HW8 - Final Project

Olivia Samples

7/16/2020

Part 1: Question of Interest and Cohort (10 points)

Select a question that you are interested under the condition that you can find a corresponding dataset to answer your question. Check to see if your dataset has enough information to support you answer the question. Define your cohort of interest based on the data, illustrate if you have any criteria to define your cohort.

Part 2: Methods

Dataset (5 points) Data will not be provided for the final project, instead, you are asked to either create one dataset or download one from some website, which contains the information you are intrested to answer. Check the data before you start modeling. Make sure that

The dataset has at least 10 variables including both numeric and categorical data The dataset has at least 500 observations There are some missing information for some predictors. The response variable (or dependent variable) is binary Data cleaning (5 points) After the data is loaded, check if you need to clean the data

Make sure each variable is in a format it supposed to be, like categorical or numeric If the variable is character, transform it to integer then categorical (if needed) If a variable has missing value, you may consider imputation Modeling Split your data into two sets (70/30), where 70% for training, and 30% as testing. Using training function in Caret to train your model with (20 points) KNN Decision tree Random forest Support vector machine find the best model, then test your modeling using the test data, report the accuracy and auc.

Use LASSO for model selection. (5 points)

Using the selected variables from LASSO then run (15 points) Logistic regression Random Forest Nueral Network Test your model with testing data, report accuracy and AUC.

Part 3: Solution

Dataset Description:

This dataset contains an airline passenger satisfaction survey.

Prediction task is to determine what factors are highly correlated to a satisfied (or dissatisfied) passenger and to predict passenger satisfaction.

https://www.kaggle.com/teejmahal20/airline-passenger-satisfaction?select=train.csv

Variable Descriptions:

Gender: Gender of the passengers (Female, Male)

Customer Type: The customer type (Loyal customer, disloyal customer)

Age: The actual age of the passengers

Type of Travel: Purpose of the flight of the passengers (Personal Travel, Business Travel)

Class: Travel class in the plane of the passengers (Business, Eco, Eco Plus)

Flight distance: The flight distance of this journey

Inflight wifi service: Satisfaction level of the inflight wifi service (0:Not Applicable;1-5)

Departure/Arrival time convenient: Satisfaction level of Departure/Arrival time convenient

Ease of Online booking: Satisfaction level of online booking

Gate location: Satisfaction level of Gate location

Food and drink: Satisfaction level of Food and drink

Online boarding: Satisfaction level of online boarding

Seat comfort: Satisfaction level of Seat comfort

Inflight entertainment: Satisfaction level of inflight entertainment

On-board service: Satisfaction level of On-board service

Leg room service: Satisfaction level of Leg room service

Baggage handling: Satisfaction level of baggage handling

Check-in service: Satisfaction level of Check-in service

Inflight service: Satisfaction level of inflight service

Cleanliness: Satisfaction level of Cleanliness

Departure Delay in Minutes: Minutes delayed when departure

Arrival Delay in Minutes: Minutes delayed when Arrival

Satisfaction: Airline satisfaction level(Satisfaction, neutral or dissatisfaction)

Cohort of Interest

I really enjoy flying and going to different places. I am curious which factors may affect a passengers flight satisfaction. Specifically, do passenger demographics and flight characteristics affect how well a person perceives their flight experience? Demographics may include Age and Gender, while an example of a flight characteristic is Flight Distance. We will look into this question using many modeling tools and techniques, along with data exploration. We will begin by cleaning our data.

Read the csv file.

```
train <- read.csv(file='train.csv', head=TRUE, sep=",")
dim(train)
```

```
## [1] 103904     25
```

```
summary(train)
```

```
##        X                id            Gender                Customer.Type
##  Min.   :     0   Min.   :     1   Female:52727   disloyal Customer:18981
##  1st Qu.: 25976   1st Qu.: 32534   Male  :51177   Loyal Customer   :84923
##  Median : 51952   Median : 64856
##  Mean   : 51952   Mean   : 64924
##  3rd Qu.: 77927   3rd Qu.: 97368
##  Max.   :103903   Max.   :129880
##
```

```
##       Age               Type.of.Travel       Class       Flight.Distance
##  Min.   : 7.00   Business travel:71655   Business:49665   Min.   :  31
##  1st Qu.:27.00   Personal Travel:32249   Eco     :46745   1st Qu.: 414
##  Median :40.00                           Eco Plus: 7494   Median : 843
##  Mean   :39.38                                            Mean   :1189
##  3rd Qu.:51.00                                            3rd Qu.:1743
##  Max.   :85.00                                            Max.   :4983
##
##  Inflight.wifi.service Departure.Arrival.time.convenient Ease.of.Online.booking
##  Min.   :0.00          Min.   :0.00                      Min.   :0.000
##  1st Qu.:2.00          1st Qu.:2.00                      1st Qu.:2.000
##  Median :3.00          Median :3.00                      Median :3.000
##  Mean   :2.73          Mean   :3.06                      Mean   :2.757
##  3rd Qu.:4.00          3rd Qu.:4.00                      3rd Qu.:4.000
##  Max.   :5.00          Max.   :5.00                      Max.   :5.000
##
##  Gate.location   Food.and.drink  Online.boarding  Seat.comfort
##  Min.   :0.000   Min.   :0.000   Min.   :0.00    Min.   :0.000
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.00    1st Qu.:2.000
##  Median :3.000   Median :3.000   Median :3.00    Median :4.000
##  Mean   :2.977   Mean   :3.202   Mean   :3.25    Mean   :3.439
##  3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.00    3rd Qu.:5.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.00    Max.   :5.000
##
##  Inflight.entertainment On.board.service Leg.room.service Baggage.handling
##  Min.   :0.000          Min.   :0.000    Min.   :0.000    Min.   :1.000
##  1st Qu.:2.000          1st Qu.:2.000    1st Qu.:2.000    1st Qu.:3.000
##  Median :4.000          Median :4.000    Median :4.000    Median :4.000
##  Mean   :3.358          Mean   :3.382    Mean   :3.351    Mean   :3.632
##  3rd Qu.:4.000          3rd Qu.:4.000    3rd Qu.:4.000    3rd Qu.:5.000
##  Max.   :5.000          Max.   :5.000    Max.   :5.000    Max.   :5.000
##
##  Checkin.service Inflight.service  Cleanliness    Departure.Delay.in.Minutes
##  Min.   :0.000   Min.   :0.00     Min.   :0.000   Min.   :   0.00
##  1st Qu.:3.000   1st Qu.:3.00     1st Qu.:2.000   1st Qu.:   0.00
##  Median :3.000   Median :4.00     Median :3.000   Median :   0.00
##  Mean   :3.304   Mean   :3.64     Mean   :3.286   Mean   :  14.82
##  3rd Qu.:4.000   3rd Qu.:5.00     3rd Qu.:4.000   3rd Qu.:  12.00
##  Max.   :5.000   Max.   :5.00     Max.   :5.000   Max.   :1592.00
##
##  Arrival.Delay.in.Minutes                  satisfaction
##  Min.   :   0.00          neutral or dissatisfied:58879
##  1st Qu.:   0.00          satisfied              :45025
##  Median :   0.00
##  Mean   :  15.18
##  3rd Qu.:  13.00
##  Max.   :1584.00
##  NA's   :310
```

Here, we have five categorical variables (Gender, Customer.Type, Type.of.Travel, Class, and satisfaction). The rest are numerical.

```
cleandata = function(data){
        cleandata = data[, c(5, 8:24)]
        cleandata = lapply(cleandata, as.numeric)
```

```
        #categorical to dummy variable
        numGender = model.matrix(~Gender, data = data)
        numCustomer.Type = model.matrix(~Customer.Type, data=data)
        numType.Of.Travel = model.matrix(~Type.of.Travel, data = data)
        numClass = model.matrix(~Class, data = data)

        cleandata = data.frame(cleandata, numGender, numCustomer.Type, numType.Of.Travel, numClass)
        cleandata = cleandata[, c(1:18, 20, 22, 24, 26:27)]
        return(cleandata)
}

Target = as.factor(train$satisfaction)
trainclean = data.frame(cleandata(train), Target)
```

Let us check how many null values we have.

```
colna = function(data){
        null = is.na.data.frame(data)
        colnames(null)
        for (i in 1:length(data)){
                print(sum(null[,i]))
        }
        percentage = sum(null[,2])/NROW(null[,2])
        return(percentage)
}

colna(trainclean)
```

```
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 310
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
```

```
## [1] 0
```

We have 310 missing values for our "Arrival.Delay.in.Minutes" variable, so we will impute to resolve this.

```
library(mice)
```

```
##
## Attaching package: 'mice'
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
imputation = function(data, m=3, method = "pmm", maxit = 10, seed = 500){
        imputed = mice(data = data, m = m, method = method, maxit = maxit, seed = seed)
        mydata = as.data.frame(complete(imputed, 1))
        return(mydata)
}
imptrain = imputation(trainclean)
```

```
coltype = function(data){
     sapply(data, class)
}
coltype(imptrain)
```

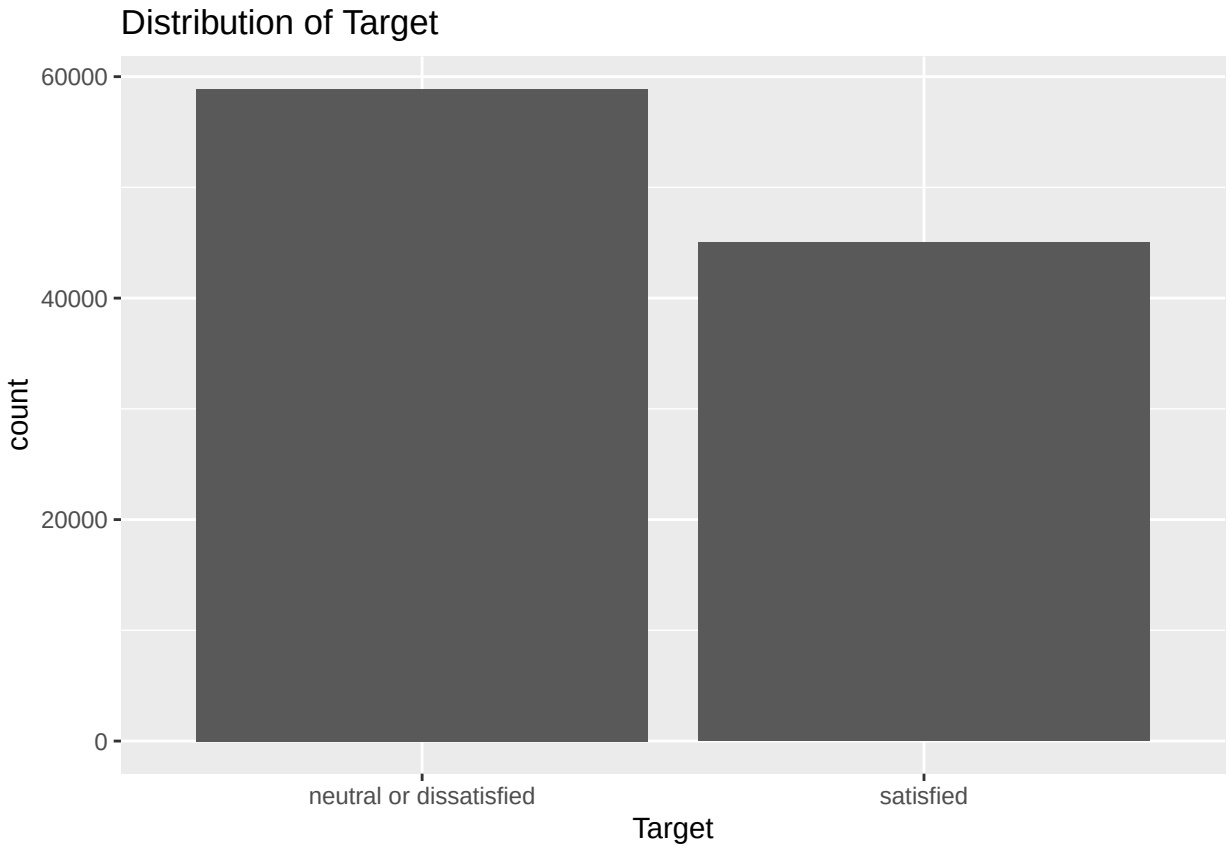
```
##                                Age                    Flight.Distance
##                          "numeric"                          "numeric"
##             Inflight.wifi.service Departure.Arrival.time.convenient
##                          "numeric"                          "numeric"
##            Ease.of.Online.booking                      Gate.location
##                          "numeric"                          "numeric"
##                      Food.and.drink                    Online.boarding
##                          "numeric"                          "numeric"
##                        Seat.comfort            Inflight.entertainment
##                          "numeric"                          "numeric"
##                   On.board.service                   Leg.room.service
##                          "numeric"                          "numeric"
##                   Baggage.handling                    Checkin.service
##                          "numeric"                          "numeric"
##                   Inflight.service                        Cleanliness
##                          "numeric"                          "numeric"
##        Departure.Delay.in.Minutes          Arrival.Delay.in.Minutes
##                          "numeric"                          "numeric"
##                         GenderMale       Customer.TypeLoyal.Customer
##                          "numeric"                          "numeric"
##     Type.of.TravelPersonal.Travel                           ClassEco
##                          "numeric"                          "numeric"
##                       ClassEco.Plus                             Target
##                          "numeric"                           "factor"
```

```
colna(imptrain)
```

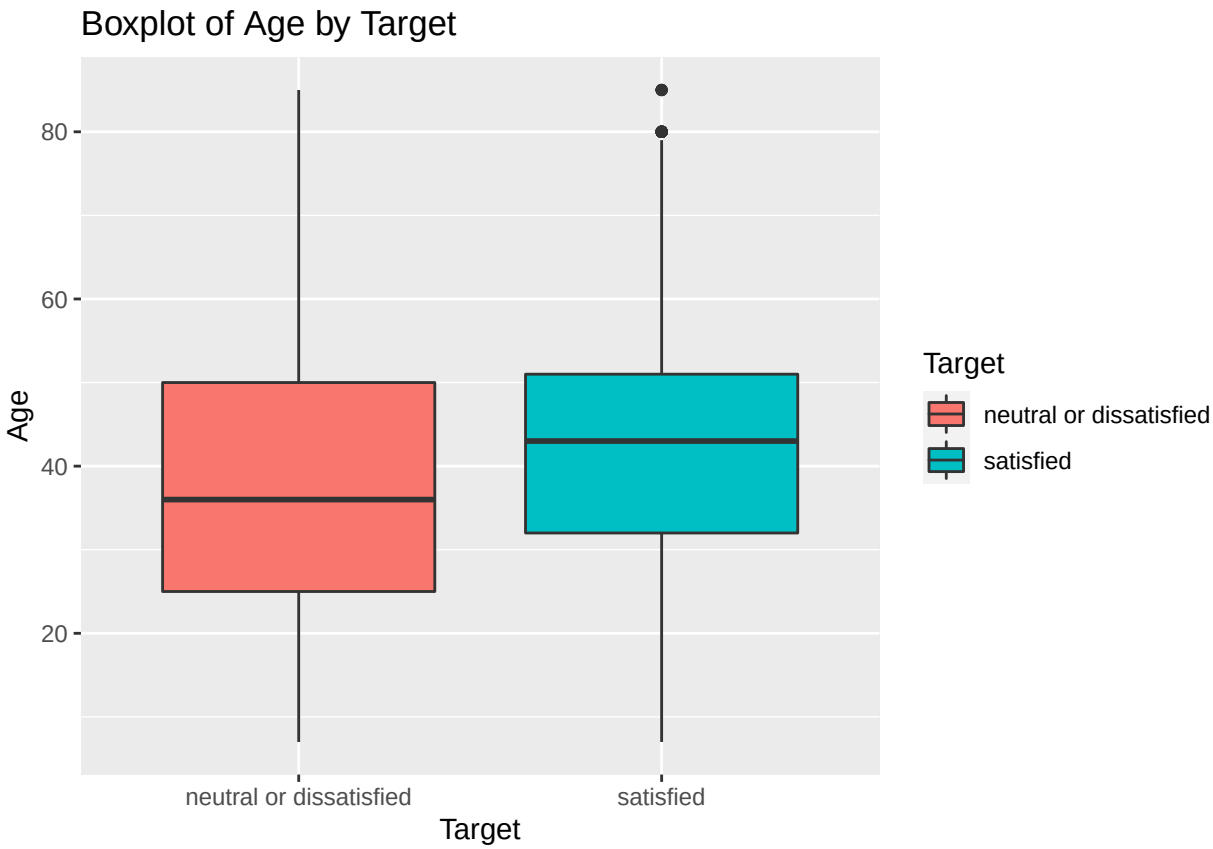
```
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
```

```
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
```

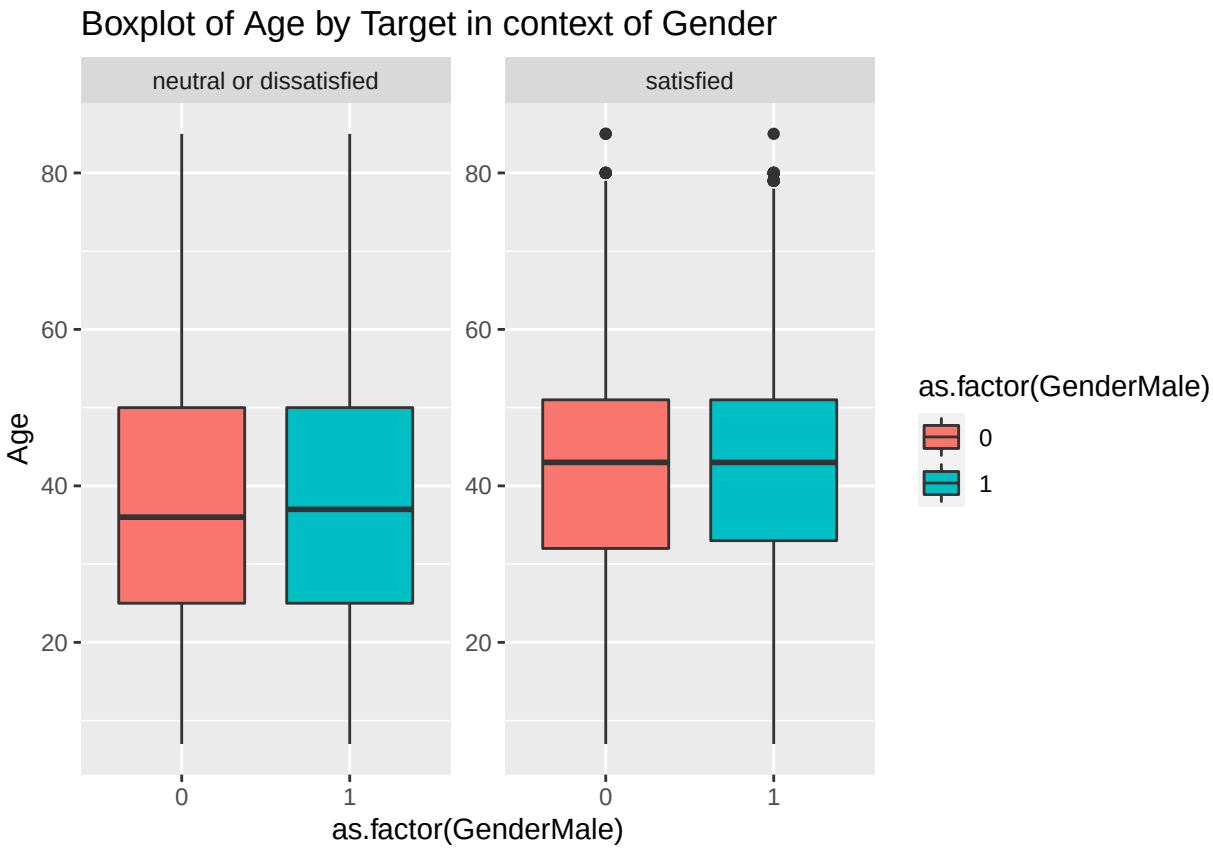
As you can see, there are no longer any null values and all of the data is still numeric except for the target variable. Hence, our data is clean. We can now begin our exploring our data visually.



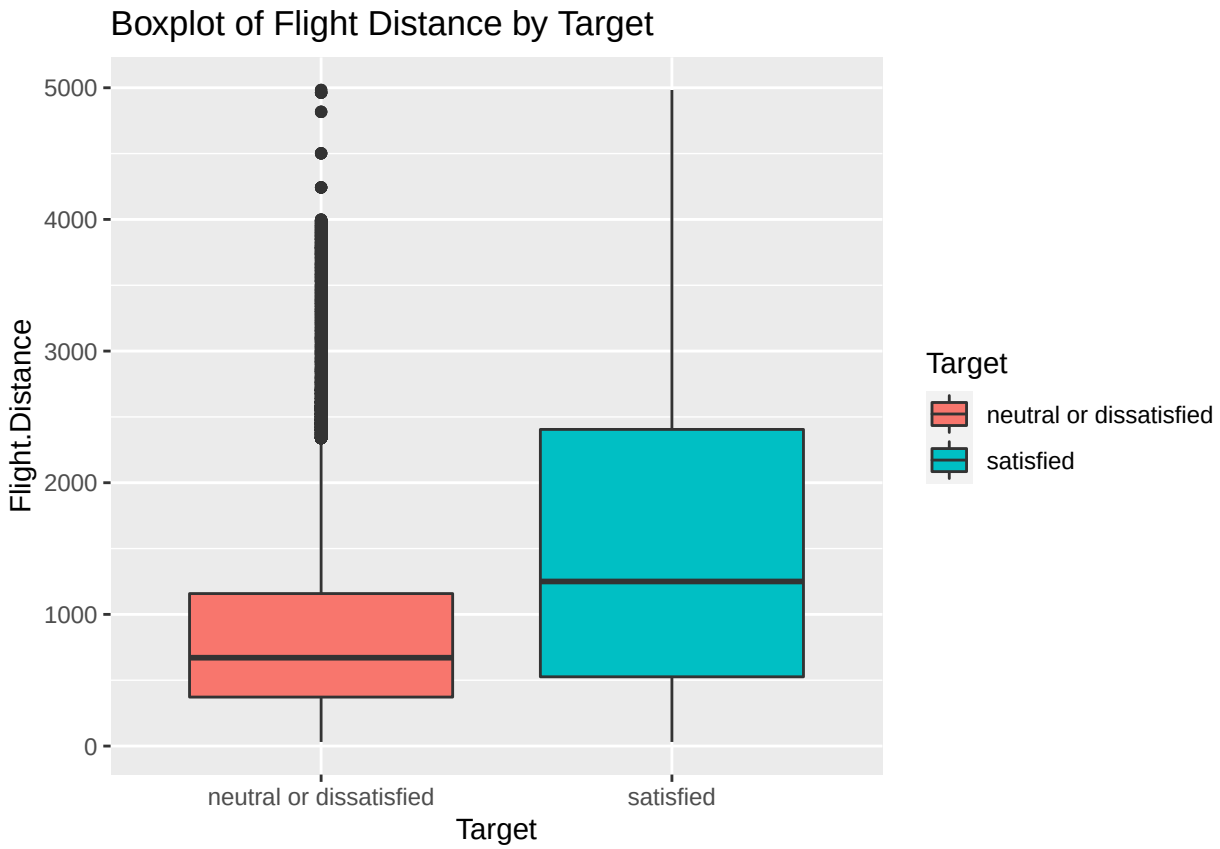
As you can see, there are more "neutral or dissatisfied" passengers than there are "satisfied". Let's take a look on how age plays in this.



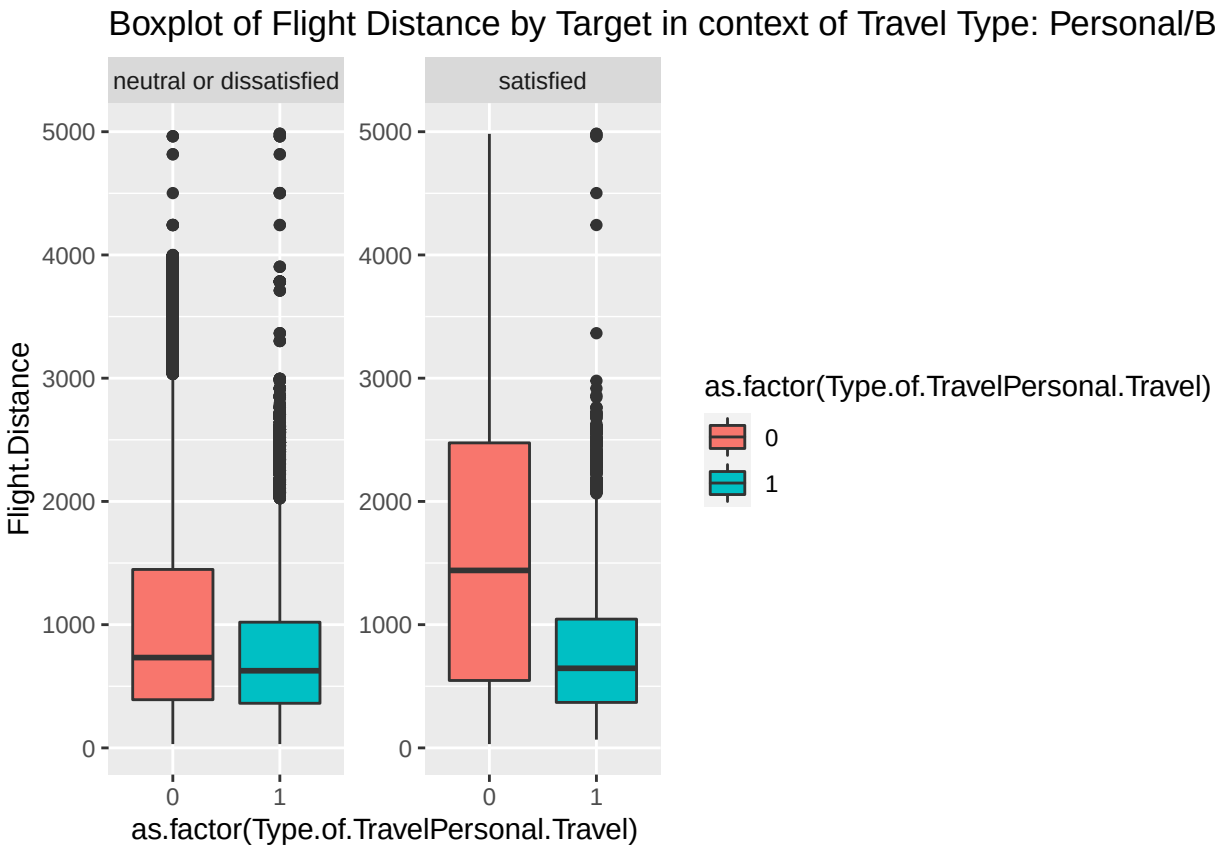
It seems that there are more of the younger crowd that claim to be "neutral or dissatisfied" whereas the majority of the "satisfied" customers are older than 30. Now let us compare this in the context of Gender.



Interestingly, Gender does not seem to affect the satisfaction rate of the passenger. Let's move on from demographics, and explore whether or not Flight Distance affects the satisfaction rate.



We can clearly see that shorter flights leave more passengers "neutral or dissatisfied".



Here you can see that Personal Travel satisfaction does not change as much depending on the Flight Distance, where as the Business Travelers are generally more satisfied than they are disatisfied over long distance flights.

Now that we have explored how our data interacts with one another, we have to split the data into two sets: the training data and the test data.

```
set.seed(101)

##First, we will choose a subset of our data to explore because the current dimensions are very large.
t1 <- sample(1:nrow(imptrain), nrow(imptrain)*0.05)
train.subset = imptrain[t1,]

##Now, we will split this subset into a training and testing set.
t2 <- sample(1:nrow(train.subset), nrow(train.subset)*0.7)
train.xy <- train.subset[t2,]
test.x <- subset(train.subset[-t2,], select = -Target)
test.y <- train.subset[-t2,]$Target
```

Now we will scale the numeric features

```
pre_proc_val <- preProcess(train.xy[,-c(24)], method = c("center", "scale"))
train.xy[,-c(24)] = predict(pre_proc_val, train.xy[,-c(24)])
test.x = predict(pre_proc_val, test.x)
```

Now we will use the training function in Caret to train our model with the following:

KNN

Decision tree

Random forest

Support vector machine

to find the best model, then test your modeling using the test data, reporting the accuracy and auc.

KNN Model

We will start by building a KNN model, determining the best k-value.

```
ktune <- train(train.xy[,-c(24)], train.xy$Target, method ="knn", tuneGrid =
  data.frame(.k = 1:10), trControl = trainControl(method = "cv"))
```

To see the result of the training print the following:

```
ktune
```

```
## k-Nearest Neighbors
##
## 3636 samples
##   23 predictor
##    2 classes: 'neutral or dissatisfied', 'satisfied'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3273, 3271, 3272, 3272, 3273, 3273, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.8809113  0.7557534
##    2  0.8740250  0.7422874
##    3  0.8976772  0.7896327
##    4  0.8979496  0.7903510
##    5  0.9042834  0.8032368
##    6  0.9004365  0.7950577
##    7  0.9018094  0.7977744
##    8  0.9015324  0.7971926
##    9  0.9042835  0.8027076
##   10  0.8990569  0.7918261
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

You can use the result of the training to perform prediction.

```
pred.ktune <- predict(ktune,test.x)
```

Now, let's determine the Accuracy and AUC of the model.

```
cm.ktune = confusionMatrix(pred.ktune, test.y)
cm.ktune
```

```
## Confusion Matrix and Statistics
##
##                          Reference
## Prediction                neutral or dissatisfied satisfied
##   neutral or dissatisfied                     851       121
##   satisfied                                    53       534
##
```

```
##                Accuracy : 0.8884
##                  95% CI : (0.8717, 0.9036)
##     No Information Rate : 0.5799
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7676
##
##  Mcnemar's Test P-Value : 3.789e-07
##
##             Sensitivity : 0.9414
##             Specificity : 0.8153
##          Pos Pred Value : 0.8755
##          Neg Pred Value : 0.9097
##              Prevalence : 0.5799
##          Detection Rate : 0.5459
##    Detection Prevalence : 0.6235
##       Balanced Accuracy : 0.8783
##
##        'Positive' Class : neutral or dissatisfied
##
```

```
#The accuracy of KNN Model
ktune.ac= cm.ktune$overall[1]
ktune.ac
```

```
## Accuracy
##  0.88839
```

```
#The AUC of KNN Model
ktune.auc= auc(pred.ktune, as.numeric(test.y))
```

```
## Setting levels: control = neutral or dissatisfied, case = satisfied
```

```
## Setting direction: controls < cases
```

```
ktune.auc
```

```
## Area under the curve: 0.8926
```

Decision Tree

We can use rpart with the train method from the caret library to find the best values for cp

10-folds cross validation

```
fitControl <- trainControl(method = 'cv', number=10)
```

We will attempt to find the best complexity parameter

```
Grid <- expand.grid(cp=seq(0, 0.05, 0.005))
```

Now we will run the training to determine the optimimal cp value.

```
trained_tree <- train(Target ~ . , data = train.xy, method = 'rpart',
trControl=fitControl, metric = 'Accuracy', maximize=TRUE, tuneGrid=Grid)
trained_tree
```

```
## CART
##
## 3636 samples
```

```
##   23 predictor
##    2 classes: 'neutral or dissatisfied', 'satisfied'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3272, 3273, 3273, 3273, 3272, 3272, ...
## Resampling results across tuning parameters:
##
##   cp     Accuracy   Kappa
##   0.000  0.9073025  0.8112998
##   0.005  0.9034450  0.8035369
##   0.010  0.8894227  0.7757383
##   0.015  0.8866754  0.7698149
##   0.020  0.8668703  0.7298630
##   0.025  0.8577975  0.7096057
##   0.030  0.8525739  0.6985500
##   0.035  0.8382708  0.6679536
##   0.040  0.8382708  0.6679536
##   0.045  0.8382708  0.6679536
##   0.050  0.8382708  0.6679536
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

Now we will use this model with the optimal cp value to perform a prediction and produce a confusion matrix.

```
pred.dt = predict(trained_tree, test.x, predictorstype="class")
cm.dt = confusionMatrix (pred.dt, test.y)
cm.dt
```

```
## Confusion Matrix and Statistics
##
##                          Reference
## Prediction                neutral or dissatisfied satisfied
##   neutral or dissatisfied                     837        71
##   satisfied                                    67       584
##
##                Accuracy : 0.9115
##                  95% CI : (0.8963, 0.9251)
##     No Information Rate : 0.5799
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8182
##
##  Mcnemar's Test P-Value : 0.7984
##
##             Sensitivity : 0.9259
##             Specificity : 0.8916
##          Pos Pred Value : 0.9218
##          Neg Pred Value : 0.8971
##              Prevalence : 0.5799
##          Detection Rate : 0.5369
##    Detection Prevalence : 0.5824
##       Balanced Accuracy : 0.9087
##
```

```
##        'Positive' Class : neutral or dissatisfied
##
```

```
#The accuracy of Decision Tree Model
dt.ac= cm.dt$overall[1]
dt.ac
```

```
##  Accuracy
## 0.9114817
```

```
#The AUC of Decision Tree Model
dt.auc= auc(pred.dt, as.numeric(test.y))
```

```
## Setting levels: control = neutral or dissatisfied, case = satisfied
```

```
## Setting direction: controls < cases
```

```
dt.auc
```

```
## Area under the curve: 0.9094
```

Random Forest

Now we will train a Random Forest model on our data.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:rattle':
##
##     importance
```

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

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

```
control <- trainControl(method="cv", number=10)

metric <- "Accuracy"
tunegrid <- expand.grid(.mtry=seq(1,4,1))
rf_default <- train(Target~., data = train.xy, method="rf",
metric=metric, tuneGrid=tunegrid, trControl=control)
```

```
print(rf_default)
```

```
## Random Forest
##
## 3636 samples
##   23 predictor
##    2 classes: 'neutral or dissatisfied', 'satisfied'
##
```

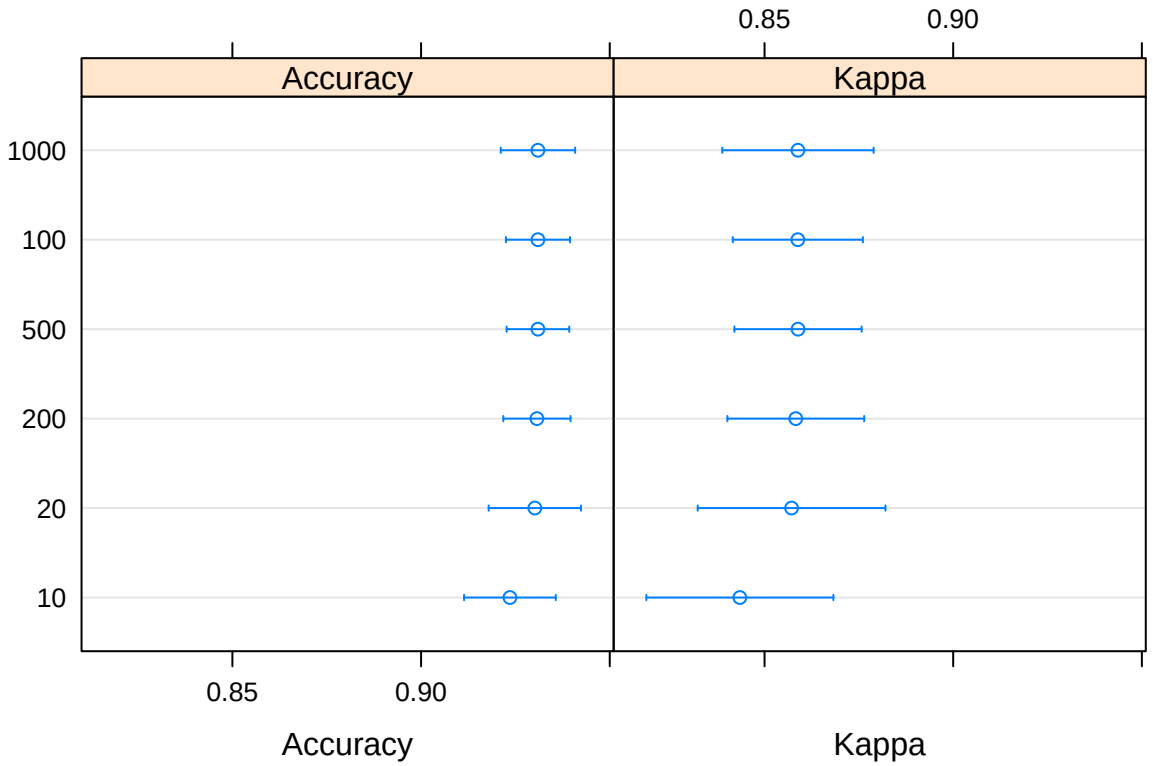
```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3272, 3271, 3273, 3273, 3272, 3272, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.9065024  0.8070220
##   2     0.9243747  0.8449894
##   3     0.9276737  0.8519480
##   4     0.9298693  0.8565559
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

```
mtry = as.integer(rf_default$bestTune$mtry)
```

We will use the above optimal number of variables randomly picked for the split.

```
control <- trainControl(method="cv", number=10, search="grid")
tunegrid <- expand.grid(.mtry=seq(1,4,1))
modellist <- list()
for (ntree in c(10, 20, 100, 200, 500, 1000)) {
    set.seed(1)
    fit <- train(Target~., data = train.xy, method="rf", metric=metric, tuneGrid=tunegrid, trControl=co
    key <- toString(ntree)
    modellist[[key]] <- fit
}
```

```
# compare results
results <- resamples(modellist)
dotplot(results)
```



Confidence Level: 0.95

```
sr = summary(results)
sr
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: 10, 20, 100, 200, 500, 1000
## Number of resamples: 10
##
## Accuracy
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 10   0.8928571 0.9175824 0.9243446 0.9235461 0.9368995 0.9449036    0
## 20   0.8983516 0.9250901 0.9340659 0.9301463 0.9436092 0.9476584    0
## 100  0.9173554 0.9196429 0.9300354 0.9309765 0.9407713 0.9504132    0
## 200  0.9090909 0.9217033 0.9300354 0.9306995 0.9407713 0.9476584    0
## 500  0.9118457 0.9265110 0.9314128 0.9309742 0.9380165 0.9476584    0
## 1000 0.9118457 0.9230769 0.9300391 0.9309780 0.9414601 0.9504132    0
##
## Kappa
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 10   0.7811151 0.8311268 0.8448136 0.8434591 0.8709424 0.8869053    0
## 20   0.7926469 0.8461560 0.8655006 0.8571615 0.8845621 0.8928055    0
## 100  0.8312523 0.8358735 0.8570919 0.8588216 0.8785159 0.8986788    0
## 200  0.8141033 0.8402935 0.8568683 0.8582816 0.8787370 0.8929685    0
## 500  0.8198735 0.8496108 0.8596227 0.8588729 0.8731939 0.8929685    0
## 1000 0.8198735 0.8424431 0.8568664 0.8588588 0.8801755 0.8986788    0
```

```
order = order(sr$statistics$Accuracy[,6], decreasing = TRUE)
#optimal number of trees is:
ntrees = as.integer(rownames(sr$statistics$Accuracy)[order[1]])
ntrees
```

```
## [1] 100
```

```
#with an accuracy of:
sr$statistics$Accuracy[order[1],6]
```

```
## [1] 0.9504132
```

We will use the above number as the optimal number of trees and the optimal mtry.

```
rfModel <- randomForest(Target~., data = train.xy,ntree=ntrees,mtry=mtry)
```

Now we will do the actual prediction and produce a confusion matrix.

```
pred.rf = predict(rfModel,test.x, predictorstype = 'class')
cm.rf = confusionMatrix (pred.rf, test.y)
cm.rf
```

```
## Confusion Matrix and Statistics
##
##                          Reference
## Prediction                neutral or dissatisfied satisfied
##   neutral or dissatisfied                     857        59
##   satisfied                                    47       596
##
##                Accuracy : 0.932
##                  95% CI : (0.9184, 0.944)
##     No Information Rate : 0.5799
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8601
##
##  Mcnemar's Test P-Value : 0.2853
##
##             Sensitivity : 0.9480
##             Specificity : 0.9099
##          Pos Pred Value : 0.9356
##          Neg Pred Value : 0.9269
##              Prevalence : 0.5799
##          Detection Rate : 0.5497
##    Detection Prevalence : 0.5876
##       Balanced Accuracy : 0.9290
##
##        'Positive' Class : neutral or dissatisfied
##
```

```
#The accuracy of Random Forest Model
rf.ac = cm.rf$overall[1]
rf.ac
```

```
##  Accuracy
## 0.9320077
```

```
#The AUC of Random Forest Model
rf.auc= auc(pred.rf, as.numeric(test.y))
```

```
## Setting levels: control = neutral or dissatisfied, case = satisfied
```

```
## Setting direction: controls < cases
```

```
rf.auc
```

```
## Area under the curve: 0.9312
```

Support vector machine

We will use the train method from the caret library to figure out which svm Kernel (linear, RBF) will work best with the training data.

To train svm for a Linear kernel using a 10-folds cross validation.

```
L_model <- train(Target~., data = train.xy, method="svmLinear",
trControl=trainControl(method='cv',number = 10))
```

To train svm for a Radial kernel using a 10-folds cross validation.

```
R_model <- train(Target~., data = train.xy,method="svmRadial",
trControl=trainControl(method='cv', number = 10))
```

To see the result of the training you can do

```
L_model
```

```
## Support Vector Machines with Linear Kernel
##
## 3636 samples
##   23 predictor
##    2 classes: 'neutral or dissatisfied', 'satisfied'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3272, 3273, 3272, 3273, 3272, 3272, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8820006  0.7583989
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
R_model
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 3636 samples
##   23 predictor
##    2 classes: 'neutral or dissatisfied', 'satisfied'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3273, 3273, 3272, 3272, 3273, 3272, ...
## Resampling results across tuning parameters:
##
```

```
##   C     Accuracy   Kappa
##   0.25  0.9097901  0.8144646
##   0.50  0.9202425  0.8365702
##   1.00  0.9251967  0.8468571
##
## Tuning parameter 'sigma' was held constant at a value of 0.02863204
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.02863204 and C = 1.
```

```
lm = L_model$results
rm = R_model$results

order.lm = order(lm$Accuracy, decreasing = TRUE)
order.rm = order(rm$Accuracy, decreasing = TRUE)

#Linear Kernel Accuracy
lm$Accuracy[order.lm[1]]
```

```
## [1] 0.8820006
```

```
#RBF Kernel Accuracy
rm$Accuracy[order.rm[1]]
```

```
## [1] 0.9251967
```

```
if(lm$Accuracy[order.lm[1]]>rm$Accuracy[order.rm[1]]){print("The Linear Kernel performs better than the
```

```
## [1] "The RBF Kernel performs better than the Linear Kernel."
```

```
if(lm$Accuracy[order.lm[1]]>rm$Accuracy[order.rm[1]]){optimal.kernel = L_model}else{optimal.kernel = R_
```

We will use it to predict on our test data.

```
pred.svm <- predict(optimal.kernel, test.x)
#Confusion Matrix of SVM Model
cm.svm = confusionMatrix (pred.svm, test.y)

#The accuracy of KNN Model
svm.ac = cm.svm$overall[1]
svm.ac
```

```
##  Accuracy
## 0.9166132
```

```
#The AUC of KNN Model
svm.auc= auc(pred.svm, as.numeric(test.y))
```

```
## Setting levels: control = neutral or dissatisfied, case = satisfied
```

```
## Setting direction: controls < cases
```

```
svm.auc
```

```
## Area under the curve: 0.9169
```

Now let's check and see which model performed best.

```
ac=data.frame(ktune.ac, dt.ac, rf.ac, svm.ac)
auc=data.frame(ktune.auc, dt.auc, rf.auc, svm.auc)
order.ac = order(ac, decreasing = TRUE)
order.auc = order(auc, decreasing = TRUE)
```

```
#The model with the best accuracy is:
ac[order.ac[1]]
```

```
##              rf.ac
## Accuracy 0.9320077
```

```
#The model with the best AUC is:
auc[order.auc[1]]
```

```
##      rf.auc
## 1 0.9312473
```

So in this case, we can use this model to accurately predict the satisfaction statistic of a passenger at the above accuracy rate.

Also, the AUC tells us how well the model is at distinguishing between satisfaction classes. The model with the highest AUC has the best ability to predict whether a passenger is "satisfied" or "neutral or dissatisfied".

Model Selection using LASSO

Now we will use LASSO for model selection to determine what the most important predictor variables are for determing passenger satisfaction. LASSO regression also helps in reducing over-fitting.

I'll use 10-fold CV.

```
train_control <- trainControl(
  method = "cv", number = 10,
  savePredictions = "final",
  classProbs = TRUE
)
```

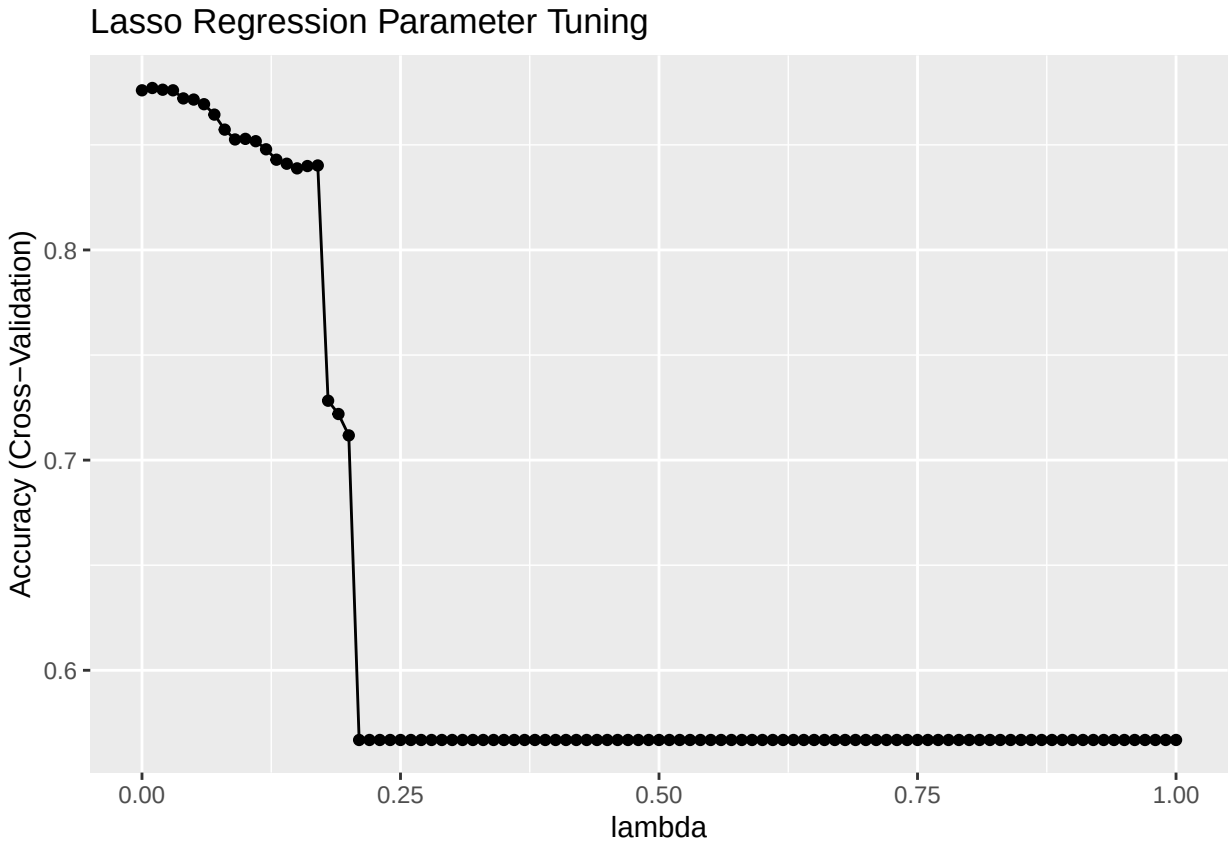
We will tune to determine the optimal lambda value for our Lasso regression.

```
set.seed(1970)
mdl_lasso <- train(
  as.matrix(train.xy[-c(24)]),
  make.names(train.xy$Target),
  method = "glmnet",
  family = "binomial",
  tuneGrid = expand.grid(
    .alpha = 1,  # optimize a lasso regression
    .lambda = seq(0, 1, length.out = 101)
  ),
  trControl = train_control,
  metric = "Accuracy"
)
mdl_lasso$bestTune
```

```
##   alpha lambda
## 2     1   0.01
```

Here we have the above lambda as our optimal tuning parameter. You can see in the below plot how the accuracy maximizes at this point.

```
ggplot(mdl_lasso) +
  labs(title = "Lasso Regression Parameter Tuning", x = "lambda")
```



Now we can see the most important variables in the Lasso Regression.

```
var = varImp(mdl_lasso)
var
```

```
## glmnet variable importance
##
##   only 20 most important variables shown (out of 23)
##
##                                    Overall
## Type.of.TravelPersonal.Travel     100.000
## Online.boarding                    70.327
## Customer.TypeLoyal.Customer        52.820
## Checkin.service                    36.072
## Inflight.wifi.service              29.102
## ClassEco                           28.562
## On.board.service                   24.992
## Leg.room.service                   24.247
## Departure.Arrival.time.convenient  14.215
## Inflight.entertainment             14.042
## ClassEco.Plus                      13.975
## Baggage.handling                   13.861
## Inflight.service                   11.500
## Cleanliness                        11.181
## Arrival.Delay.in.Minutes            5.149
## Seat.comfort                        4.799
## Gate.location                       2.179
## Ease.of.Online.booking              1.677
```

```
## Food.and.drink                       0.000
## GenderMale                           0.000
```

```
order.var = order(var$importance, decreasing = TRUE)[1:10]
predictors = rownames(var$importance)[order.var]
#The most important variables are:
predictors
```

```
##  [1] "Type.of.TravelPersonal.Travel"     "Online.boarding"
##  [3] "Customer.TypeLoyal.Customer"       "Checkin.service"
##  [5] "Inflight.wifi.service"             "ClassEco"
##  [7] "On.board.service"                  "Leg.room.service"
##  [9] "Departure.Arrival.time.convenient" "Inflight.entertainment"
```

We will use the first ten predictors based on importance when we run our future models.

```
confusionMatrix(mdl_lasso)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##                          Reference
## Prediction                neutral.or.dissatisfied satisfied
##   neutral.or.dissatisfied                    51.4       7.0
##   satisfied                                   5.3      36.3
##
##  Accuracy (average) : 0.8771
```

So our model has the above accuracy on our training dataset. Let's see how it performs with our validation set.

```
predicted_classes <- predict(mdl_lasso, newdata = test.x)
cm.lr = table(predicted_classes, test.y)
cm.lr
```

```
##                          test.y
## predicted_classes         neutral or dissatisfied satisfied
##   neutral.or.dissatisfied                     814       115
##   satisfied                                    90       540
```

This has ~88.08% accuracy on our validation set. We will now set the ten most signficant variables as our predictors.

```
predictors = noquote(predictors)
preds = ""
for (pred in predictors){
        preds = paste(preds, "+", pred)
}
preds = noquote(substring(preds,4))
formula = formula(paste(noquote("Target ~"), preds))
```

Using the selected variables from LASSO, we will run the following models:

Logistic regression

Random Forest

Nueral Network

Logistic Regression

We will focus on the selected predictors that we used after the Lasso Model selection. We will focus on these ten variables when applying Logistic Regression, Random Forest, and Nueral Network.

```
logit <- glm(formula, data = train.xy, family = "binomial")
summary(logit)
```

```
##
## Call:
## glm(formula = formula, family = "binomial", data = train.xy)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5683  -0.4786  -0.1512   0.3867   3.6894
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      -0.56393    0.05523 -10.211  < 2e-16 ***
## Type.of.TravelPersonal.Travel    -1.28386    0.07368 -17.425  < 2e-16 ***
## Online.boarding                   0.89764    0.07013  12.799  < 2e-16 ***
## Customer.TypeLoyal.Customer       0.72322    0.05559  13.010  < 2e-16 ***
## Checkin.service                   0.60525    0.05783  10.467  < 2e-16 ***
## Inflight.wifi.service             0.45664    0.06719   6.796 1.07e-11 ***
## ClassEco                         -0.32354    0.06123  -5.284 1.26e-07 ***
## On.board.service                  0.45285    0.06211   7.292 3.06e-13 ***
## Leg.room.service                  0.36758    0.05868   6.264 3.76e-10 ***
## Departure.Arrival.time.convenient -0.32114    0.05878  -5.463 4.67e-08 ***
## Inflight.entertainment            0.36289    0.06258   5.799 6.67e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4975.4  on 3635  degrees of freedom
## Residual deviance: 2382.0  on 3625  degrees of freedom
## AIC: 2404
##
## Number of Fisher Scoring iterations: 6
```

As you can see, the significant variables are in the same order as our Lasso Regression which confirms their significance. We will now use this model to predict on our validation dataset.

```
outlr = predict(logit, test.x, type="response")
```

Below we are going to assign our labels with decision rule that if the prediction is greater than 0.5, assign it 1 else 0. We will then perform our prediction on our test data and give Accuracy and AUC of our model.

```
log.prediction <- ifelse(outlr > 0.5, 1, 0)
levels(test.y) <- list("0"="neutral or dissatisfied", "1"="satisfied")
cm.lr = confusionMatrix(factor(log.prediction), test.y)
cm.lr
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 805 115
##          1  99 540
##
##                Accuracy : 0.8627
##                  95% CI : (0.8446, 0.8794)
##     No Information Rate : 0.5799
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7173
##
##  Mcnemar's Test P-Value : 0.3052
##
##             Sensitivity : 0.8905
##             Specificity : 0.8244
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.8451
##              Prevalence : 0.5799
##          Detection Rate : 0.5164
##    Detection Prevalence : 0.5901
##       Balanced Accuracy : 0.8575
##
##        'Positive' Class : 0
##
```

```
#The accuracy of Logistic Regression Model using predictor variables
lr.ac = cm.lr$overall[1]
lr.ac
```

```
##  Accuracy
## 0.8627325
```

```
#The AUC of Logistic Regression Model using predictor variables
lr.auc= auc(factor(log.prediction), as.numeric(test.y))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
lr.auc
```

```
## Area under the curve: 0.86
```

Random Forest

```
library(randomForest)
```

We already know the optimal mtry and ntrees to use for our random forest from our previously run model, so we will use those here.

```
rfModel2 <- randomForest(formula, data = train.xy,ntree=ntrees,mtry=mtry)
```

Now we will do the actual prediction on our test data and produce a confusion matrix.

```
outrf2 = predict(rfModel2,test.x, predictorstype = 'class')
levels(outrf2) <- list("0"="neutral or dissatisfied", "1"="satisfied")
cm.rf2 = confusionMatrix(outrf2, test.y)
cm.rf2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 856  57
##          1  48 598
##
##                Accuracy : 0.9326
##                  95% CI : (0.9191, 0.9446)
##     No Information Rate : 0.5799
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8615
##
##  Mcnemar's Test P-Value : 0.435
##
##             Sensitivity : 0.9469
##             Specificity : 0.9130
##          Pos Pred Value : 0.9376
##          Neg Pred Value : 0.9257
##              Prevalence : 0.5799
##          Detection Rate : 0.5491
##    Detection Prevalence : 0.5856
##       Balanced Accuracy : 0.9299
##
##        'Positive' Class : 0
##
```

```
#The accuracy of the Random Forest Model using predictor variables
rf2.ac = cm.rf2$overall[1]
rf2.ac
```

```
##  Accuracy
## 0.9326491
```

```
#The AUC of the Random Forest Model using predictor variables
rf2.auc= auc(outrf2, as.numeric(test.y))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
rf2.auc
```

```
## Area under the curve: 0.9316
```

```
if(rf2.ac>rf.ac){print("The second Random Forest model that utilizes the predictor variables determined
```

```
## [1] "The second Random Forest model that utilizes the predictor variables determined by LASSO is mor
```

Neural Network

```
grid <- expand.grid(size=seq(from = 5, to = 17, by = 1), decay=c(0,0.01,0.1,1))
model.nn<-train(formula, data=train.xy, method="nnet", tuneGrid=grid,skip=FALSE,linout=FALSE)
model.nn
```

```r
#Here, we are using our optimized size and decay to obtain the below accuracy with our model.
model.nn$bestTune
```

```
##    size decay
## 52   17     1
```

```r
test.pred <- as.factor(predict(model.nn$finalModel, test.x, type="class"))
levels(test.pred) <- list("0"="neutral or dissatisfied", "1"="satisfied")
cm.nn <- confusionMatrix(test.pred, test.y)
cm.nn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 844  68
##          1  60 587
##
##                Accuracy : 0.9179
##                  95% CI : (0.9031, 0.931)
##     No Information Rate : 0.5799
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8312
##
##  Mcnemar's Test P-Value : 0.5361
##
##             Sensitivity : 0.9336
##             Specificity : 0.8962
##          Pos Pred Value : 0.9254
##          Neg Pred Value : 0.9073
##              Prevalence : 0.5799
##          Detection Rate : 0.5414
##    Detection Prevalence : 0.5850
##       Balanced Accuracy : 0.9149
##
##        'Positive' Class : 0
##
```

```r
#The accuracy of Neural Network Model using predictor variables
nn.ac = cm.nn$overall[1]
nn.ac
```

```
##  Accuracy
## 0.9178961
```

```r
#The AUC of Neural Network Model using predictor variables
nn.auc= auc(test.pred, as.numeric(test.y))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
nn.auc
```

```
## Area under the curve: 0.9164
```

Now let's check and see which final model performed best.

```
ac2=data.frame(lr.ac, rf2.ac, nn.ac)
auc2=data.frame(lr.auc, rf2.auc, nn.auc)
order.ac2 = order(ac2, decreasing = TRUE)
order.auc2 = order(auc2, decreasing = TRUE)
#The model with the best accuracy is:
ac2[order.ac2[1]]
```

```
##             rf2.ac
## Accuracy 0.9326491
```

```
#The model with the best AUC is:
auc2[order.auc2[1]]
```

```
##     rf2.auc
## 1 0.9316325
```

```
if(colnames(ac2[order.ac2[1]]) == "rf2.ac"){print("The Random Forest model performs best. The below con
        c(cm.rf2$overall[3], cm.rf2$overall[4])}else if(colnames(ac2[order.ac2[1]]) == "lr.ac"){print("
        c(cm.lr$overall[3], cm.lr$overall[4])}else{print("The Neural Network model performs best. The b
        c(cm.nn$overall[3], cm.nn$overall[4])}
```

```
## [1] "The Random Forest model performs best. The below confidence interval confirms that we are 95% c
```

```
## AccuracyLower AccuracyUpper
##     0.9190528     0.9445865
```

Now we have our final, optimized model with it's best performing accuracy and AUC, using the ten most important predictors determined in LASSO regression. We can use this model to most accurately predict the satisfaction statistic of a passenger within the above confidence interval.

Also, the AUC tells us how well the model is at distinguishing between satisfaction classes. The above model with the highest AUC has the best ability to predict whether a passenger is "satisfied" or "neutral or dissatisfied".

This optimal model is recomended for further exploration of this data. Further model builds on the full dataset may provide a more accurate result.