlab("Fitted")+
     ylab("Residual")+
     ggtitle("Redisuals vs Fitted");

grid.arrange(g, h, ncol=2);
```




```
par(mfrow = c(2,2))
plot(model_analog_simplified_stepAIC)
```



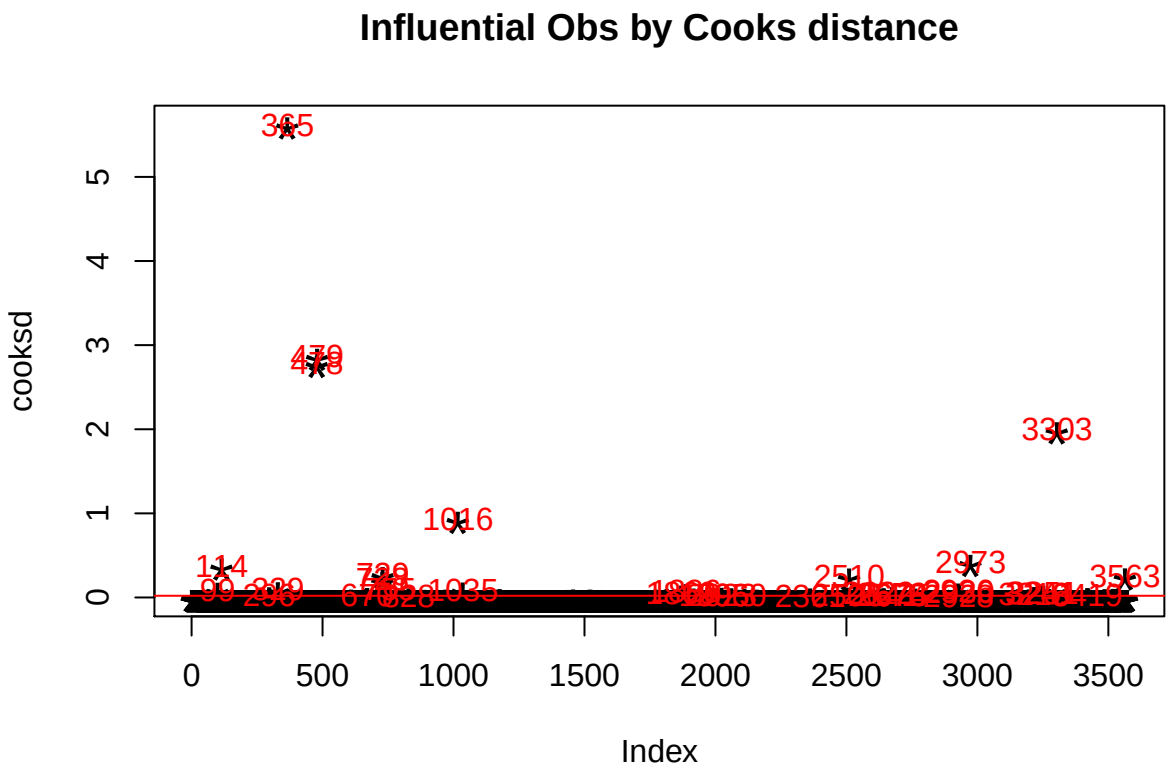
#Analyse Outliers

First using the car package's outlierTest (uses Bonferonni p-value for most extreme obs given the model) Then using the cook distance, both methods identify the same data points which cna be excluded from calculation of the model.

The top 10 outliers are identifed below.

```
library(car)
outlierTest(model_analog_simplified_stepAIC)
outliers <- as.numeric(attributes(outlierTest(model_analog_simplified_stepAIC)$p)$names)
```

```
cooksd <- cooks.distance(model_analog_simplified_stepAIC)
plot(cooksd, pch="*", cex=2, main="Influential Obs by Cooks distance")  # plot cook's distance
abline(h = 4*mean(cooksd, na.rm=T), col="red")  # add cutoff line
text(x=1:length(cooksd)+1, y=cooksd, labels=ifelse(cooksd>4*mean(cooksd, na.rm=T),
                                                   names(cooksd),""),
                                                   col="red")
```



Remove outliers from model, run new model on ALL data (inc outliers)

Our accuracy, before removing outliers is 95%, meaning we are only just better than a guess, which gets 91%. We still don't have formal test and train sets, we simply want to see if its possible, even with an over trained system, to beat a guess.

The below results show that excluding outliers from the training (but not from testing) raises the balanced accuracy (we are better at prediciting worthwhile=0), but not greatly improving overall accuracy.

```
chains_analog_V2<-chains_analog[-outliers,]

model_analog_simplified  <- lm(CHAIN_PROFIT_ADJ ~
                               ADVICE_QTY_TRADEIN_BOUGHT + ADVICE_SALES +
```

```
                              ChainSeq_CumulativeProfit + ChainSeq_CumulativeProfitAdj +
                              TradeIn_YEARS_OLD + TradeIn_Cost_Adj + ExpectedProfitFromChain,
                              data=chains_analog_V2)

model_analog_simplified_stepAIC <- stepAIC(model_analog_simplified, direction=c("both"), trace=F)

#apply confusion matrix to ALL data, inc outliers.
print(
confusionMatrix(as.factor(ifelse(predict(model_analog_simplified_stepAIC, newdata = chains_analog)
                        * CHAIN_PROFIT_ADJ_sd + CHAIN_PROFIT_ADJ_mean > 0, 1 ,0)),
                as.factor(ifelse(chains_analog$CHAIN_PROFIT_ADJ
                        * CHAIN_PROFIT_ADJ_sd + CHAIN_PROFIT_ADJ_mean > 0, 1, 0)))
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  230   67
##          1   87 3187
##
##                Accuracy : 0.9569
##                  95% CI : (0.9497, 0.9633)
##     No Information Rate : 0.9112
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7256
##  Mcnemar's Test P-Value : 0.1258
##
##             Sensitivity : 0.72555
##             Specificity : 0.97941
##          Pos Pred Value : 0.77441
##          Neg Pred Value : 0.97343
##              Prevalence : 0.08877
##          Detection Rate : 0.06441
##    Detection Prevalence : 0.08317
##       Balanced Accuracy : 0.85248
##
##        'Positive' Class : 0
##
```

Generalise the model using cross validation, train and test sets

In creating a simple model using all the data we have developed a better understanding of the data. We have also found that our model is better than guessing, if we had guessed 'worthwhile=1' every time then we would have missed all 317 of the deals belonging to loss making chains. Yet the model found 230 of those. with the promise of a useful model despite the skewed data, it is now worth taking a more robust approach.

Robust means ... we will test three competing models using k-fold validation on 75% of our data That 75% to be split into to 10 sets of train/validation data (in k-fold, k=10) It also means leaving outliers in the data. The effect of outliers should be diluted by k-fold anyway Finally, robust means leaving 25% of our data as a test set by which we compare the three models

We'll use the "caret" R-package to try regresion model and those mahcine learnign tools which are commonly used in Kaggle competitions. Recently these have tended to be tree based models. The full list of model are: linear regression (using step AIC to winnow out predictors) linear regression with lasso (using lasso to winnow out predictors) logistic regression (boosted version) random forest

Create test and train sets

```
#############################################################
#Split data into 2 chunks, training/validation and test

getdatasplit <- function(dataset, ProportionInTrain){

  ## Training set = proportion of the total data
  sample_size <- floor(ProportionInTrain * nrow(dataset))

  ## set the seed to make partition reproducible
  set.seed(25062018)
  training_indices <- sample(seq_len(nrow(dataset)), size = sample_size, replace = F)

  train <- dataset[ training_indices, ]
  test  <- dataset[-training_indices, ]

  return(list(train, test))
}

chains_analog_split <- getdatasplit(chains_analog, 0.75) #include all chains, do not exclude outliers
chains_analog_train <- chains_analog_split[[1]]
chains_analog_test  <- chains_analog_split[[2]]

chains_logit_split  <- getdatasplit(chains_logit, 0.75) #include all chains, do not exclude outliers
chains_logit_train  <- chains_logit_split[[1]]
chains_logit_test   <- chains_logit_split[[2]]
```

Set up the formulas (chosen predictors)

Each model should have access to the same predictors, these are defined in the 'formula', as stated below. The predictors include all those from the above model and some other hand picked candidates.

Cross validaiton is set to 10x

```
#we use cross validation throughout this paper. For ML models k=10, no repeat.
k <- 10

#later we will create an ensemble model so need to ensure same indices across all models...
 #for repeatedcv we would need times = ...
index_logit    <- createFolds(y=chains_logit_train$Worthwhile, k=k)

# Define train control for k fold cross validation
library(caret)
train_control_logit <- trainControl(method="cv",
                                    number=k,
                                    savePredictions = TRUE,
                                    classProbs = TRUE,
```

```
                                    summaryFunction = twoClassSummary,
                                    index = index_logit
                                    )

train_control_analog <- trainControl(method="cv",
                                     number=k,
                                     savePredictions = TRUE
                                     # no classProbs for analog
                                     # no index for analog because we don't do ensemble later
                                    )
#provide all the models with candidate predictors

formula_analog <- quote(CHAIN_PROFIT_ADJ ~ ADVICE_QTY_TRADEIN_BOUGHT + ADVICE_SALES +
                          ChainSeq_CumulativeProfit + ChainSeq_CumulativeProfitAdj +
                          TradeIn_YEARS_OLD + TradeIn_Cost_Adj + ExpectedProfitFromChain +
                          RatioSalesThisYrVsPrevYr + Debtor_OutstandingMth3Plus + REP +
                          DATE_SOLD_Month + TradeIn_MachineTypeDesc)

formula_logit <- quote(Worthwhile ~ ADVICE_QTY_TRADEIN_BOUGHT + ADVICE_SALES +
                         ChainSeq_CumulativeProfit + ChainSeq_CumulativeProfitAdj +
                         TradeIn_YEARS_OLD + TradeIn_Cost_Adj + ExpectedProfitFromChain +
                         RatioSalesThisYrVsPrevYr + Debtor_OutstandingMth3Plus + REP +
                         DATE_SOLD_Month + TradeIn_MachineTypeDesc)
```

Use Caret to Evaluate Multiple Models

So here we use the caret package to evaluate multiple types of model. Each is operated with 10x cross validation and default parameters, meaning no parameter optimisation has been attempted. This is irrelevant for models such as the Random Forest or Linear Regression which take no parameters, but arguably the Support Vector Machine and C5.0 would benefit from some optimisation.

The result is that the classification tree models outperform regression, although not by much. Note, when this experiemnt was run using non-scaled data, the C5.0 model outperformed others, getting higher F1 than any using scaled data.

```
# create the various models and evaluate them on the test set

getModelResults <- function (name, description, OutcomeIsBinary){

  # normally we'd use caret's trainControl() to ensure same data across all models
  # but here we're mixing logistic with analog models

  # get data, depends whether you want it scaled!
  if (OutcomeIsBinary) {
    formula   <- formula_logit
    traindata <- chains_logit_train
    testdata  <- chains_logit_test
    truth     <- chains_logit_test$Worthwhile
    sd        <- 1
    mean      <- 0
    tc        <- train_control_logit
    } else {
    formula   <- formula_analog
```

```
    traindata <- chains_analog_train
    testdata  <- chains_analog_test
    truth     <- chains_analog_test$CHAIN_PROFIT_ADJ
    sd        <- CHAIN_PROFIT_ADJ_sd
    mean      <- CHAIN_PROFIT_ADJ_mean
    tc        <- train_control_analog
    }

  start_time <- Sys.time()

  #here's the training command, note we use train_control, which is cross validation 10x
  set.seed(25062018)
  suppressWarnings(model <- train(method = name,
                                  eval(formula),
                                  data = traindata,
                                  trControl = tc,
                                  trace=F))
  end_time <- Sys.time()

  # evaluate on the test set
  if (OutcomeIsBinary) {
    model_testresult <- confusionMatrix(predict(model, newdata = testdata), truth)
    } else {
    model_testresult <- confusionMatrix(
                            as.factor(ifelse(predict(model,
                                                     newdata = testdata)*sd + mean > 0, 1 ,0)),
                            as.factor(ifelse(truth*sd + mean > 0, 1, 0)))
    }

  # record test results
  model_results <- list(description,                # Model name
                        model_testresult$overall[1],# Accuracy
                        model_testresult$byClass[7],# F1 Score
                        model_testresult$table[1,1],# True Positive (loss making)
                        model_testresult$table[1,2],# False Positive
                        model_testresult$table[2,2],# True Negative (profitable)
                        model_testresult$table[2,1],# False Negative
                        end_time - start_time       # Time elapsed
                        )

  results_ml[nrow(results_ml)+1,]  <<- model_results
  return(model)
}
```

```
results_ml <- data.frame(Model   = character(0),
                         Accuracy= numeric(0),
                         F1Score = numeric(0),
                         TruePos = numeric(0),
                         FalsePos= numeric(0),
                         TrueNeg = numeric(0),
                         FalseNeg= numeric(0),
                         SecondsSysTime= numeric(0), stringsAsFactors=F)

#Logistic models
```

```
model_lr <- getModelResults("LogitBoost", "Boosted Logistic Regression", OutcomeIsBinary = 1)
model_ca <- getModelResults("treebag", "Bagged CART", OutcomeIsBinary = 1)
model_rf <- getModelResults("rf", "Random Forest", OutcomeIsBinary = 1)
model_ad <- getModelResults("adaboost", "Ada Boost", OutcomeIsBinary = 1)
model_c5 <- getModelResults("C5.0", "C5.0", OutcomeIsBinary = 1)

#analog models mapped to logistic result
model_lo <- getModelResults("lasso", "Lasso Linear Regression", OutcomeIsBinary = 0)
model_lm <- getModelResults("glmStepAIC", "Generalised Linear Regression", OutcomeIsBinary = 0)
model_sv <- getModelResults("svmLinear2", "Support Vector Machine", OutcomeIsBinary = 0)
```

```
kable(results_ml[order(-results_ml$F1),])
```

	Model	Accuracy	F1Score	TruePos	FalsePos	TrueNeg	FalseNeg	SecondsSysTime
3	Random Forest	0.9619261	0.7702703	57	9	802	25	12.054049
5	C5.0	0.9619261	0.7671233	56	8	803	26	1.088332
4	Ada Boost	0.9608063	0.7586207	55	8	803	27	3.337432
6	Lasso Linear Regression	0.9596865	0.7428571	52	13	805	23	1.668056
7	Generalised Linear Regression	0.9574468	0.7397260	54	17	801	21	2.888465
2	Bagged CART	0.9552072	0.7333333	55	13	798	27	4.544947
1	Boosted Logistic Regression	0.9529675	0.7200000	54	14	797	28	5.219667
8	Support Vector Machine	0.9350504	0.6777778	61	44	774	14	22.164916

Investigation of the Random Forest (RF) Model

Since the RF appears at the top of the list let's spend some time investigating this model. First let's see how many predcitors and which predictors it uses most. Remember, the above models have: a. filtered the predictors to the shortlist found in the 'formula', as derived by analysing p values in linear regression b. some of those predictors are factors, eg REP. Each factor can be expanded into many predictors, eg REP is actually; REP_MW, REP_JP, REP_ND etc. This affects mtry, the number of predictors a random forest should use.

Firstly, let's create an RF model which is presented with all predictors and compare it with our model above

```
model_rf_allpreds <- train(Worthwhile ~.,
                           method = "rf",
                           data = chains_logit_train,
                           metric = "ROC",
                           trControl = train_control_logit,
                           trace=F)

# Standard model, as presented by caret with no 'tuning'
preds_rf_allpreds <- predict(model_rf_allpreds, chains_logit_test[,-27])
cm_rf_allpreds <- confusionMatrix(preds_rf_allpreds, chains_logit_test$Worthwhile)
```

```
paste0("Results from standard 'rf' model, no tuning")
```

```
## [1] "Results from standard 'rf' model, no tuning"
```

```
print(cm_rf_allpreds$table)
```

```
##           Reference
## Prediction  X0  X1
```

```
##         X0  58  12
##         X1  24 799
```

```
print(cm_rf_allpreds$byClass[7])
```

```
##        F1
## 0.7631579
```

Not much difference, in fact its marginally worse, so lets return to first RF model and view the most important predictors within that model.

Random Forest: Model Summary

Returning to the original, simpler RF model, let's get a summary...

```
print(model_rf)
```

```
## Random Forest
##
## 2678 samples
##   12 predictor
##    2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 267, 268, 268, 268, 268, 268, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens         Spec
##    2    0.8833481  0.003328147  1.0000000
##   29    0.9087311  0.561393499  0.9781098
##   57    0.8991711  0.589263485  0.9731327
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 29.
```

Random Forest: Most Important Predictors

We already know which predictors the linear regression thought most important. Let's see what predictors the original random forest found useful. Higher score = more important

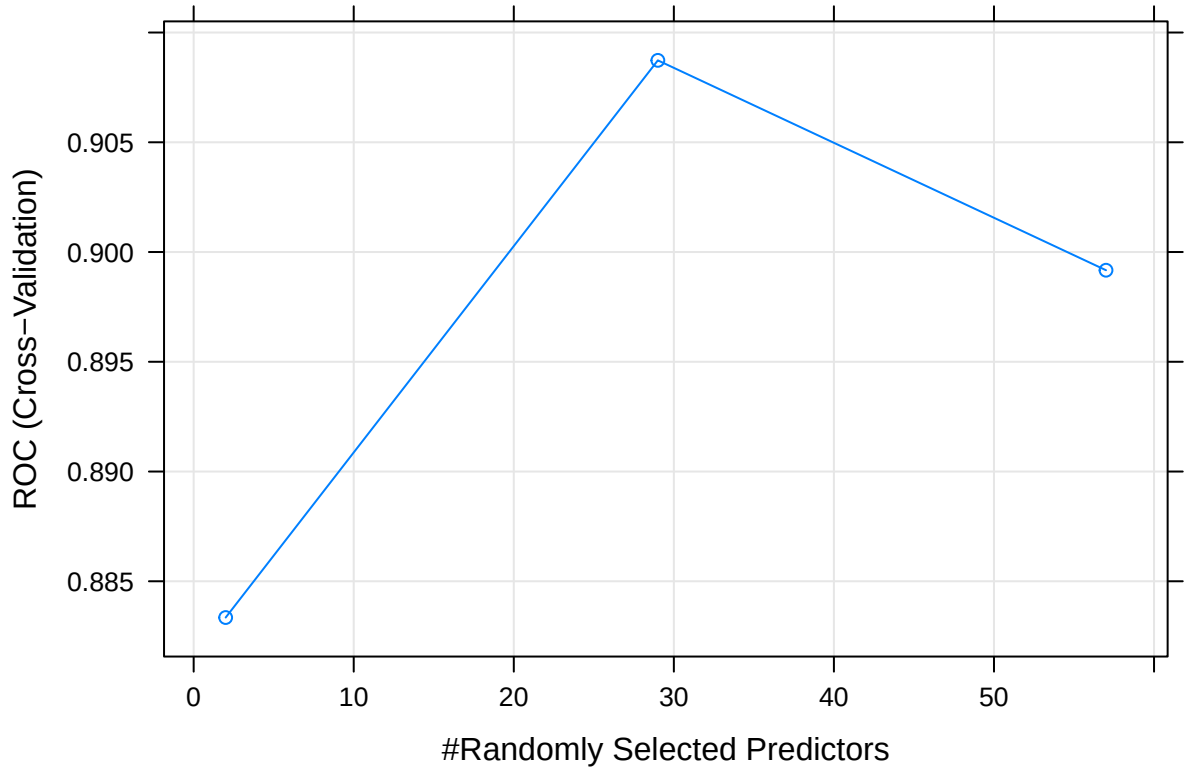
```
library(dplyr)
rf_important_vars <- varImp(model_rf)$importance %>%
                       as.data.frame() %>%
                         tibble::rownames_to_column() %>%
                           arrange(Overall) %>%
                             top_n(10) %>%
                               arrange(desc(Overall))
kable(rf_important_vars)
```

rowname	Overall
ChainSeq_CumulativeProfitAdj	100.000000
ExpectedProfitFromChain	92.292061
ChainSeq_CumulativeProfit	32.778822

rowname	Overall
TradeIn_Cost_Adj	21.437475
RatioSalesThisYrVsPrevYr	11.828276
ADVICE_SALES	10.828833
TradeIn_YEARS_OLD	6.970559
DATE_SOLD_Month	6.862413
Debtor_OutstandingMth3Plus	4.484070
REPJP	2.168244

We can also see how the number of predictors affects the model...

```
plot(model_rf)
```



Random Forest: Optimising the ROC with Skewed Dataset

The Random forest was the best model by F1 score. But as stated above, we want to capture as many True Positivies as possible and are willing to tolerate wading through some false negatives in order to find those true positives.

This is the common problem of tuning a model not for its internal paramters, but for the analysing a skewed data set such as ours. The creator of caret has standard code for exactly this problem at: https://topepo.github.io/caret/using-your-own-model-in-train.html#Illustration5

The following three code chunks are taken directly from that source, using a random forest. It plots all the options for setting the probability threshold at which the model indicates 'loss making'.

```
## Get the model code for the original random forest method:
thresh_code <- getModelInfo("rf", regex = FALSE)[[1]]
thresh_code$type <- c("Classification")

## Add the threshold as another tuning parameter
thresh_code$parameters <- data.frame(parameter = c("mtry", "threshold"),
                                     class     = c("numeric", "numeric"),
                                     label     = c("#Randomly Selected Predictors",
                                                   "Probability Cutoff"))
## The default tuning grid code:
thresh_code$grid <- function(x, y, len = NULL, search = "grid") {
  p <- ncol(x)
  if(search == "grid") {
    grid <- expand.grid(mtry = floor(sqrt(p)),
                        threshold = seq(.01, .99, length = len))
    } else {
      grid <- expand.grid(mtry = sample(1:p, size = len),
                          threshold = runif(1, 0, size = len))
      }
  grid
  }

## Here we fit a single random forest model (with a fixed mtry)
## and loop over the threshold values to get predictions from the same
## randomForest model.

thresh_code$loop = function(grid) {
  library(plyr)
  loop <- ddply(grid, c("mtry"),
                function(x) c(threshold = max(x$threshold)))
  submodels <- vector(mode = "list", length = nrow(loop))
  for(i in seq(along = loop$threshold)) {
    index <- which(grid$mtry == loop$mtry[i])
    cuts <- grid[index, "threshold"]
    submodels[[i]] <- data.frame(threshold = cuts[cuts != loop$threshold[i]])
    }
  list(loop = loop, submodels = submodels)
  }

## Fit the model independent of the threshold parameter
thresh_code$fit = function(x, y, wts, param, lev, last, classProbs, ...) {
  if(length(levels(y)) != 2)
    stop("This works only for 2-class problems")
  randomForest(x, y, mtry = param$mtry, ...)
  }

## Now get a probability prediction and use different thresholds to
## get the predicted class
thresh_code$predict = function(modelFit, newdata, submodels = NULL) {
  class1Prob <- predict(modelFit,
                        newdata,
                        type = "prob")[, modelFit$obsLevels[1]]
  ## Raise the threshold for class #1 and a higher level of
```

```
  ## evidence is needed to call it class 1 so it should
  ## decrease sensitivity and increase specificity
  out <- ifelse(class1Prob >= modelFit$tuneValue$threshold,
                modelFit$obsLevels[1],
                modelFit$obsLevels[2])
  if(!is.null(submodels)) {
    tmp2 <- out
    out <- vector(mode = "list", length = length(submodels$threshold))
    out[[1]] <- tmp2
    for(i in seq(along = submodels$threshold)) {
      out[[i+1]] <- ifelse(class1Prob >= submodels$threshold[[i]],
                           modelFit$obsLevels[1],
                           modelFit$obsLevels[2])
      }
    }
  out
  }

## The probabilities are always the same but we have to create
## mulitple versions of the probs to evaluate the data across
## thresholds

thresh_code$prob = function(modelFit, newdata, submodels = NULL) {
  out <- as.data.frame(predict(modelFit, newdata, type = "prob"))
  if(!is.null(submodels)) {
    probs <- out
    out <- vector(mode = "list", length = length(submodels$threshold)+1)
    out <- lapply(out, function(x) probs)
    }
  out
  }
```

Use the above functions...

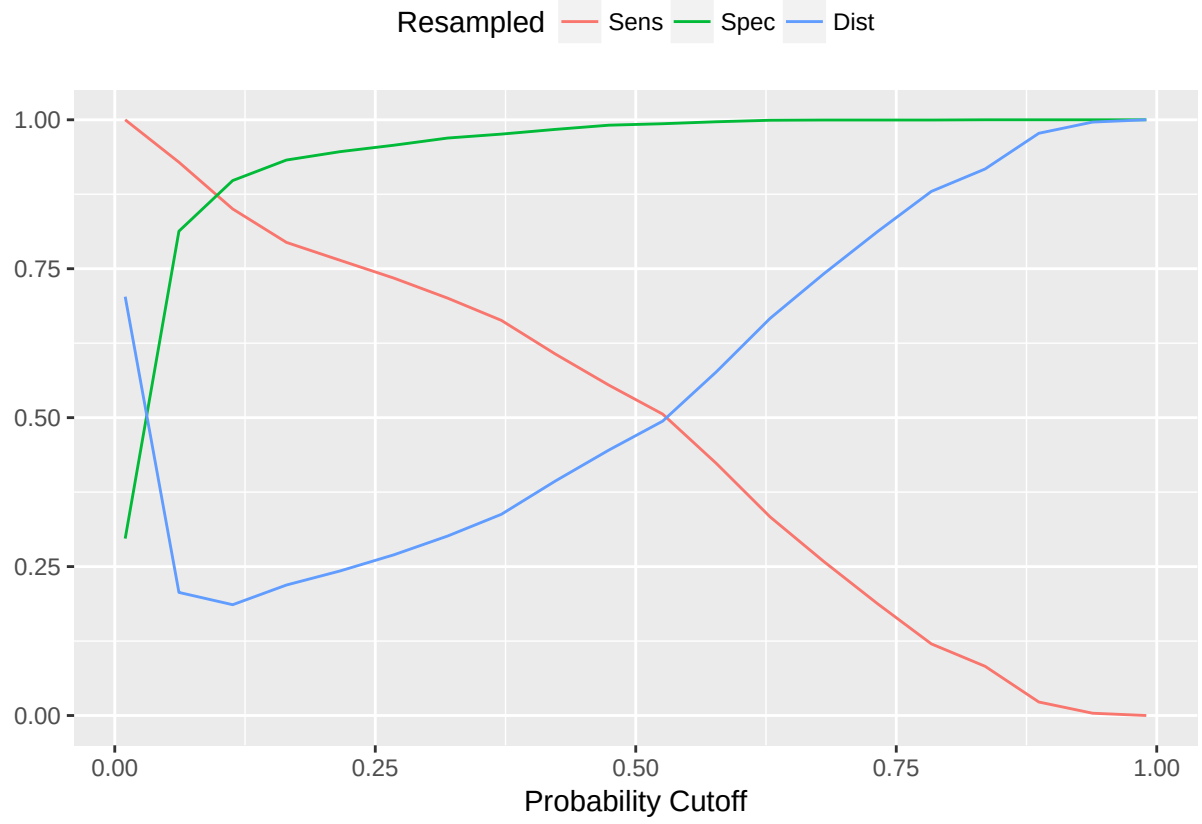
```
fourStats <- function (data, lev = levels(data$obs), model = NULL) {
  ## This code will get use the area under the ROC curve and the
  ## sensitivity and specificity values using the current candidate
  ## value of the probability threshold.
  out <- c(twoClassSummary(data, lev = levels(data$obs), model = NULL))

  ## The best possible model has sensitivity of 1 and specificity of 1.
  ## How far are we from that value?
  coords <- matrix(c(1, 1, out["Spec"], out["Sens"]),
                   ncol = 2,
                   byrow = TRUE)
  colnames(coords) <- c("Spec", "Sens")
  rownames(coords) <- c("Best", "Current")
  c(out, Dist = dist(coords)[1])
}

set.seed(949)
mod1 <- train(eval(formula_logit),
              data   = chains_logit_train,
              method = thresh_code,
```

```
              ## Minimize the distance to the perfect model
              metric = "Dist",
              maximize = FALSE,
              tuneLength = 20,
              trControl = trainControl(method = "cv",
                                       number = k,
                                       classProbs = TRUE,
                                       summaryFunction = fourStats))
```

Let's plot these results for each probability threshold (aka cut-off)

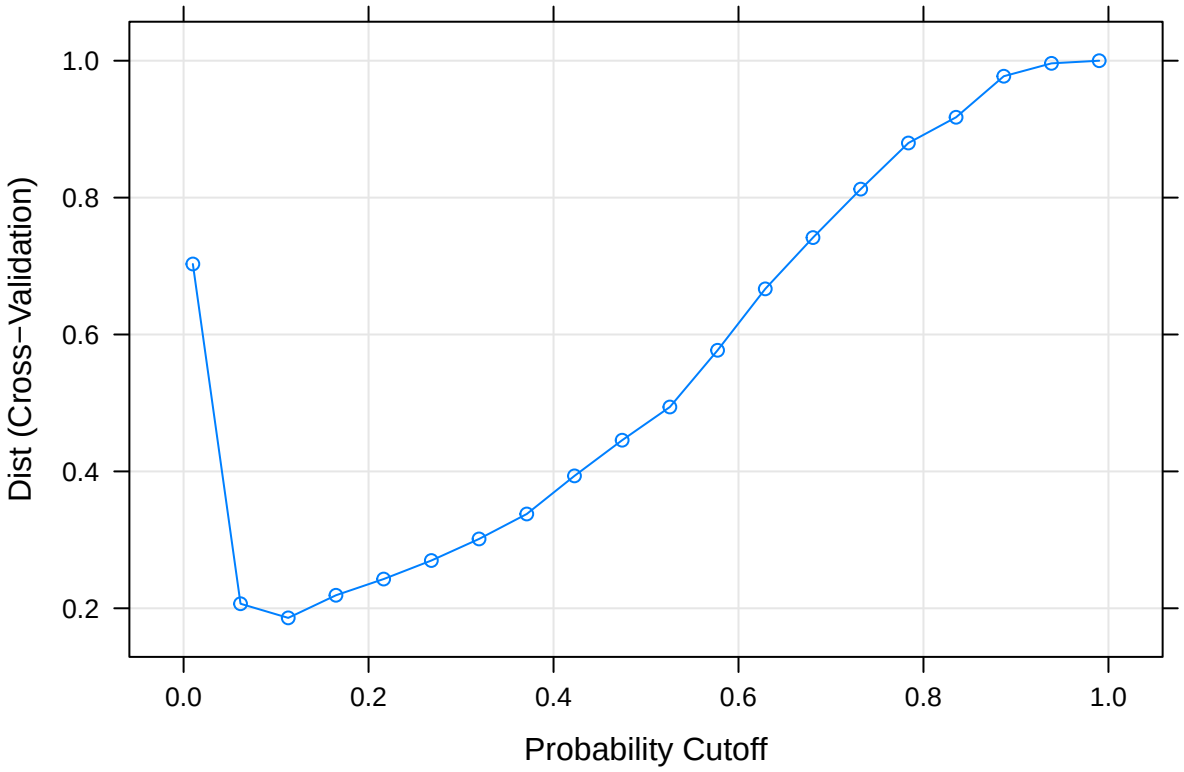
Red: Sensitivity = P(True Positive), i.e correctly identfied a loss making deal Green: Specificity = 1-P(False Positive), i.e incorrectly identified a loss making deal Blue: Distance to perfect model, lower is better

```
library(reshape2)
metrics <- mod1$results[, c(2, 4:6)]
metrics <- melt(metrics, id.vars = "threshold",
                variable.name = "Resampled",
                value.name = "Data")

ggplot(metrics, aes(x = threshold, y = Data, color = Resampled)) +
  geom_line() +
  ylab("") + xlab("Probability Cutoff") +
  theme(legend.position = "top")
```



We cna see how the threshold affects the distance to a perfectly accurate model by simply plotting:

```
plot(mod1)
```



Random Forest: Prediction Probability threshold at 0.075

To repeat from the SVM tuning above, "We want to catch as many loss making deals as possible without causing so many false positivies that we waste our time reviewing vast numbers of good deals. In other words, we want a high P(True Positive), aka sensitivity, and will tolerate a reasonable number of false positives, but not too many."

With this in mind, the above chart invites us to select a probability around 0.075 for the threshold between classifications. This has a low distance to the best model, but errs on the side of sensitivity.

The below code shows that moving the probability threshold to 0.075 increases the true positives from 57 to 74, but we increase false positives from 9 to 116! This may not be a good trade, but it depends on the financial cost of those loss making deals, not in remit of this investigation.

```
# Standard model, as presented by caret with no 'tuning'
rfProbs_std <- predict(model_rf, chains_logit_test[,-27])
cm_rf_std <- confusionMatrix(rfProbs_std, chains_logit_test$Worthwhile)

# Tuned model, changed probability threshold to achieve higher P(True Positive)
# understanding there will be more false positives
threshold <- 0.075
rfProbs <- predict(mod1, chains_logit_test[,-27], type = "prob")

#Remember, prob of X0 (factor level 2), i.e. deal is loss making
cm_rf <- confusionMatrix(factor(ifelse(rfProbs$X0>threshold,0,1)), #preds
                         factor(ifelse(chains_logit_test$Worthwhile=="X0",0,1))) #truth
```

```
paste0("Results from standard 'rf' model, no tuning")
```

```
## [1] "Results from standard 'rf' model, no tuning"
```

```
cat("\n")
```

```
paste0("Probability threshold set automatically for optimal model")
```

```
## [1] "Probability threshold set automatically for optimal model"
```

```
print(cm_rf_std$table)
```

```
##           Reference
## Prediction  X0  X1
##         X0  57   9
##         X1  25 802
```

```
print(cm_rf_std$byClass[7])
```

```
##        F1
## 0.7702703
```

```
cat("\n")
```

```
paste0("Results from tuned 'rf' model.")
```

```
## [1] "Results from tuned 'rf' model."
```

```
cat("\n")
```

```
paste0("Probability threshold set to 0.075.
        Achieves higher P(True Positive), but also more false positives")
```

```
## [1] "Probability threshold set to 0.075. \n        Achieves higher P(True Positive), but also more fa
```

```
print(cm_rf$table)
```

```
##           Reference
## Prediction   0   1
##          0  74 116
##          1   8 695
```

```
print(cm_rf$byClass[7])
```

```
##        F1
## 0.5441176
```

Random Forest: Sampling Weights

We can also use caret's 'weights' parameter to the train() function for handling our skewed data set. This allows us to 'up' sample a minority member, or 'down' sample it.

```
#repeated cross validation 10x3 for train control, with summaryfunction=twoclass for ROC optim.
train_control_rfweights <- trainControl(method  = "cv",
                                        number  = k,
                                        summaryFunction = twoClassSummary,
                                        classProbs = TRUE,
                                        allowParallel = TRUE)
```

```r
# build the tunegrid. mtry must be defined for tuning random forests
grid <- expand.grid(mtry = seq(from=1, to=model_rf$finalModel$mtry, by=3))#,
                    # NB for C5.0, these tuning grid options would be, as follows, mtry not req'd.
                    # .model  = "tree",
                    # .trials = c(1:100),
                    # .winnow = FALSE

set.seed(20180625)
model_rf_weights <- train(eval(formula_logit),
                          data      = chains_logit_train,
                          method    = "rf",
                          # for rf we could set the number of trees in forest,
                          #  but will let caret do it. ntree = 1000
                          tuneGrid  = grid,
                          metric = "ROC",
                          trControl = train_control_rfweights)
```

Evaluating the 'Standard Weights' Model

```r
# Evaluate the weighted 'up' model
preds_rf_weights  <- predict(model_rf_weights, newdata = chains_logit_test[,-27])
confMat_f_weights <- confusionMatrix(preds_rf_weights, chains_logit_test$Worthwhile)
print(confMat_f_weights$table)
```

```
##            Reference
## Prediction  X0  X1
##         X0  56   9
##         X1  26 802
```

```r
print(confMat_f_weights$byClass[7]) #F1 score
```

```
##        F1
## 0.7619048
```

Evaluating the 'Weighted Up' Model

```r
# evaluate on test set
preds_rf_weights   <- predict(model_rf_weights, chains_logit_test[,-27])
confMat_rf_weights <- confusionMatrix(preds_rf_weights, chains_logit_test$Worthwhile)

#We'll now
# Create model weights which sum to one
model_weights <- ifelse(chains_logit_train$Worthwhile == "Class1",
                        (1/table(chains_logit_train$Worthwhile)[1]) * 0.5, # level 1 is 'profitable'
                        (1/table(chains_logit_train$Worthwhile)[2]) * 0.5) # level 2 is 'unprofitable'

# Use the same seed to ensure same cross-validation splits as above sum model
train_control_rfweights_2 <- train_control_rfweights

#Set sampling = "up" for upsampling of the minority case, this is done to the train_control object
train_control_rfweights_2$sampling <- "up"
```

```
# Weighted model
set.seed(20180625)
model_rf_weights_up <- train(eval(formula_logit),
                             data      = chains_logit_train,
                             method    = "rf",
                             tuneGrid  = grid,
                             metric    = "ROC",
                             ## Here's the weights!
                             weights   = model_weights,
                             ## We've already set sampling = "up", which is the important change
                             trControl = train_control_rfweights_2)
```

```
# Evaluate the weighted 'up' model
preds_rf_weights_up  <- predict(model_rf_weights_up, newdata = chains_logit_test[,-27])
confMat_f_weights_up <- confusionMatrix(preds_rf_weights_up, chains_logit_test$Worthwhile)
print(confMat_f_weights_up$table)
```

```
##            Reference
## Prediction  X0  X1
##         X0  60  13
##         X1  22 798
```

```
print(confMat_f_weights_up$byClass[7]) #F1 score
```

```
##        F1
## 0.7741935
```

The results of the weighted up model align with a probability threshold of approx 0.3. It delivers a useful result but is less flexible than simply changing the threshold.

The Support Vector Machine

Having thoroughly considered the Random Forest model and satisfied ourselves that the original model is as good as can reasonably be, let's consider the worst model, the Support Vector Machine (SVM).

Although the SVM has the lowest F1 score, it has yet to be tuned. Furthermore, we want to catch as many loss making deals as possible without causing so many false positives (profitable, but flagged as loss making) that we waste our time reviewing vast numbers of good deals. In other words, we want a high P(True Positive), aka sensitivity, and will tolerate a reasonable number of false positives, but not too many.

The SVM identifies the highest number of True Positives, so may be worth tuning.

SVM's have a cost paramter 'Cost' or 'C' to be tuned, the below code attempts to do that.

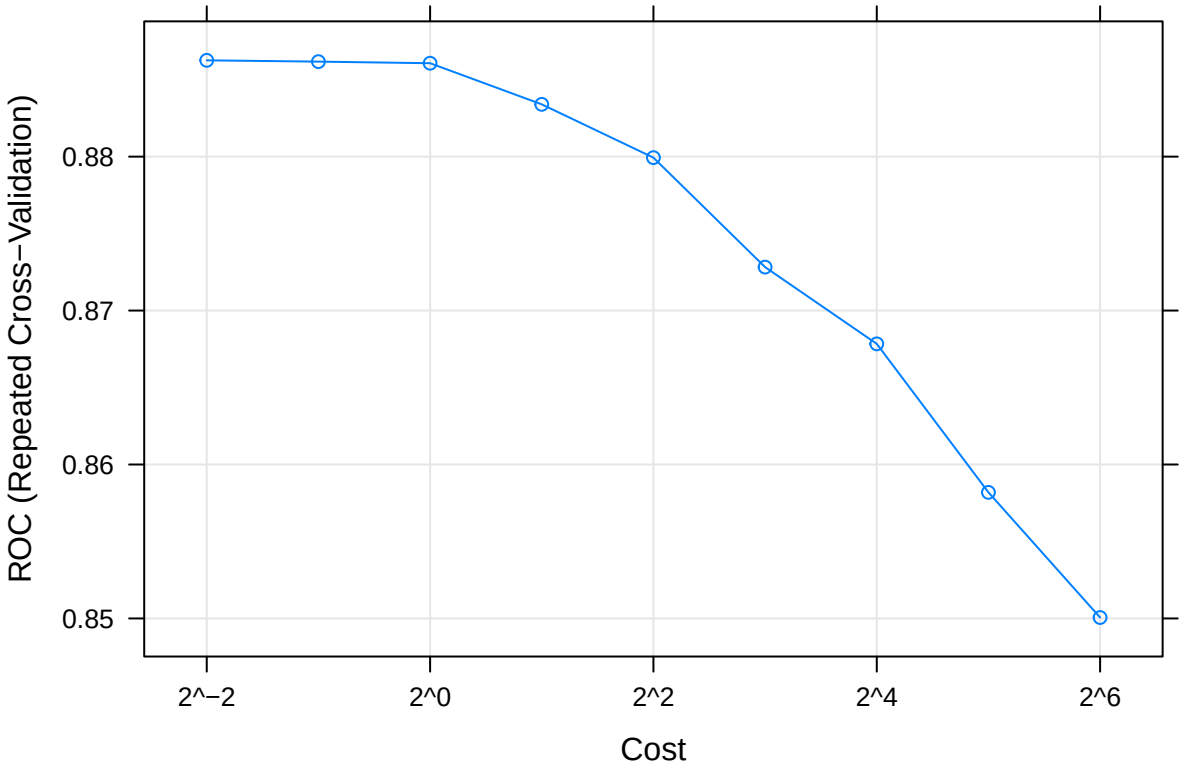
```
#We won't use the formula, we'll present all data.
#Before we start, remove constant columns ie max=min. The formula removed those in above train cycls
svm_train_data <- chains_logit_train
svm_test_data  <- chains_logit_test
constant_col   <- which(apply(svm_train_data,
                              MARGIN = 2,
                              function(x) max(x, na.rm = TRUE) == min(x, na.rm = TRUE)))
svm_train_data <- svm_train_data[,-c(constant_col,27)] # col 27 is 'Worthwhile', the outcome.
svm_test_data  <- svm_test_data[,-c(constant_col,27)]  # col 27 is 'Worthwhile', the outcome.

# SVM's need numerics, not factors
svm_train_data <- sapply(svm_train_data, function(x) as.numeric(x))
```

```r
svm_test_data  <- sapply(svm_test_data, function(x) as.numeric(x))

#repeated cross validation 10x3 for train control, with summaryfunction=twoclass for ROC optimisation
train_control_svm <- trainControl(method = "repeatedcv",
                                  number = k,
                                  repeats = 3,
                                  summaryFunction = twoClassSummary,
                                  classProbs = TRUE)

#train the svm
svmTune <- train(x = svm_train_data,
                 y = chains_logit_train$Worthwhile,
                 method = "svmLinear2",
                 # The default grid of cost parameters go from 2^-2,0.5 to 1
                 # We'll fit 9 values in that sequence via the tuneLength argument.
                 tuneLength = 9,
                 trControl = train_control_svm,
                 metric = "ROC")
```

```r
print(svmTune)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2678 samples
##   31 predictor
##    2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 2410, 2409, 2410, 2410, 2411, 2410, ...
## Resampling results across tuning parameters:
##
##   C      ROC        Sens       Spec
##    0.25  0.8862540  0.3309592  0.9850697
##    0.50  0.8861713  0.3309592  0.9852086
##    1.00  0.8860670  0.3222697  0.9881108
##    2.00  0.8833962  0.3235043  0.9886641
##    4.00  0.8799336  0.3298196  0.9871432
##    8.00  0.8728231  0.3049383  0.9894922
##   16.00  0.8678373  0.3099715  0.9910120
##   32.00  0.8581894  0.2985755  0.9918413
##   64.00  0.8500605  0.2795821  0.9919807
##
## Tuning parameter 'sigma' was held constant at a value of 0.03529632
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.03529632 and C = 0.25.
```

```r
#Plot ROC vs tuning paramter 'Cost'
plot(svmTune, metric = "ROC", scales = list(x = list(log =2)))
```



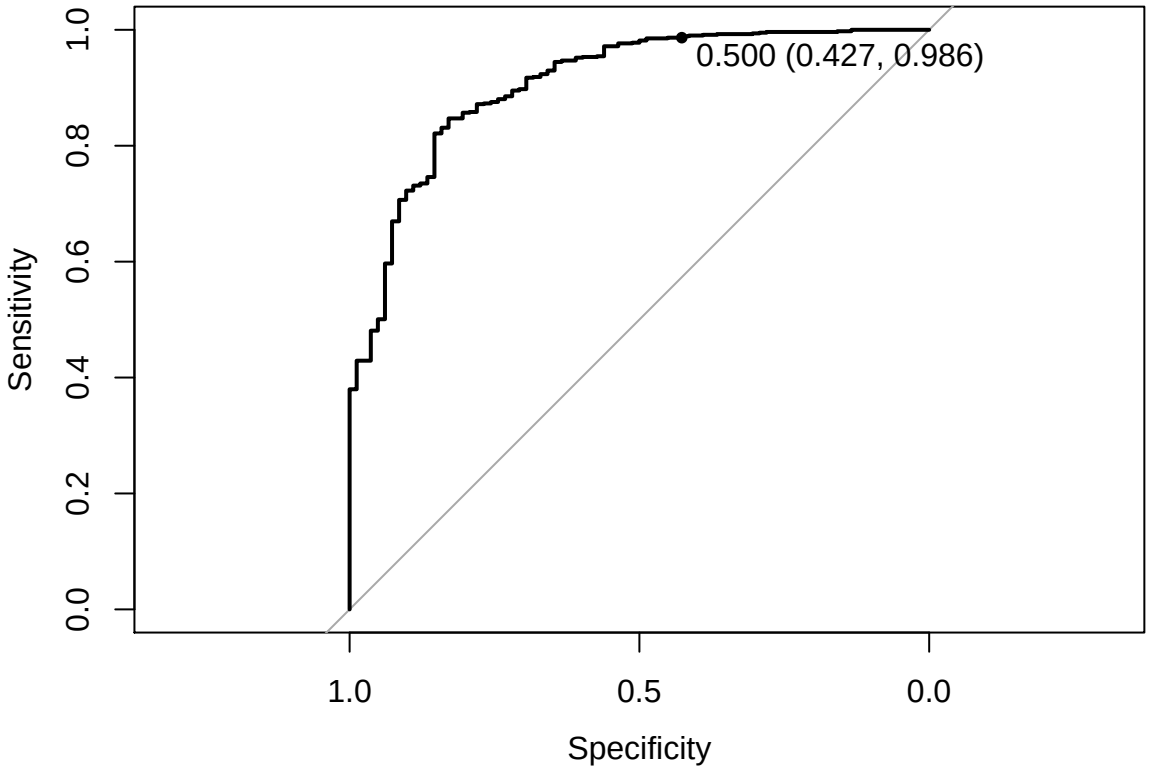
So the automated process chose the same value for 'Cost' that we would have, the value with highest sensitivity, 0.25.

SVM ROC Curve

ROC (Receiver Operated Characteristic) curves are used where the algorithm is expected to give a binary response, but actually provides a probability. Plotted as P(True Positive) vs 1-P(False Positive). i.e. Sensitivity vs Specificity.

We already know we can't tune the cost any further, but for completeness we'll plot the SVM ROC curve and calculate the area under that curve, AUC.

```
# Remember, we want to identify loss making deals, will tolerate false positive, but not false negative
library(pROC)
svmProbs <- predict(svmTune, svm_test_data, type = "prob")
svmROC   <- roc(chains_logit_test$Worthwhile, #outcome
                svmProbs[, "X0"],             #prob of X0 (factor level 2), i.e. deal is loss making
                levels = levels(chains_logit_train$Worthwhile)) # available levels in outcome factor

plot(svmROC, type = "S", print.thres = .5)
```




```
# Plots...
# Sensitivity = P(True  Positive), i.e correctly identfied a loss making deal
# 100% at top, by flagging all deals as loss making, hence many false positives.
# Specificity = 1-P(False Positive), i.e All LESS incorrectly identified a loss making deal
```

```
# We also need to calculate Area Under Curve (AUC)
# Remember, its a skewed dataset, so even guessing 'Profitable' every time will get high score
auc(svmROC)
```

```
## Area under the curve: 0.9068
```

More can be found on tuning at: https://www.r-project.org/conferences/useR-2013/Tutorials/kuhn/user_caret_2up.pdf

Ensemble: Bagging, Boosting and Stacking

We have used models that 'bag' (multiple configurations of the model operating in parallel to sample the data and average the result) and 'boost' (models trained to act in sequence, each model taking the output of its predecessor, to iteratively reduce error), but have not yet 'Stacked'. Stacking is where we train a model whose input is the output predicitons of a selection of other models. For the sake of completeness, let's see what results that gives.

We'll examine only categorical models, so we can use the caretEnsemble package more easily. First step is to see how correlated the models are.

Model Correlation

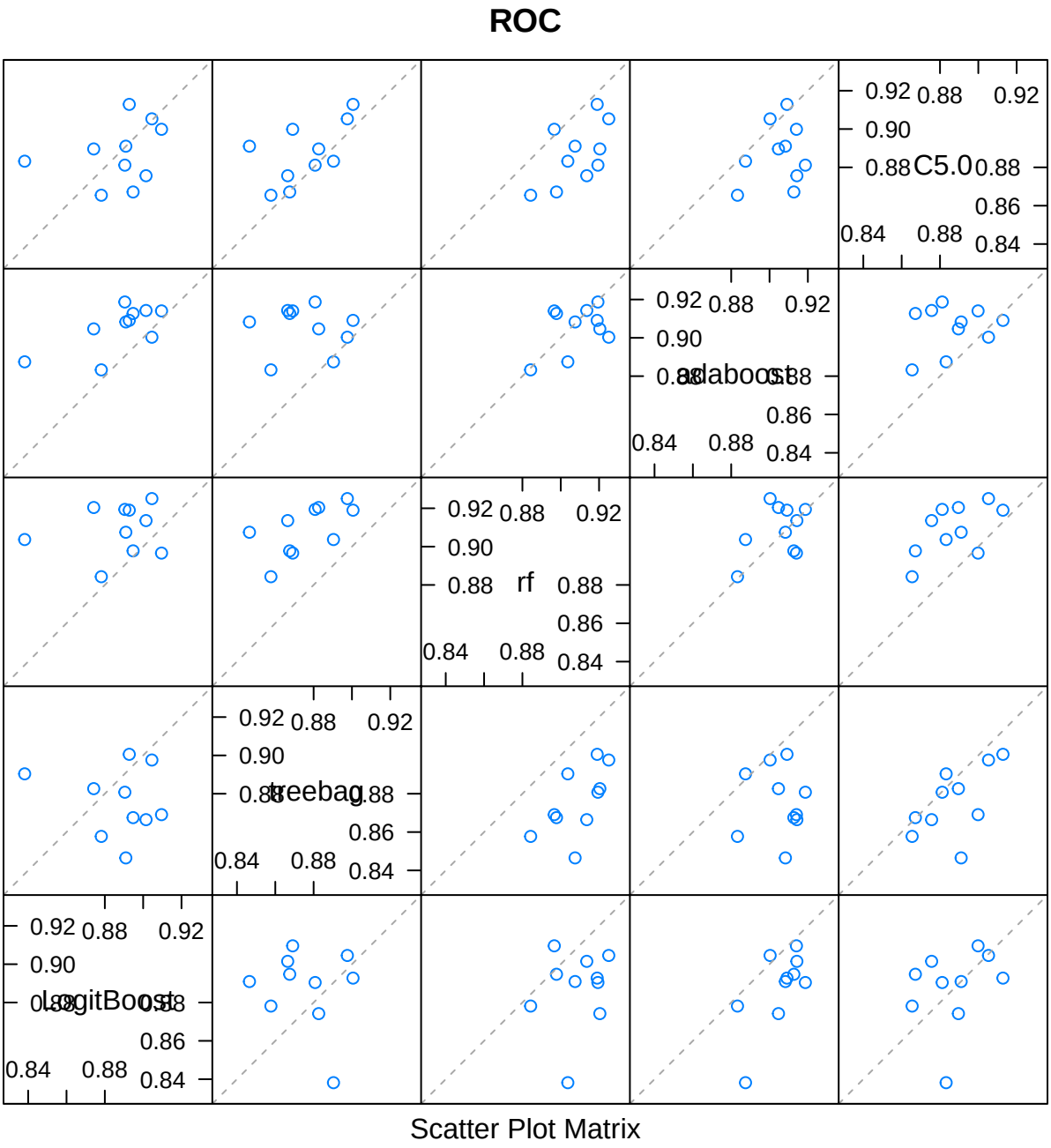
All correlations are quite low (greater than 0.75 is considered high) meaning that the models may each have their own perspective on the problem. Furthermore, each has a similar accuracy, so ensembling won't 'average down' the result. An ensemble model is worth analysing.

```
library(caretEnsemble)

#group the models into a list, MUST use caret names for each model, eg random forest = "rf"
models <- list(LogitBoost= model_lr,
               treebag   = model_ca,
               rf        = model_rf,
               adaboost  = model_ad,
               C5.0      = model_c5)

#resample for sake of correlation testing
resamps   <- resamples(models)

#view cross tab of the model correlations
modelCor(resamps)
```

```
##            LogitBoost     treebag        rf    adaboost      C5.0
## LogitBoost  1.0000000 -0.17520357 0.1670152  0.67003514 0.2687640
## treebag    -0.1752036  1.00000000 0.6194194 -0.05662106 0.5759076
## rf          0.1670152  0.61941942 1.0000000  0.41552335 0.5834986
## adaboost    0.6700351 -0.05662106 0.4155234  1.00000000 0.2070463
## C5.0        0.2687640  0.57590760 0.5834986  0.20704631 1.0000000
```

We can get another feel for the correlation by plotting the results, highly correlated models will result in straight lines (see dotted line in the below charts). This shows that there is clear some overlap between the models, but being none with 0.75 correlation means it is worthwhile pursuing the ensemble model.

```
splom(resamps)
```



Create Stacked Model

Let's train a C5 model to use the predictions made by the individual classification models. We will see that the ensemble can achieve F1 score of 0.7746, which is excellent but only minutely better than previously achieved with C5.0 and not any better than random forest on its own.

```
# the stack must have its own train control
train_control_stack <- trainControl(method = "boot",
                                    number = 10,
                                    savePredictions = "final",
```

```
                                    classProbs = TRUE,
                                    summaryFunction=twoClassSummary)

# before we can stack we must present the models as a "caretList"
# normally this is done by training the models within the caretList statement
# but because we built our logit models using a common index and claasProbs=T (in the train control)
# we can force our list of models into a caretList by simply re-designating it as such
class(models) <- "caretList"

#Train the ensemble model
set.seed(25062018)
stack <- caretStack(models,
                    method="C5.0",
                    metric="ROC",
                    trControl=train_control_stack)

# Get predictions
preds <- predict(stack, newdata = chains_logit_test)

#caretEnsemble BUG - known issue that caretEnsemble sometimes inverts predictions
#We have a highly skewed set, so easy problem to spot and rectify
if(sum(preds==levels(preds)[1])/length(preds) > 0.5) {
  #invert
  preds_inverted <- preds
  indexlist <- which(preds==levels(preds)[1])
  preds_inverted[indexlist]  <-  levels(preds)[2]
  preds_inverted[-indexlist] <-  levels(preds)[1]
  preds <- preds_inverted
  rm(preds_inverted)
}
```

```
#Evaluate the ensemble on the test set
stack_result <- confusionMatrix(preds, chains_logit_test$Worthwhile)
stack_result
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  X0  X1
##         X0  55   5
##         X1  27 806
##
##                Accuracy : 0.9642
##                  95% CI : (0.9498, 0.9754)
##     No Information Rate : 0.9082
##     P-Value [Acc > NIR] : 5.7e-11
##
##                   Kappa : 0.7557
##  Mcnemar's Test P-Value : 0.0002054
##
##             Sensitivity : 0.67073
##             Specificity : 0.99383
##          Pos Pred Value : 0.91667
##          Neg Pred Value : 0.96759
```

```
##              Prevalence : 0.09183
##          Detection Rate : 0.06159
##    Detection Prevalence : 0.06719
##       Balanced Accuracy : 0.83228
##
##        'Positive' Class : X0
##
```

```
stack_result$byClass[7] #F1 score
```

```
##        F1
## 0.7746479
```

Unfortunately the stacked (aka ensemble) model does not improve the F1 score much. So let's move to summarising our results from these statistical models.

Conclusion of Statistical Models

The orignal random forest model cannot easily be improved upon, although we could adjust the probability threshold a little. It offers F1 score of 0.77.

We can now consider deep learning alternatives to statistical models.

Deep Learning

We would be amiss not to attempt a neural network. Here we will use Keras on the Tensorflow engine. This is a locally installed instance using a GPU.

Since we have only a few thousand records we will need a small network, else risk over-fitting. A simple model with two hidden layers, each of 64 units is a standard way to start the exploration.

Prep the data

Deep learning requires data is presented in a 'tensor' (a matrix) of numeric values. With Tensorflow as the backend they need to be float32. No integers or factors. Fields which we want to use as training but which were presented to the above models as factors need to be in 'one hot' format. We also need to separate the data into test and train plus separate the outcome (aka 'labels') into vectors of their own.

```
# Convert categorical data to one-hot vectors
# For example, REP is expanded into 12 columns, one for each rep.
library(tidyr)
library(dplyr)
onehot <- function(dataframe, column_to_one_hot){
  dataframe %>%
    mutate(i=1) %>%
      spread(key=eval(column_to_one_hot), value=i, fill=0)
}

chains_logit_onehot <- onehot(chains_logit, quote("REP"))
chains_logit_onehot <- onehot(chains_logit_onehot, quote("TradeIn_MachineTypeDesc"))
chains_logit_onehot <- onehot(chains_logit_onehot, quote("WholegoodSold_NEW_USED"))

#remove other factors as unhelpful (eg, due too many factors, hence too few examples per factor)
```

```
chains_logit_onehot <- chains_logit_onehot %>%
                          select(-WholegoodSold_MAKE_CODE, -TradeIn_MAKE_CODE,
                                 -TradeIn_NEW_USED, -WholegoodSold_MachineTypeDesc)

#split into training and test and convert to numeric matrix or vector, as required for Tensors
split <- getdatasplit(chains_logit_onehot, ProportionInTrain = 0.75)
train_data   <- as.matrix(split[[1]] %>% select(-Worthwhile))
train_labels <- as.vector(as.numeric(split[[1]]$Worthwhile))
train_labels <- train_labels -1 # so 1 and 0, else 1 and 2
test_data    <- as.matrix(split[[2]] %>% select(-Worthwhile))
test_labels  <- as.vector(as.numeric(split[[2]]$Worthwhile))
test_labels  <- test_labels -1 # so 1 and 0, else 1 and 2
```

Before we start we should confirm that the installation of keras/tensorflow is using the local GPU. We want the following processing to be as fast as possible, althouhg most the time will be consumed with k-fold, which does not improve with GPU use.

```
# confirm the installation is using GPU
library(tensorflow)
with(tf$device("/gpu:0"), {
  const <- tf$constant(42) #because 42 is the answer to life, the universe and everything.
})
sess <- tf$Session()
sess$run(const)
```

```
## [1] 42
```

The Deep Learning Model

To start we'll use a simple neural network of two hidden dense (ie fully connected) layers. Activation between layers will be ReLu for simplicity and final activiation will be sigmoid for a binary outcome. Suhc a configuration should handle most non-linear problems.

The question of how many units we should have in the hidden layers is of interest. We don't want to lose information and there are 76 columns of data, so we want at least 76 units. A rule of thumb says start with 1.5x, so let's use 128.

```
library(keras)
#install_keras(tensorflow = "gpu")

build_model_0 <- function(hiddenlayer_units){
  #specify the graph
 the_model <- keras_model_sequential() %>%
   layer_dense(units = hiddenlayer_units, activation = "relu", input_shape = ncol(train_data)) %>%
   layer_dense(units = hiddenlayer_units, activation = "relu") %>%
   layer_dense(units = 1, activation = "sigmoid") #need sigmoid for 0 to 1 outcome.

 #compile the model
 the_model <- the_model %>% compile(optimizer = "rmsprop",
                                    loss = "binary_crossentropy",
                                    metrics = c("mae", "categorical_accuracy"))
 return(the_model)
}
```

K-Fold Validation

As with the models above, K-fold validation is used to make the most of the limited data.

```
###############################################################
# Function to train model
###############################################################

kfold_train <- function(k, num_epochs, training_data,
                        training_labels, hiddenlayer_units,
                        build_model)
                        #Note, build_model is a function.
  {

  #  folds
  indices <- sample(1:nrow(training_data))
  folds   <- cut(indices, breaks=k, labels = FALSE)

  #  epochs
  all_scores <- c()
  all_mae_histories <- data.frame()

  # Start the clock!
  ptm <- proc.time()

  for (i in 1:k) {
    cat("procesing fold #", i, "\n")

    val_indices <- which(folds == i, arr.ind = TRUE)
    val_data    <- training_data[val_indices, ]
    val_targets <- training_labels[val_indices]

    partial_train_data    <- training_data[-val_indices,]
    partial_train_targets <- training_labels[-val_indices]

    the_model <- build_model(hiddenlayer_units)

    #train the model (silent mode, verbose=0)
    history <- the_model %>% fit(partial_train_data,
                                 partial_train_targets,
                                 epochs = num_epochs,
                                 batch_size = 1,
                                 verbose = 0)

    mae_history <- history$metrics$mean_absolute_error
    all_mae_histories <- rbind(all_mae_histories, mae_history)
  }
  duration <- proc.time() - ptm

  # Stop the clock
  paste0("Time to process all folds, all epochs = ")
  cat("\n")
  duration
```

```
  return(list(the_model, all_mae_histories, duration))
}

#################################################################
# Function to chart how model performance develops with each epoch
#################################################################

plot_progress <-function(all_mae_histories, num_epochs){
  #chart progress
  average_mae_history <- data.frame(epoch = seq(1:num_epochs),
                                    validation_mae = apply(all_mae_histories,2,mean))
  library(ggplot2)
  ggplot(average_mae_history, aes(x=epoch, y=validation_mae)) + geom_line()
}

#################################################################
# Function to evaluate model on the test data set
#################################################################

evaluate_model <- function(model, testing_data, testing_labels){
  # Evaluate on test data
  result <- model %>% evaluate(testing_data, testing_labels)

  # prediction
  pred <- model %>% predict(testing_data, batch_size = 128)

  # The model returns decimal values, not integers, so need to convert using round()
  test_labels_pred = round(pred)

  # Confusion matrix
  library(caret)
  confusionMatrix(as.factor(test_labels_pred), as.factor(testing_labels))
}
```

Now let's execute our model with 128 neurons per hidden layer and a whopping 300 epochs, let's see if we over train or simply get to a solid result. We arrived at 128 by using a rule of thumb, neurons = 1.5x inputs.

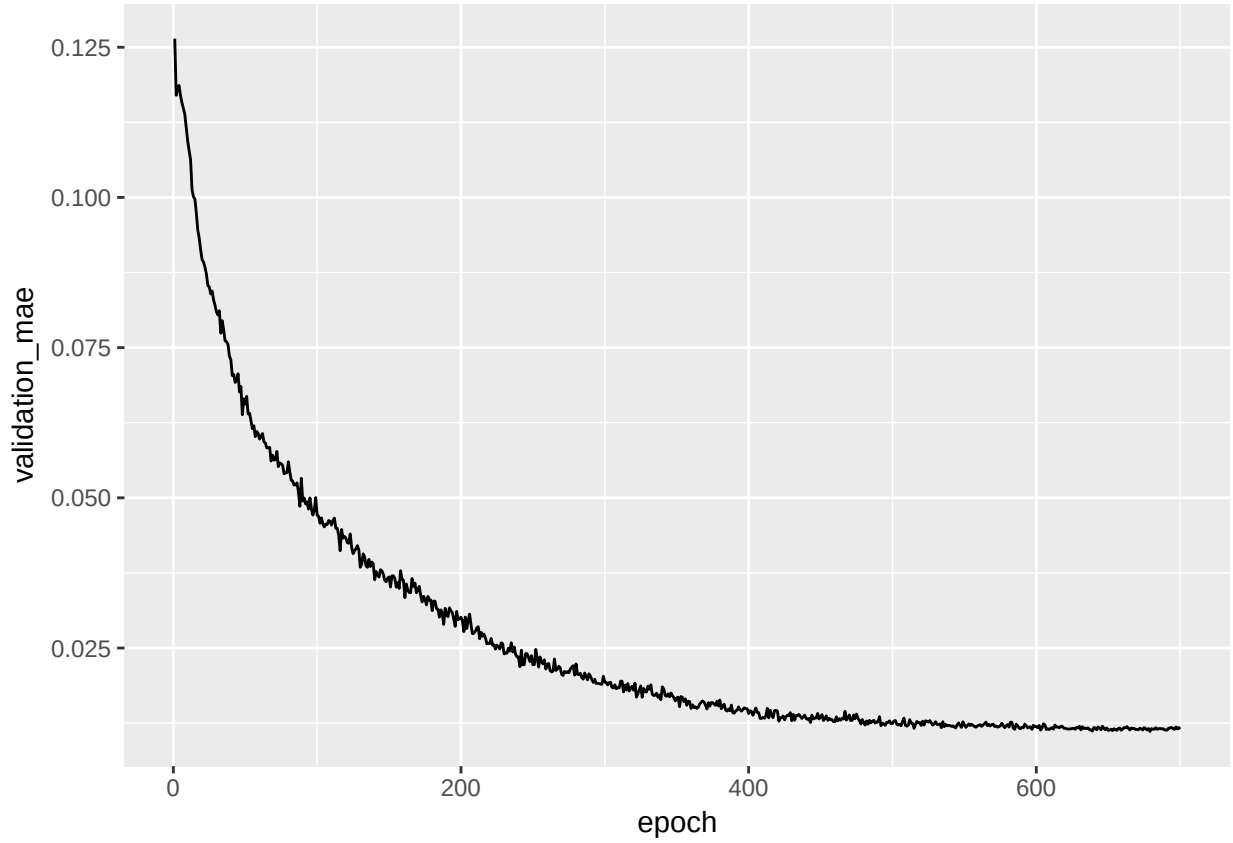
```
hiddenlayer_units <- 128
num_epochs <- 700
k <- 5

model0_list      <- kfold_train(k, num_epochs, train_data, train_labels,
                                hiddenlayer_units, build_model_0)
model0_model     <- model0_list[[1]]
model0_histories <- model0_list[[2]]
model0_duration  <- model0_list[[3]]

# save keras model, must serialise, else model will not be available to next R session.
# returns an R "raw" object containing an hdf5 version of the Keras model.
# unserialize_model() returns a Keras model.
model0_serialized<- serialize_model(model0_model, include_optimizer = TRUE)
```

```
print(model0_duration)
```

```
##     user   system  elapsed
```

```
## 31087.35  3052.07 31673.44
```

```
plot_progress(model0_histories, num_epochs)
```



Deep Learning, Model 0 Test Results

The mode has converged, let's see what accuracy metrics it can achieve on the test data (held out from any part of the model building process)...

```
model0_testresults <- evaluate_model(model0_model, test_data, test_labels)
model0_testresults$table
```

```
##            Reference
## Prediction   0   1
##          0  52  70
##          1  30 741
```

```
cat("\n")
```

```
model0_testresults$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##           0.63414634           0.91368681           0.42622951
##       Neg Pred Value            Precision               Recall
##           0.96108949           0.42622951           0.63414634
##                   F1           Prevalence       Detection Rate
##           0.50980392           0.09182531           0.05823068
```

```
## Detection Prevalence    Balanced Accuracy
##           0.13661814           0.77391657
```

Deep Learning Model 1, Go Deeper!

So despite 9.1hrs of training (elapsed time = 32,762s = 9.1hrs), the deep learning model is not the best we have. Is it not deep enough? Let's try three hidden layers. This will lead to far more parameters to train which may cause over-training. So drop out has been added as a simple form of regularisation.

```
build_model_1 <- function(hiddenlayer_units){
  #specify the graph
 the_model <- keras_model_sequential() %>%
   layer_dense(units = hiddenlayer_units, activation = "relu", input_shape = ncol(train_data)) %>%
   layer_dropout(rate = 0.5) %>%
   layer_dense(units = hiddenlayer_units, activation = "relu") %>%
   layer_dropout(rate = 0.5) %>%
   layer_dense(units = hiddenlayer_units, activation = "relu") %>%
   layer_dropout(rate = 0.5) %>%
   layer_dense(units = 1, activation = "sigmoid") #need sigmoid for 0 to 1 outcome.

 #compile the model
 the_model <- the_model %>% compile(optimizer = "rmsprop",
                                    loss = "binary_crossentropy",
                                    metrics = c("mae", "categorical_accuracy"))
 return(the_model)
}
```

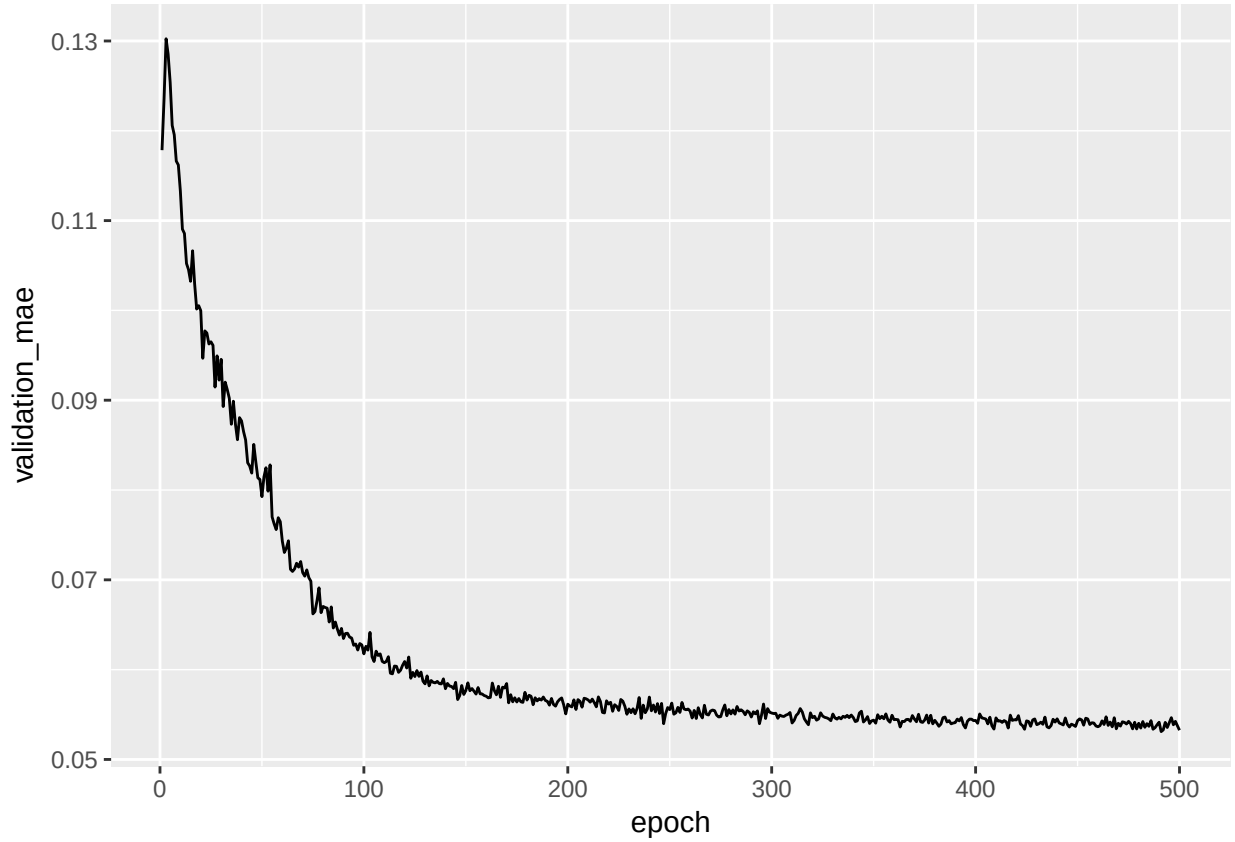
Now let's execute that model...

```
hiddenlayer_units <- 128
num_epochs1 <- 500
k <- 5

model1_list      <- kfold_train(k, num_epochs1, train_data, train_labels,
                                hiddenlayer_units, build_model_1)
model1_model     <- model1_list[[1]]
model1_histories <- model1_list[[2]]
model1_duration  <- model1_list[[3]]

# save keras model, must serialise, else model will not be available to next R session.
# returns an R "raw" object containing an hdf5 version of the Keras model.
# unserialize_model() returns a Keras model.
model1_serialized<- serialize_model(model1_model, include_optimizer = TRUE)
```

```
print(model1_duration)
```

```
##     user   system  elapsed
## 26003.35  2238.45 24702.18
```

```
plot_progress(model1_histories, num_epochs1)
```



Deep Learning, Model 1 Test Results

Again, the model has converged, although to a value of validation_mae which appears higher than the first model. Let's try the model on the test data:

```
model1_testresults <- evaluate_model(model1_model, test_data, test_labels)
model1_testresults$table
```

```
##            Reference
## Prediction   0   1
##          0  25  12
##          1  57 799
```

```
cat("\n")
```

```
model1_testresults$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##           0.30487805           0.98520345           0.67567568
##       Neg Pred Value            Precision               Recall
##           0.93341121           0.67567568           0.30487805
##                   F1           Prevalence       Detection Rate
##           0.42016807           0.09182531           0.02799552
## Detection Prevalence    Balanced Accuracy
```

```
##           0.04143337           0.64504075
```

The F1 score is worse than the first, yet simpler, deep learning model.

Conclusion

More time could be invested in exploring deep learning architectures, but time is pressing. Proceed with the Random Forest model, or find more data for the model to use.

APPENDIX: SQL for data extract from ERP/Accounting System

Extract Step 1: Deals and chains

The key section of the first step of SQL, linking deals into chains, is an 'anchor recursive' query which links deals (deals are known as an 'advice' in the accounting system) together chains of deals using the logic that the trade-in from one deal becomes the machine being sold in the next deal of a chain.

Note, each chain can have many deals (aka advices). Each advice can have many machines (aka wholegoods) sold and many traded-in. In other words, there is a

one-to-many relationship between machines and deals, where the machine is sold one-to-many between machines and advices, where the machine is received as a trade-in.

Note, a single machine (defined by a 'wholegood_no') will appear on multiple advices, once as a trade-in and once as a sale.

Also, consider a deal in which there are two machines received as trade-ins. The trade-ins are likely to be sold separately at a later date. Perhaps each of these subsequent deals receives a trade-in in turn. The original deal is now the parent to multiple chains. Therefore, we have a

many-to-many relationship between deals and chains

```
--Prepare temp tables

IF OBJECT_ID(N'tempdb..#tmp_AllChains', N'U') IS NOT NULL
DROP TABLE [#tmp_AllChains];

--commence query

WITH
TRADEINS
AS
(
    SELECT
        WHOLEGOOD_SOLD_DETAILS.WHOLEGOOD_NO AS WHOLEGOOD_SOLD,
        WHOLEGOOD_SOLD_DETAILS.[STATUS] AS WHOLEGOOD_SOLD_STATUS,
        WHOLEGOOD_SOLD_DETAILS.SALES_ADVICE AS WHOLEGOOD_SOLD_ONADVICE,
        WHOLEGOOD_SOLD_DETAILS.DATE_SOLD AS WHOLEGOOD_SOLD_DATE,
        WHOLEGOOD_SOLD_DETAILS.SoldToRef AS WHOLEGOOD_SOLD_CUST,
        COALESCE(PUBLIC_WGOOD_SALES_INV_TRADEIN.WHOLEGOOD_NO,
                 'END_'+ WHOLEGOOD_SOLD_DETAILS.WHOLEGOOD_NO)
             AS TRADE_IN_RECEIVED,
        TRADEIN_DETAILS.[STATUS] AS TRADE_IN_STATUS,
        TRADEIN_DETAILS.SALES_ADVICE AS TRADEIN_SOLD_ONADVICE
```

```
    FROM
        [11_WholegoodsStockUnion] AS WHOLEGOOD_SOLD_DETAILS

        LEFT OUTER JOIN
        PUBLIC_WGOOD_SALES_INV_TRADEIN
        ON WHOLEGOOD_SOLD_DETAILS.SALES_ADVICE = PUBLIC_WGOOD_SALES_INV_TRADEIN.ADVICE_NO

        LEFT OUTER JOIN
        [11_WholegoodsStockUnion] AS TRADEIN_DETAILS
        ON PUBLIC_WGOOD_SALES_INV_TRADEIN.WHOLEGOOD_NO = TRADEIN_DETAILS.WHOLEGOOD_NO

    WHERE
        --Ensure transaction did complete
        WHOLEGOOD_SOLD_DETAILS.[STATUS] IN ('Sold','Sold On Order')

    --ORDER BY
        --WHOLEGOOD_SOLD
)
,
/*=============================================================================================*/
-- Now we follow the tradeins down the chain using anchor recursion
/*=============================================================================================*/
AnchorRecursive
AS
(
--The anchor
    SELECT DISTINCT
        WHOLEGOOD_SOLD,
        TRADE_IN_RECEIVED,
        0 As Distance
    FROM
        TRADEINS

    UNION ALL

--The recursive, runs multiple times until no results
    SELECT
        AnchorRecursive.WHOLEGOOD_SOLD,
        Step2.TRADE_IN_RECEIVED ,
        AnchorRecursive.Distance + 1
    FROM
        AnchorRecursive
        JOIN TRADEINS AS Step2
        ON AnchorRecursive.TRADE_IN_RECEIVED = Step2.WHOLEGOOD_SOLD
    WHERE
        AnchorRecursive.Distance + 1 < 20 --No washout should be longer than 20
)
,
ChainNumbers AS
(
    SELECT DISTINCT
        AnchorRecursive.TRADE_IN_RECEIVED,
        AnchorRecursive.WHOLEGOOD_SOLD,
```

```
        AnchorRecursive.Distance,
        DENSE_RANK() OVER(ORDER BY AnchorRecursive.TRADE_IN_RECEIVED) AS CHAIN_NO_STEP1
    FROM
        AnchorRecursive
)
,
AllChains
AS
(
SELECT
    WHOLEGOOD_SOLD,
    MAX(CHAIN_NO_STEP1) AS CHAIN_NO_STEP2
FROM
    ChainNumbers
GROUP BY
    WHOLEGOOD_SOLD
--ORDER BY
    --CHAIN_NO_STEP2
)
SELECT
    TRADEINS.WHOLEGOOD_SOLD,
    TRADEINS.WHOLEGOOD_SOLD_STATUS,
    TRADEINS.WHOLEGOOD_SOLD_ONADVICE,
    TRADEINS.WHOLEGOOD_SOLD_DATE,
    TRADEINS.WHOLEGOOD_SOLD_CUST,
    CASE WHEN LEFT(TRADEINS.TRADE_IN_RECEIVED,4) = 'END_'
         THEN NULL
         ELSE TRADEINS.TRADE_IN_RECEIVED
         END
         AS TRADE_IN_RECEIVED,
    TRADEINS.TRADE_IN_STATUS,
    TRADEINS.TRADEIN_SOLD_ONADVICE,
    DENSE_RANK() OVER(ORDER BY AllChains.CHAIN_NO_STEP2) AS CHAIN_NO_STEP3
INTO
    [#tmp_AllChains]
FROM
    TRADEINS
    INNER JOIN
    AllChains
    ON TRADEINS.WHOLEGOOD_SOLD = AllChains.WHOLEGOOD_SOLD
ORDER BY
    CHAIN_NO_STEP3,
    WHOLEGOOD_SOLD
;
```

Extract Step 2: Aggregated Profit for Each Chain

The next step is to get the outcome of the model: whether a chain is profitable or not. This is the aggregate of all the profit for each deal (aka advice) within each chain. Each deal's profit is the aggregate of the profit for each machine in that deal: each machine's sale price less its purchase cost.

The system holds at least two purchase costs for each machine: the simple cost agreed with the supplier AND the total cost including various eventualities such as the service required on a used machine after it was

purchased. This model will focus on this adjusted cost.

```
IF OBJECT_ID(N'tempdb..#tmp_AdviceAndChainSummary', N'U') IS NOT NULL
DROP TABLE [#tmp_AdviceAndChainSummary];

WITH
ADVICE_VALUES
AS
(
    SELECT DISTINCT
        CHAIN_NO_STEP3,
        WHOLEGOOD_SOLD_ONADVICE AS ADVICE,
        WHOLEGOOD_SOLD_DATE AS DATE_SOLD,
        WHOLEGOOD_SOLD_CUST AS CUSTOMER,
        SALES_INVOICE AS INVOICE,
        SellingRep AS REP,
        COUNT(DISTINCT WHOLEGOOD_SOLD) AS QTY_WHOLEGOODS_SOLD,
        COUNT(DISTINCT TRADE_IN_RECEIVED) AS QTY_TRADEIN_BOUGHT,
        SUM(SALES_VALUE) AS ADVICE_SALES,
        SUM(SALES_VALUE - COALESCE(COST_PAID,0)) AS ADVICE_PROFIT,
        SUM(SALES_VALUE - COALESCE(Cost_Purch_SBill_LessOwnOAS_PlusTradeInOAS_Jrnl_PDIRefurb,0)) AS ADVI
    FROM
        [#tmp_AllChains]
        INNER JOIN
        [11_WholegoodsStockUnion]
        ON [#tmp_AllChains].WHOLEGOOD_SOLD = [11_WholegoodsStockUnion].WHOLEGOOD_NO
    GROUP BY
        CHAIN_NO_STEP3,
        WHOLEGOOD_SOLD_ONADVICE,
        WHOLEGOOD_SOLD_DATE,
        WHOLEGOOD_SOLD_CUST,
        SALES_INVOICE,
        SellingRep
)
,
CHAIN_VALUE
AS
(
    SELECT
        CHAIN_NO_STEP3,
        COUNT(DISTINCT WHOLEGOOD_SOLD) AS QTY_WHOLEGOODS_SOLD,
        COUNT(DISTINCT TRADE_IN_RECEIVED) AS QTY_TRADEIN_BOUGHT,
        SUM(SALES_VALUE) AS CHAIN_SALES,
        SUM(SALES_VALUE - COALESCE(COST_PAID,0)) AS CHAIN_PROFIT,
        SUM(SALES_VALUE - COALESCE(Cost_Purch_SBill_LessOwnOAS_PlusTradeInOAS_Jrnl_PDIRefurb,0))
          AS CHAIN_PROFIT_ADJ

    FROM
        [#tmp_AllChains]
        INNER JOIN
        [11_WholegoodsStockUnion]
        ON [#tmp_AllChains].WHOLEGOOD_SOLD = [11_WholegoodsStockUnion].WHOLEGOOD_NO
    GROUP BY
        CHAIN_NO_STEP3
```

```
)

SELECT
    ADVICE_VALUES.ADVICE,
    ADVICE_VALUES.INVOICE,
    ADVICE_VALUES.DATE_SOLD,
    ADVICE_VALUES.CUSTOMER,
    ADVICE_VALUES.REP,
    ADVICE_VALUES.CHAIN_NO_STEP3 as CHAIN_NO,
    ADVICE_VALUES.QTY_WHOLEGOODS_SOLD AS ADVICE_QTY_WHOLEGOODS_SOLD,
    ADVICE_VALUES.QTY_TRADEIN_BOUGHT AS ADVICE_QTY_TRADEIN_BOUGHT,
    ADVICE_VALUES.ADVICE_SALES,
    ADVICE_VALUES.ADVICE_PROFIT,
    ADVICE_VALUES.ADVICE_PROFIT_ADJ,
    CHAIN_VALUE.CHAIN_SALES,
    CHAIN_VALUE.CHAIN_PROFIT,
    CASE WHEN CHAIN_VALUE.CHAIN_SALES > 0
         THEN CHAIN_VALUE.CHAIN_PROFIT/CHAIN_VALUE.CHAIN_SALES
         ELSE 0
         END
         AS CHAIN_MARGIN,
    CHAIN_VALUE.CHAIN_PROFIT_ADJ,
    CHAIN_VALUE.QTY_WHOLEGOODS_SOLD AS CHAIN_QTY_WHOLEGOODS_SOLD
INTO
    [#tmp_AdviceAndChainSummary]
FROM
    ADVICE_VALUES
    LEFT OUTER JOIN
    CHAIN_VALUE
    ON ADVICE_VALUES.CHAIN_NO_STEP3 = CHAIN_VALUE.CHAIN_NO_STEP3
;
```

Extract Step 3: Predictors

The model will need some predictors, the following are available, or can be derived from the accouting system data:

-Deal margin after service costs are incluued (the sales price less the adjusted cost) -Who made the deal, the 'REP' -Debtor Profile of the customer (his aged debt with the company) -MachineType, model and age of machine sold, and whether it was new or a trade-in (i.e. used) -MachineType, model and age of machine received as a trade-in -Who the customer was (anonymised reference ID), useful where a customer agrees numerous deals over the years -What profit we might expect from the trade in - using previous experience of that machine type and age -Whether the market, i.e. sales over in the past 12months, is improving with respect to sales in the previous 12mths

When it comes to details of the machines (age, model etc), we need to specify which machine, as a single deal may have many machines being sold and many being traded-in. For the sake of the model, we seek a maximum of one sale machine and one trade-in per deal. The SQL must first identify which is the most valuable machine within the deal, reporting only its details. This is a fair representaiton as most deals are single machines, furthermore, many deals with multiple machines are actually a main machine with attachments; eg tractor with trailer or fork.

```
--=============================================================================
--Details of main wholegood sold
```

```
--=============================================================================

IF OBJECT_ID(N'tempdb..#tmp_HighestValueWholegoodSold', N'U') IS NOT NULL
DROP TABLE [#tmp_HighestValueWholegoodSold];

IF OBJECT_ID(N'tempdb..#tmp_HighestValueTradeIn', N'U') IS NOT NULL
DROP TABLE [#tmp_HighestValueTradeIn];

WITH
HighestValuePerAdvice
AS
(
    SELECT
        SALES_ADVICE,
        DATE_SOLD,
        MAX(SALES_VALUE) AS HIGHEST_SALES_VALUE
    FROM
        [11_WholegoodsStockUnion]
    GROUP BY
        SALES_ADVICE,
        DATE_SOLD
)
,
--occassionally there may be multiple machines on one advice with precisely the same price
--(eg 2xtrailer @ ?30k) we need to select just one, we'll simply take the first (lowest wholegood ID)

HighestValueWholegoodPerAdvice
AS
(
    SELECT
        [11_WholegoodsStockUnion].SALES_ADVICE,
        [11_WholegoodsStockUnion].DATE_SOLD,
        MIN(WHOLEGOOD_NO) AS HIGHEST_SALES_VALUE_WG
    FROM
        [11_WholegoodsStockUnion]
        INNER JOIN
        HighestValuePerAdvice
        ON  [11_WholegoodsStockUnion].SALES_ADVICE = HighestValuePerAdvice.SALES_ADVICE
        AND [11_WholegoodsStockUnion].DATE_SOLD    = HighestValuePerAdvice.DATE_SOLD
        AND [11_WholegoodsStockUnion].SALES_VALUE  = HighestValuePerAdvice.HIGHEST_SALES_VALUE
    GROUP BY
        [11_WholegoodsStockUnion].SALES_ADVICE,
        [11_WholegoodsStockUnion].DATE_SOLD
)
SELECT
    HighestValueWholegoodPerAdvice.SALES_ADVICE,
    HighestValueWholegoodPerAdvice.DATE_SOLD,
    WHOLEGOOD_NO,
    NEW_USED,
    MachineTypeDescription,
    MODEL_CODE,
    MAKE_CODE,
    CASE WHEN [11_WholegoodsStockUnion].YEAR_OF_BUILD > 1985
        THEN YEAR([11_WholegoodsStockUnion].DATE_SOLD) - [11_WholegoodsStockUnion].YEAR_OF_BUILD
```

```
        ELSE NULL
        END
        AS YEARS_OLD
INTO
    [#tmp_HighestValueWholegoodSold]
FROM
    HighestValueWholegoodPerAdvice
    INNER JOIN
    [11_WholegoodsStockUnion]
    ON HighestValueWholegoodPerAdvice.HIGHEST_SALES_VALUE_WG = [11_WholegoodsStockUnion].WHOLEGOOD_NO
;
--=============================================================================
--Details of main trade-in received
--=============================================================================

WITH
TradeInValues
AS
(
    SELECT DISTINCT
        WHOLEGOOD_SOLD_ONADVICE AS SALES_ADVICE,
        WHOLEGOOD_SOLD_DATE AS [DATE],
        TRADE_IN_RECEIVED,
        COST_PAID
    FROM
        [#tmp_AllChains]
        INNER JOIN
        [11_WholegoodsStockUnion]
        ON [#tmp_AllChains].TRADE_IN_RECEIVED = [11_WholegoodsStockUnion].WHOLEGOOD_NO
    --ORDER BY SALES_ADVICE
)
,
HighestValuePerAdvice
AS
(
    SELECT
        SALES_ADVICE,
        [DATE],
        MAX(COST_PAID) AS HighestValueOfTradeIn
    FROM
        TradeInValues
    GROUP BY
        SALES_ADVICE,
        [DATE]
)
,
HighestValueTradeInPerAdvice
AS
(
    SELECT
        TradeInValues.SALES_ADVICE,
        TradeInValues.[DATE],
        MIN(TRADE_IN_RECEIVED) AS HIGHEST_COST_VALUE_TRADEIN
```

```
    FROM
        TradeInValues
        INNER JOIN
        HighestValuePerAdvice
        ON TradeInValues.SALES_ADVICE = HighestValuePerAdvice.SALES_ADVICE
        AND TradeInValues.[DATE] = HighestValuePerAdvice.[DATE]
        AND TradeInValues.COST_PAID = HighestValuePerAdvice.HighestValueOfTradeIn
    GROUP BY
        TradeInValues.SALES_ADVICE,
        TradeInValues.[DATE]
)
SELECT
    HighestValueTradeInPerAdvice.SALES_ADVICE,
    HighestValueTradeInPerAdvice.[DATE],
    WHOLEGOOD_NO,
    NEW_USED,
    MachineTypeDescription,
    MODEL_CODE,
    MAKE_CODE,
    CASE WHEN [11_WholegoodsStockUnion].YEAR_OF_BUILD > 1985
         THEN YEAR(DATE_SOLD) - [11_WholegoodsStockUnion].YEAR_OF_BUILD
         ELSE NULL
         END
         AS YEARS_OLD,
    [11_WholegoodsStockUnion].Cost_Purch_SBill_LessOwnOAS_PlusTradeInOAS_Jrnl_PDIRefurb AS COST_ADJ
INTO
    [#tmp_HighestValueTradeIn]
FROM
    HighestValueTradeInPerAdvice
    INNER JOIN
    [11_WholegoodsStockUnion]
    ON HighestValueTradeInPerAdvice.HIGHEST_COST_VALUE_TRADEIN = [11_WholegoodsStockUnion].WHOLEGOOD_NO
;
```

Extract Step 3b. Predictors: Debtor Information

The historical ability of a customer to pay bills may represent the quality of deal it was possible to do with that customer. The accounting system keeps a record of customer debts on most (not all) business days from 2012 onwards. On any given day this information is kept as 'aged debt', ie debt incurred over 1, 2 and 3 months prior to the day in question. Note, in most businesses 3mth old debt is considered 'bad debt', but it is common in agriculture where cash flows are dictated by the seasons.

The debtor record has gaps which need filling in. Here are the rules used:

Prior to 2012 it may be reasonable to use the customer's average debts. Where we cannot find the records on the precise day of the deal, we must use the nearest records before the deal. If the accounting system finds no debts (NULL records) for the customer and the deal date is after 2012, then customer can be assumed to have ZERO debts

```
IF OBJECT_ID(N'tempdb..#tmp_DebtorInfo', N'U') IS NOT NULL
DROP TABLE [#tmp_DebtorInfo];

--Closest date for which we have debtor information
WITH
```

```
DebtorInfo
AS
(
    SELECT
        ADVICE,
        CUSTOMER,
        DATE_SOLD,
        COALESCE(DateOfFirstDebtorRecord,[PUBLIC_CUSTOMER_MASTER].DATE_CREATED)
          AS DateOfFirstDebtorRecord,
        MAX(DateOfData) AS DateOfClosestDebtorData
    FROM
        [#tmp_AdviceAndChainSummary]

        LEFT OUTER JOIN
        [03_DebtorDetailsCopiedEachUpdate]
        ON [#tmp_AdviceAndChainSummary].DATE_SOLD > [03_DebtorDetailsCopiedEachUpdate].DateOfData
        AND [#tmp_AdviceAndChainSummary].CUSTOMER = [03_DebtorDetailsCopiedEachUpdate].ACCOUNT_NO

        LEFT OUTER JOIN
        (SELECT ACCOUNT_NO, MIN(DateOfData) AS DateOfFirstDebtorRecord
         FROM [03_DebtorDetailsCopiedEachUpdate]
         GROUP BY ACCOUNT_NO)
         AS DateOfFirstDebtorRecords
        ON [#tmp_AdviceAndChainSummary].CUSTOMER = DateOfFirstDebtorRecords.ACCOUNT_NO

        LEFT OUTER JOIN
        [PUBLIC_CUSTOMER_MASTER]
        ON [#tmp_AdviceAndChainSummary].CUSTOMER = [PUBLIC_CUSTOMER_MASTER].ACCOUNT_NO

    GROUP BY
        ADVICE,
        CUSTOMER,
        DATE_SOLD,
        COALESCE(DateOfFirstDebtorRecord,[PUBLIC_CUSTOMER_MASTER].DATE_CREATED)
)
SELECT
    ADVICE,
    CUSTOMER,
    DATE_SOLD,
    DateOfClosestDebtorData,
    DateOfFirstDebtorRecord,
    --When is null debts actually zero debts? This is important...
    --If customer existed less than 30 days ago, but there are no debtor records, then he's new and had
    --OR we are after 2012 (first debtor records) and there being no debts also means there are NO debt
    --ELSE we are before 2012 and we just don't know what debts there are, the model in R will use the
    COALESCE(OutstandingCurrentPeriod,
             CASE WHEN (DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) < 30
                        AND DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) >= 0)
                       OR
                       (DATE_SOLD > CONVERT(DATETIME, '2012-01-30',102))
                  THEN 0
                  ELSE NULL
                  END
```

```
            ) AS OutstandingCurrentPeriod,

    COALESCE(OutstandingMth1,
             CASE WHEN (DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) < 30
                        AND DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) >= 0)
                       OR
                       (DATE_SOLD > CONVERT(DATETIME, '2012-01-30',102))
                  THEN 0
                  ELSE NULL
                  END
            ) AS OutstandingMth1,

    COALESCE(OutstandingMth2,
             CASE WHEN (DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) < 30
                  AND DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) >= 0)
                       OR
                       (DATE_SOLD > CONVERT(DATETIME, '2012-01-30',102))
                  THEN 0
                  ELSE NULL
                  END
            ) AS OutstandingMth2,

    COALESCE([OutstandingMth3+],
             CASE WHEN (DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) < 30
                        AND DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) >= 0)
                       OR
                       (DATE_SOLD > CONVERT(DATETIME, '2012-01-30',102))
                  THEN 0
                  ELSE NULL
                  END
            ) AS [OutstandingMth3+],

    COALESCE([TotalOutstandingIncVAT],
             CASE WHEN (DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) < 30
                       AND DATEDIFF(dd, DateOfFirstDebtorRecord, DATE_SOLD) >= 0)
                       OR
                       (DATE_SOLD > CONVERT(DATETIME, '2012-01-30',102))
                  THEN 0
                  ELSE NULL
                  END
            ) AS [TotalOutstandingIncVAT]

INTO
    [#tmp_DebtorInfo]
FROM
    DebtorInfo
    LEFT OUTER JOIN
    [03_DebtorDetailsCopiedEachUpdate]
    ON [03_DebtorDetailsCopiedEachUpdate].ACCOUNT_NO = DebtorInfo.CUSTOMER
    AND [03_DebtorDetailsCopiedEachUpdate].DateOfData = DebtorInfo.DateOfClosestDebtorData
;
```

Extract Step 3c: 'Chain Profit' and Date, then Focus on Incomplete Chains

If a human were asked to make a judgement about wether a chain will be profitable or not, the first fact they would seek would be the profitability of the chain so far. Clearly the model needs the same facts.

The model should also focus on incomplete deals. We don't need a model to tell us whether a chain is profitable after it has already been completed (aka washed out).

```
IF OBJECT_ID(N'tempdb..#tmp_ChainCumulativeToDate', N'U') IS NOT NULL
DROP TABLE [#tmp_ChainCumulativeToDate];

IF OBJECT_ID(N'tempdb..#tmp_ChainsWithMoreThanOneAdvice', N'U') IS NOT NULL
DROP TABLE [#tmp_ChainsWithMoreThanOneAdvice];

WITH
AdvicesAndTheirPriorsWithinChain
As
(
    SELECT DISTINCT
        Chains.CHAIN_NO_STEP3 AS CHAIN_NO,
        Chains.WHOLEGOOD_SOLD_ONADVICE AS ADVICE,
        Chains.WHOLEGOOD_SOLD_DATE AS DATE_SOLD
    FROM
        (SELECT DISTINCT CHAIN_NO_STEP3, WHOLEGOOD_SOLD_ONADVICE, WHOLEGOOD_SOLD_DATE
         FROM [#tmp_AllChains])
        As Chains
        LEFT OUTER JOIN
        (SELECT DISTINCT CHAIN_NO_STEP3, WHOLEGOOD_SOLD_ONADVICE, WHOLEGOOD_SOLD_DATE
         FROM [#tmp_AllChains])
        As ChainsPrior
        ON Chains.CHAIN_NO_STEP3 = ChainsPrior.CHAIN_NO_STEP3
        AND Chains.WHOLEGOOD_SOLD_DATE <= ChainsPrior.WHOLEGOOD_SOLD_DATE

)
SELECT
    AdvicesAndTheirPriorsWithinChain.CHAIN_NO,
    AdvicesAndTheirPriorsWithinChain.ADVICE,
    AdvicesAndTheirPriorsWithinChain.DATE_SOLD,
    ADVICE_PROFIT,
    ADVICE_PROFIT_ADJ,
    ROW_NUMBER() OVER(PARTITION BY AdvicesAndTheirPriorsWithinChain.CHAIN_NO ORDER BY AdvicesAndTheirPr
    SUM(ADVICE_PROFIT) OVER(PARTITION BY AdvicesAndTheirPriorsWithinChain.CHAIN_NO ORDER BY AdvicesAndTh
    SUM(ADVICE_PROFIT_ADJ) OVER(PARTITION BY AdvicesAndTheirPriorsWithinChain.CHAIN_NO ORDER BY Advices
INTO
    [#tmp_ChainCumulativeToDate]
FROM
    AdvicesAndTheirPriorsWithinChain
    LEFT OUTER JOIN
    [#tmp_AdviceAndChainSummary]
    ON AdvicesAndTheirPriorsWithinChain.CHAIN_NO = [#tmp_AdviceAndChainSummary].CHAIN_NO
    AND AdvicesAndTheirPriorsWithinChain.ADVICE = [#tmp_AdviceAndChainSummary].ADVICE
ORDER BY
    AdvicesAndTheirPriorsWithinChain.CHAIN_NO,
    SequenceInChain
```

```
;

--Chains with more than one advice - not interested in straight deals which have only one deal in the c
SELECT
    CHAIN_NO,
    MAX(SequenceInChain) AS MaxSequence
INTO
    [#tmp_ChainsWithMoreThanOneAdvice]
FROM
    [#tmp_ChainCumulativeToDate]
GROUP BY
    CHAIN_NO
HAVING
    MAX(SequenceInChain) > 1
ORDER BY
    CHAIN_NO
;
```

Extract Step 3d. Predictors: Market Movement

An experienced human making a judgement about the profitability of a chain may well consider whether the market for the trade-ins being received in increasing or reducing. This can be approximated by a ratio of the sales for the machine type in previous 12 months (i.e. months 0-12) vs sales in the prior 12 months (i.e. months 12-24).

```
IF OBJECT_ID(N'tempdb..#tmp_MarketMovement', N'U') IS NOT NULL
DROP TABLE [#tmp_MarketMovement];

/*
================================================================================
Is market improving or declining? Sales over Mths 0-12 vs Sales over Mths 12-24
================================================================================
*/

SELECT
    Sales0To12.MonthEnd,
    Sales0To12.CalendarYear,
    Sales0To12.CalendarMonth,
    Sales0To12.[MachineTypeDescription],
    SUM(Sales0To12.SALES_VALUE) AS Rolling12MthSales_0To12,
    SUM(Sales12To24.SALES_VALUE) AS Rolling12MthSales_12To24,
    SUM(Sales0To12.SALES_VALUE) / SUM(Sales12To24.SALES_VALUE) As RatioSalesThisYrVsPrevYr
INTO
    [#tmp_MarketMovement]
FROM
    [24_PivotData_WholegoodsStock] AS Sales0To12
    INNER JOIN
    [24_PivotData_WholegoodsStock] AS Sales12To24
    ON Sales0To12.CalendarMonth = Sales12To24.CalendarMonth
    AND Sales0To12.CalendarYear = Sales12To24.CalendarYear+1
    AND Sales0To12.MachineTypeDescription = Sales12To24.MachineTypeDescription
WHERE
    Sales0To12.RecordType = 'Rolling12MthSales'
```

```
    AND
    Sales12To24.RecordType = 'Rolling12MthSales'
    AND
    Sales0To12.MonthEnd > CONVERT(DATETIME, '2010-06-01',102)
    AND
    Sales0To12.NEW_USED = 'U'
    AND
    Sales12To24.NEW_USED = 'U'
GROUP BY
    Sales0To12.MonthEnd,
    Sales0To12.CalendarYear,
    Sales0To12.CalendarMonth,
    Sales0To12.[MachineTypeDescription]
ORDER BY
    MonthEnd,
    MachineTypeDescription
;
```

Extract Step 3e. Predictors: Expected Future Profit for a Machine

Each year the company is required to present a 'net realisable value' to its auditors for each machine in stock. Having this figure would allow us stimate the expected future profit of selling a machine, which is directly relevant to our model.

Unfortunately, the auditors receive estimates only for mahcine in stock at year end. We need to build our own model to estimate the sales value of any given machine. This can be done using a simple logisitic regression which derives a trend line for the ratio of the sales price to the purchase cost (aka Mark Up) as a function of the period of time the mahcine is in stock. Usually machines start that line with a ratio of 1.1 (a 10% mark up if sold immediately after purchase) which erodes as the machine lingers in stock. The rate of erosion of the sales price is typically 9% per 12mths in stock, so approximately break even after 12mths. Of course, it is possible to derive similar trends for each machine type as there are 10,000 machine sales in the 10yrs of history available.

Note, expected future profit of selling a machine is different to the expected profitability of an entire chain of deals. There may be multiple future deals and sales yet to pass in the chain.

This question of the 'expected future profit' could warrant investigation of a model in its own right, but for the sake of simplicity we will satisfy ourselves with a simple (one predictor) linear regession calculated in SQL for convenience.

```
IF OBJECT_ID(N'tempdb..#tmp_MarkUpVsYrs_SlopeAndIntercept', N'U') IS NOT NULL
DROP TABLE [#tmp_MarkUpVsYrs_SlopeAndIntercept];

/*================================================================================
What sale price can we except? MarkUp Linear Regression, Slope and Intercept
================================================================================*/

WITH
CoreData
AS
(
    SELECT
        MachineTypeDescription,
        WHOLEGOOD_NO,
```

```
        SALES_VALUE,
        COST_PAID,
        SALES_VALUE-COST_PAID AS GrossProfit,
        (SALES_VALUE-COST_PAID)/COST_PAID AS MarkUp,
        CAST(DaysInStockToDate AS DECIMAL(10,5))/365 AS YearsOnBooks
    FROM
        [11_WholegoodsStockUnion]
    WHERE
        SALES_VALUE > 0
        AND
        COST_PAID > 0
        AND
        NEW_USED = 'U'
)
,
IdentifyOutliersByMarkUp
AS
(
    SELECT
        MachineTypeDescription,
        --3 SD from the Mean is far enough!
        AVG(MarkUp)+3*STDEV(MarkUp) AS Threshold
    FROM
        CoreData
    GROUP BY
        MachineTypeDescription
)
,
CoreData_ExcOutliers
AS
(
    SELECT
        CoreData.MachineTypeDescription,
        WHOLEGOOD_NO,
        SALES_VALUE,
        COST_PAID,
        GrossProfit,
        MarkUp,
        YearsOnBooks
    FROM
        CoreData
        INNER JOIN
        IdentifyOutliersByMarkUp
        ON CoreData.MachineTypeDescription = IdentifyOutliersByMarkUp.MachineTypeDescription
    WHERE
        --Filter out the outliers
        ABS(CoreData.MarkUp) < IdentifyOutliersByMarkUp.Threshold
)
,
xbar_ybar
AS
(
    SELECT
```

```
        CoreData_ExcOutliers.MachineTypeDescription,
        COUNT(WHOLEGOOD_NO) AS QtyRecordsInAnalysis,
        AVG(MarkUp) AS MarkUp_Avg, --This is is ybar
        AVG(YearsOnBooks) AS YearsOnBooks_Avg --This is is xbar
    FROM
        CoreData_ExcOutliers
    GROUP BY
        CoreData_ExcOutliers.MachineTypeDescription
)
,
slopes
AS
(
    SELECT
        CoreData_ExcOutliers.MachineTypeDescription,
        SUM((YearsOnBooks - YearsOnBooks_Avg) * (MarkUp - MarkUp_Avg))
            /
            SUM((YearsOnBooks - YearsOnBooks_Avg) * (YearsOnBooks - YearsOnBooks_Avg)) AS slope,

        MAX(MarkUp_Avg) AS ybar,
        MAX(YearsOnBooks_Avg) AS xbar,
        MAX(QtyRecordsInAnalysis) AS QtyRecordsInAnalysis
    FROM
        CoreData_ExcOutliers
        INNER JOIN
        xbar_ybar
        ON CoreData_ExcOutliers.MachineTypeDescription = xbar_ybar.MachineTypeDescription
    GROUP BY
        CoreData_ExcOutliers.MachineTypeDescription
    HAVING
        SUM((YearsOnBooks - YearsOnBooks_Avg) * (YearsOnBooks - YearsOnBooks_Avg))<> 0
)
SELECT
    MachineTypeDescription,
    QtyRecordsInAnalysis,
    slope,
    ybar,
    xbar,
    ybar - (slope * xbar) as intercept
INTO
    [#tmp_MarkUpVsYrs_SlopeAndIntercept]
FROM
    slopes
ORDER BY
    MachineTypeDescription
;
```

Step 3f. Predictors: Average wait time before sale

The above valuation method requires an estimate for how long we should expect each machine to remain in stock, ie how much should its mark up erode from the perfect case of 10%? There are a number of different ways of doing this, but we can use a simple average wait time for our model.

```
/*================================================================================
How long should we expect to wait before selling the trade-in? Avg wait times
================================================================================*/
IF OBJECT_ID(N'tempdb..#tmp_WaitTime', N'U') IS NOT NULL
DROP TABLE [#tmp_WaitTime];

SELECT
    MachineTypeDescription,
    COUNT([11_WholegoodsStockUnion].WHOLEGOOD_NO) AS QtyWholegoods,
    (SUM(CAST(DaysInStockToDate AS DECIMAL(10,5)) * COST_PAID) / SUM(COST_PAID))/365
      AS YearsToWait_ValueWeighted
INTO
    [#tmp_WaitTime]
FROM
    [11_WholegoodsStockUnion]
WHERE
    (SALES_VALUE > 0 OR SALES_VALUE IS NULL)
    AND
    COST_PAID > 0
    AND
    NEW_USED = 'U'
    AND
    --constrain years, so we are sure of purchase date and sales date.
    DATE_ACQUIRED > CONVERT(DATETIME, '2011-01-01',102) --any earlier and machine may have been in stoc
    AND
    DATE_ACQUIRED < CONVERT(DATETIME, '2014-12-31',102) --any later and machine may not have had time t
    AND
    STOCK_STATUS NOT IN('F','H') --not fixed asset nor hire machine as these intentionally wait a long
GROUP BY
    MachineTypeDescription
ORDER BY
    MachineTypeDescription
;
```

Collation of SQL extract

All the above data is stored in temp tables in SQL server, it now needs extracting such that each row is a deal (aka advice), with multiple columns representing: a) outcome (whether it belongs profitable chain b) the above predictors

The below SQL creates a table in SQL Server to which we can connect using the RODBC package.

```
--================================================================================================
--Bring it all together and replace the tab le of data which is the output of this entire procedure
--================================================================================================
IF EXISTS (SELECT *
           FROM sys.objects
           WHERE object_id = object_id(N'[41_ChainAnalysisByAdvice]')
           AND   TYPE = N'U')
    BEGIN
        DROP TABLE [41_ChainAnalysisByAdvice]
```

```
    END;

SELECT
    [#tmp_AdviceAndChainSummary].ADVICE,
    [#tmp_AdviceAndChainSummary].INVOICE,
    [#tmp_AdviceAndChainSummary].DATE_SOLD,
    MONTH([#tmp_AdviceAndChainSummary].DATE_SOLD) AS DATE_SOLD_Month,
    YEAR([#tmp_AdviceAndChainSummary].DATE_SOLD) AS DATE_SOLD_Year,
    [#tmp_AdviceAndChainSummary].CUSTOMER,
    [#tmp_AdviceAndChainSummary].REP,
    [#tmp_AdviceAndChainSummary].CHAIN_NO,
    [#tmp_AdviceAndChainSummary].ADVICE_QTY_WHOLEGOODS_SOLD,
    [#tmp_AdviceAndChainSummary].ADVICE_QTY_TRADEIN_BOUGHT,
    [#tmp_AdviceAndChainSummary].ADVICE_SALES,
    [#tmp_AdviceAndChainSummary].ADVICE_PROFIT,
    [#tmp_AdviceAndChainSummary].ADVICE_PROFIT_ADJ,
    [#tmp_AdviceAndChainSummary].CHAIN_SALES,
    [#tmp_AdviceAndChainSummary].CHAIN_PROFIT,
    [#tmp_AdviceAndChainSummary].CHAIN_MARGIN,
    [#tmp_AdviceAndChainSummary].CHAIN_PROFIT_ADJ,
    CASE WHEN [#tmp_AdviceAndChainSummary].CHAIN_PROFIT_ADJ >= 100 THEN 1 ELSE 0 END AS Worthwhile,
    [#tmp_AdviceAndChainSummary].CHAIN_QTY_WHOLEGOODS_SOLD,

    [#tmp_ChainCumulativeToDate].SequenceInChain AS ChainSeq_SeqNo,
    [#tmp_ChainCumulativeToDate].ADVICE_PROFIT_CUMULATIVE AS ChainSeq_CumulativeProfit,
    [#tmp_ChainCumulativeToDate].ADVICE_PROFIT_ADJ_CUMULATIVE AS ChainSeq_CumulativeProfitAdj,

    --[#tmp_DebtorInfo].DateOfClosestDebtorData,
    --[#tmp_DebtorInfo].DateOfFirstDebtorRecord,
    [#tmp_DebtorInfo].OutstandingCurrentPeriod AS Debtor_OutstandingMth0,
    [#tmp_DebtorInfo].OutstandingMth1 AS Debtor_OutstandingMth1,
    [#tmp_DebtorInfo].OutstandingMth2 AS Debtor_OutstandingMth2,
    [#tmp_DebtorInfo].[OutstandingMth3+] AS Debtor_OutstandingMth3Plus,
    [#tmp_DebtorInfo].[TotalOutstandingIncVAT] AS Debtor_OutstandingTotal,

    [#tmp_HighestValueWholegoodSold].NEW_USED AS WholegoodSold_NEW_USED,
    [#tmp_HighestValueWholegoodSold].MachineTypeDescription AS WholegoodSold_MachineTypeDesc,
    [#tmp_HighestValueWholegoodSold].MODEL_CODE AS WholegoodSold_MODEL_CODE,
    [#tmp_HighestValueWholegoodSold].MAKE_CODE AS WholegoodSold_MAKE_CODE,
    [#tmp_HighestValueWholegoodSold].YEARS_OLD AS WholegoodSold_YEARS_OLD,

    [#tmp_HighestValueTradeIn].NEW_USED AS TradeIn_NEW_USED,
    [#tmp_HighestValueTradeIn].MachineTypeDescription AS TradeIn_MachineTypeDesc,
    [#tmp_HighestValueTradeIn].MODEL_CODE AS TradeIn_MODEL_CODE,
    [#tmp_HighestValueTradeIn].MAKE_CODE AS TradeIn_MAKE_CODE,
    [#tmp_HighestValueTradeIn].YEARS_OLD AS TradeIn_YEARS_OLD,

    [#tmp_HighestValueTradeIn].COST_ADJ AS TradeIn_Cost_Adj,
    [#tmp_MarkUpVsYrs_SlopeAndIntercept].slope,
    [#tmp_MarkUpVsYrs_SlopeAndIntercept].intercept,
    [#tmp_WaitTime].YearsToWait_ValueWeighted AS YearsToWait,

    ((([#tmp_WaitTime].YearsToWait_ValueWeighted * [#tmp_MarkUpVsYrs_SlopeAndIntercept].slope)
```

```
        + [#tmp_MarkUpVsYrs_SlopeAndIntercept].intercept)
        * [#tmp_HighestValueTradeIn].COST_ADJ)
        AS ExpectedProfitFromTradeIn,

    [#tmp_AdviceAndChainSummary].CHAIN_PROFIT +
    ((([#tmp_WaitTime].YearsToWait_ValueWeighted * [#tmp_MarkUpVsYrs_SlopeAndIntercept].slope)
        + [#tmp_MarkUpVsYrs_SlopeAndIntercept].intercept)
        * [#tmp_HighestValueTradeIn].COST_ADJ)
        AS ExpectedProfitFromChain,

    COALESCE([#tmp_MarketMovement].RatioSalesThisYrVsPrevYr,
             [MarketMovement_Avg].RatioSalesThisYrVsPrevYr_Avg)
             AS RatioSalesThisYrVsPrevYr

INTO
    [41_ChainAnalysisByAdvice]

FROM
    [#tmp_AdviceAndChainSummary]

    --constrain to only chains with more than one deal.
    --cahins with just one deal (straight deals) are easy to assess as they are self contained, no need
    INNER JOIN
    [#tmp_ChainsWithMoreThanOneAdvice]
    ON [#tmp_AdviceAndChainSummary].CHAIN_NO = [#tmp_ChainsWithMoreThanOneAdvice].CHAIN_NO

    LEFT OUTER JOIN
    [#tmp_ChainCumulativeToDate]
    ON [#tmp_AdviceAndChainSummary].ADVICE = [#tmp_ChainCumulativeToDate].ADVICE

    LEFT OUTER JOIN
    [#tmp_DebtorInfo]
    ON [#tmp_AdviceAndChainSummary].ADVICE = [#tmp_DebtorInfo].ADVICE

    LEFT OUTER JOIN
    [#tmp_HighestValueWholegoodSold]
    ON [#tmp_AdviceAndChainSummary].ADVICE = [#tmp_HighestValueWholegoodSold].SALES_ADVICE

    LEFT OUTER JOIN
    [#tmp_HighestValueTradeIn]
    ON [#tmp_AdviceAndChainSummary].ADVICE = [#tmp_HighestValueTradeIn].SALES_ADVICE

    LEFT OUTER JOIN
    [#tmp_MarketMovement]
    ON YEAR([#tmp_AdviceAndChainSummary].DATE_SOLD) =
        [#tmp_MarketMovement].CalendarYear
    AND MONTH([#tmp_AdviceAndChainSummary].DATE_SOLD) =
        [#tmp_MarketMovement].CalendarMonth
    AND [#tmp_HighestValueTradeIn].MachineTypeDescription =
        [#tmp_MarketMovement].MachineTypeDescription

    LEFT OUTER JOIN
    --Need a way of using an average value where there would be nulls
```

```
    (SELECT
        MachineTypeDescription,
        AVG(RatioSalesThisYrVsPrevYr) AS RatioSalesThisYrVsPrevYr_Avg
     FROM
        [#tmp_MarketMovement]
     GROUP BY
        MachineTypeDescription) AS [MarketMovement_Avg]
    ON [#tmp_HighestValueTradeIn].MachineTypeDescription =
       [MarketMovement_Avg].MachineTypeDescription

    LEFT OUTER JOIN
    [#tmp_WaitTime]
    ON [#tmp_HighestValueTradeIn].MachineTypeDescription =
       [#tmp_WaitTime].MachineTypeDescription

    LEFT OUTER JOIN
    [#tmp_MarkUpVsYrs_SlopeAndIntercept]
    ON [#tmp_HighestValueTradeIn].MachineTypeDescription =
       [#tmp_MarkUpVsYrs_SlopeAndIntercept].MachineTypeDescription

WHERE
    --constrain to deals which has trade-ins, so chain is incomplete.
    --These are the deals we need to forecast.
    --Profit of straight deals and deals at end of chain are easy to calculate.
    [#tmp_HighestValueTradeIn].WHOLEGOOD_NO IS NOT NULL

ORDER BY
    CHAIN_NO,
    ChainSeq_SeqNo
;

--============================================================
--CLEAN UP the temp tables
--============================================================

IF OBJECT_ID(N'tempdb..#tmp_AllChains', N'U') IS NOT NULL
DROP TABLE [#tmp_AllChains];

IF OBJECT_ID(N'tempdb..#tmp_AdviceAndChainSummary', N'U') IS NOT NULL
DROP TABLE [#tmp_AdviceAndChainSummary];

IF OBJECT_ID(N'tempdb..#tmp_HighestValueWholegoodSold', N'U') IS NOT NULL
DROP TABLE [#tmp_HighestValueWholegoodSold];

IF OBJECT_ID(N'tempdb..#tmp_HighestValueTradeIn', N'U') IS NOT NULL
DROP TABLE [#tmp_HighestValueTradeIn];

IF OBJECT_ID(N'tempdb..#tmp_ChainCumulativeToDate', N'U') IS NOT NULL
DROP TABLE [#tmp_ChainCumulativeToDate];

IF OBJECT_ID(N'tempdb..#tmp_ChainsWithMoreThanOneAdvice', N'U') IS NOT NULL
DROP TABLE [#tmp_ChainsWithMoreThanOneAdvice];
```

```
IF OBJECT_ID(N'tempdb..#tmp_DebtorInfo', N'U') IS NOT NULL
DROP TABLE [#tmp_DebtorInfo];

IF OBJECT_ID(N'tempdb..#tmp_MarketMovement', N'U') IS NOT NULL
DROP TABLE [#tmp_MarketMovement];

IF OBJECT_ID(N'tempdb..#tmp_WaitTime', N'U') IS NOT NULL
DROP TABLE [#tmp_WaitTime];

IF OBJECT_ID(N'tempdb..#tmp_MarkUpVsYrs_SlopeAndIntercept', N'U') IS NOT NULL
DROP TABLE [#tmp_MarkUpVsYrs_SlopeAndIntercept];

END
;
```