Eric_Hirsch_622_Final_Assignment

Predicting the Space Titanic Kaggle Competition

Eric Hirsch

12/11/2022

Contents

```
knitr::opts_chunk$set(echo = TRUE, warning =  FALSE, message = FALSE)
```

```
dfTable <- read.csv("D:\\Rstudio\\Cuny_622\\Space\\622a.csv")
```

Discussion

Introduction

In machine learning, we predict target variables based on input variables. For this final exercise, we will apply various machine learning algorithms to a Kaggle data set (Spaceship Titanic) in order to predict which passengers have been transported to another dimension.

While it's tempting to throw as many algorithms at the problem as possible to see what sticks, the statistical fact is that while it is rare that a poor model will perform well on a holdout set, the chances of making false conclusions based on performance increases if we simply try one model after another. Besides, if we don't understand our model and our data, and the model becomes much more difficult to troubleshoot and maintain.

When choosing models, we are balancing simplicity and complexity, and therefore tendencies to underfit or overfit. When the relationships in the data are simple and certain statistical conditions are met, parametric methods like OLS work well and have the advantage of being easily interpretable. If, for example, we are predicting height from weight, the relationship is simple enough that we can create a linear regression model and capture most of the variation that can be explained for these two variables.

When we increase our dimensions and/or complexity of relationships within the dataset, parametric methods are likely to underfit the data. Even in our simple height and weight example, if the relationship between height and weight varies considerably at lower weights, medium weights and higher weights, spline regression or another nonparametric technique will be necessary. As dimensions and complexity increases, we adopt techniques that are more powerful at morphing the data shape so that we can model the underlying structure, such as trees, SVM and neural nets.

Choosing the more complex algorithm will likely fit the training data better, but may be less interpretable and more subject to overfitting. With this in mind, each of these techniques has its advantages and disadvantages. In my experience with earlier datasets in this class, trees will pick up autonomous clusters in the data set better than SVMs. For example, if there were a small but significant anomalous cluster of individuals for whom height and weight were inversely related, trees will incorporate the cluster while SVMs will ignore it. Of course, clusters like this might signal a missing variable, but not all of the necessary variables will be found in any given data set. On the other hand, when the relationships are more systematic and class boundaries are clear, SVMs may perform better because the kernel trick allows SVMs to radically change the data shape in order to find the class boundary. SVMs can also perform better when there is less data.

While decision trees are powerful, they are generally more so when bagged (e.g., Random Forest) or boosted (e.g. xgBoost). Because xgBoost is an active learner, it will often have the upper hand in fitting the training data.

One of the biggest advantages of neural networks is that they effectively do the feature engineering for you if you can apply enough layers. They are also subject to the "double descent" phenomenon, which helps with managing underfitting. However, for a student using a home computer like myself, it's often impractical to take advantage of these facts as the algorithm would run too long. Neural networks, like SVMs, also powerfully change the data shape in order to find class boundaries.

Accurate prediction depends not only on algorithm choice. We also need to engineer features (except possibly in very large neural nets) and tune hyperparameters. We also need to choose metrics that tell us whether or not our model is effective.

Prediction using the Kaggle Spaceship Titanic Data Set

For this exercise I've chosen a Kaggle Competition – the Kaggle Spaceship data set. The advantages of using this a competition data set are that we can compare our performance to those of others. Achieving 90% on a holdout set in and of itself tells us nothing - we don't know if achieving 95% would have been easy or impossible. In this competition, the 2,000 or so submitted accuracies on the leaderboard range from about 76% to 82%, which can give us a good idea of how well our model is working.

The main disadvantages of this data set are that the data is made up and the scenario a bit far-fetched. However, I wanted a data set that had a simple class as a target, as opposed to an image example. The standard Titanic data set has been over analyzed, so this was one of the few good choices left.

The Business Problem

In the year 2912, the Spaceship Titanic, an interstellar passenger liner with almost 13,000 passengers on board, collided with a spacetime anomaly hidden within a dust cloud. Though the ship stayed intact, almost half of the passengers were transported to an alternate dimension. Our job is to predict which passengers were transported by the anomaly using a set of partial records recovered from the spaceship's damaged computer system.

Data Summary and Preparation

The data set consists of 8693 records and 13 variables, including spending on the ship's various amenities (VR Deck, Spa, Room Service, Food Court, and Shopping Mall), cabin number, whether the individual was traveling with the group, whether the individual was a VIP, planet of origin and destination, and so on. These columns map to some degree with the original Titanic database.

The target variable, Transported, is roughly equally distributed between false (4315) and true (4378).

Some of the numeric variables, particularly spending variables, are highly skewed - most passengers spend no money while a few spend a great deal. We can see that spending on luxuries (the spa, room service, etc.) is strongly negatively correlated with being transported - this supports the supposition that the rich were spared. Spending on more budget–friandly amenities like the food court and shopping mall are also negatively correlated but less so. Age has a small negative correlation as well.

We decide not to log transform the numeric variables as normal distributions for predictors are not required by our models and interpretability suffers.

Missing Values 1073, or 12%, of records have missing values. The vast majority of missing values are found in the amenity expenditure columns. Oddly, the amenity expenditure rows with missing values are completely independent of each other - there are almost no records where more than one of these values is missing.

This may be an artifact of the fact that the data is manufactured. In order to confirm that there is no systematic relationship between missing data and the target variable, we look at the Chi square between the target and a flag designating missing data. We do this for each amenity expenditure column and find no relationship between missing data and the target variable. We therefore eliminate rows with missing values for the training set. For the test set, we impute the median.

There is very little, even surprisingly little, multicollinearity in the database. In the case of variables that track spending on amenities this is most surprising, and may suggest that passengers were working within a budget and only spent money on the activities they liked most.

Outliers All of the spending variables are highly skewed, with very large instances occurring at the very end of the distribution. However, as most of our techniques are robust for outliers, records with extreme values remain in the database, as there is no reason to think that the spending is a data entry error or an anomalous occurrence.

Feature Engineering The data set holds a number of opportunities for feature engineering. Through testing, it was found that the following new features were significant in predicting transportation. They are:

1. Groups - Passenger ID numbers suggest that some passengers boarded the ship as part of a group. Although we could not establish that members of a group tended to meet the same fate, being a member of a group influenced whether a passenger was transported.
2. Cabins - Cabin codes were parsed for location n the ship - this information was correlated with the target variable.
3. Interaction variables - there were a number of interactions among variables which appeared when the variables were examined in isolation. We only retained one (group passengers were more likely to be transported if they shopped at the mall) because the others did not significantly increase accuracy when running the overall model - but there would be more to explore here.

At this point, a picture of the transported passengers begins to emerge - they are the poorer passengers, most likely to shop on a budget or, even cheaper, spend the trip in cryosleep. They tend to enter the ship in groups and inhabit lower class cabins.

Modelling

Choosing and Testing Models Understanding what models are doing and how is a key part of prediction. We compared tree models, svm, neural net, and logistic regression.

The first task is to understand the requirements of the business problem. In this case, we have a partial roster of passengers where we know who was transported and who wasn't. As for the rest of the passengers, it is our job to predict which of them were transported as well. Insight into the data is necessary insofar as it helps us make better predictions, but we need no more from the data than that. Accuracy is our metric, as this is the metric used in the Kaggle (e.g., there is no penalty for false positives or false negatives which might suggest a different metric). This suggests we should use the most complex model we can that has the highest accuracy.

We used tenfold cross validation and compared models on the metric of accuracy – our results varied widely. There was some, but not perfect, match between the accuracy predicted by the cross validation, and the accuracy achieved in the Kaggle. For example, going on cross validation alone, the svm with the radial kernel underperformed compared to a linear and polynomial kernel - however, in the Kaggle it significantly outperformed both kernels. Of course, without the Kaggle we would not have known this and would not have chosen the radial kernel to put into production.

On the other hand, our best model on cross validation (xgboost) also performed best on the Kaggle. We suspect the neural net would have performed well also, but we did not have the computer power to hypertune it properly.

Hyperparameter Tuning The caret package in R automatically hypertunes models on basic parameters and chooses the most optimal based on the metric determined by the user. With the exception of xgBoost, all of our models were tuned that way. This left two tasks – tune xgBoost, and experiment with manual tuning of other models.

The tuned xgBoost model (number of rounds was reduce from 55 to 14) increased accuracy from 79% to 80%, and improved our percentile position in the Kaggle from the 73rd to the 78th percentile.

The fact that our radial-kernal SVM had lower cross validation accuracy but higher Kaggle accuracy might have a number of different causes - but one is the possibility that there are idiosyncrasies in the training set that are not found in the Kaggle set. In this case, we can increase sigma to decrease the risk of overfitting.

As we experimented by increasing sigma and decreasing sigma on the radial SVM model. Reducing sigma (tighter fit) improved accuracy during cross validation, but not when applied to the Kaggle. Because the differences were very small (.0026 improvement in accuracy), the models probably only varied by a handful of predictions. Increasing sigma (looser fit) reduced accuracy both in cross validation and on the Kaggle set.

Results

```
knitr::kable(dfTable)
```

Model	Holdout.Set.Accuracy	Kaggle.Accuracy	Kaggle.Percentile
XGBoost Tuned (14 Rounds)	0.8000	0.80383	78.2%
XGBoost Untuned (55 Rounds)	0.7900	0.80243	73.8%
SVM Rad	0.7715	0.80149	70.0%
SVM Rad Lower Sigma	0.7741	0.80032	61.1%
SVM Rad Higher Sigma	0.7669	0.79682	57.3%
SVM Linear	0.7800	0.79775	56.8%
SVM Poly	0.7800	0.79611	55.3%
Random Forest	0.7900	0.78793	34.5%
Logistic Regression with feature engineering	0.7800	0.77975	26.2%
Neural Net (severely limited due to computer power)	0.7600	0.77881	25.5%
Logistic Regression without feature engineering	0.7800	0.69227	9.5%

Discussion

For this exercise, we entered a Kaggle competition (Space Titanic) in order to make predictions and test their accuracy against other competition participants. The target variable was whether or not a passenger was transported to another dimension while the ship was moving through a dangerous and destructive space anomaly. We reached the 78th percentile, with an idea of further modifications which might help us do better.

In a sense, measuring performance in a Kaggle is a cheat - after a time the "unknown" data in the Kaggle test set becomes increasingly "known" as you test more and more models against it. However, it also provides a real-world check on how well your model is actually performing.

After an initial logistic regression, we improved performance in three ways:

1. Feature Engineering
2. Model Selection
3. Hypertuning

The exercise highlighted the importance of Understanding the data in solid and dependable prediction. First, we needed to clean the data of anomalies (missing values, outliers, etc.). In our case, missing values appear to be at random and outliers didn't present a problem.

Second, an understanding of the data highly conditioned our ability to perform feature engineering. Transported passengers tended to spend their money at the food court rather than get room service, they occupied lower-class cabins, and many of them were in cryosleep during the journey. Knowing this helped us parse out cabin locations and discover interaction terms within the data.

All of our models came in between 77% and 80% accuracy (based on a no information rate of 51%) after tenfold cross validation and optimal hypertuning. This was also the range for individuals who submitted predictions in the Kaggle competition - and so a score of 77% achieved a much lower percentile in the competition than a score a few percentage points higher. Without the Kaggle, we still would have chosen the xgBoost model, but if we were going to choose an SVM we would likely have chosen a less than optimal kernel.

While it makes sense that the boosted tree model outperforms the bagged tree model, it more difficult to speculate on why tree models perform better than the Support Vector Machines. Given the small difference in accuracy between them (.003), it is possible that there is no significant difference, and that with slightly different feature engineering the SVM would be the better model.

There would be a number of ways to improve accuracy even further:

1. Search for more features, particularly interaction features between cryosleep and spending, and understanding better how groupings and cabins work.
2. Borrow a more powerful computer to perform a properly tuned neural net, and then create a prediction set based on neural net, xgBoost and radial SVM.
3. Experiment more with manual tuning of models.

In the meantime, however, we are satisfied with the nearly 80th percentile compared to the 26th percentile using logistical regression which we would have achieved prior to this class.

```
#dfTrain <- read.csv("C:\\Users\\erico\\Documents\\R\\Space\\train.csv")
#dfTest <- read.csv("C:\\Users\\erico\\Documents\\R\\Space\\test.csv")

dfTrain <- read.csv("D:\\Rstudio\\Cuny_622\\Space\\train.csv")
dfTest <- read.csv("D:\\Rstudio\\Cuny_622\\Space\\test.csv")


dfTrain$Transported = ifelse(dfTrain$Transported=="True",1,0)

#dfTrain <- read.csv("C:\\Users\\erico\\Documents\\R\\Space\\train.csv")
#dfTest <- read.csv("C:\\Users\\erico\\Documents\\R\\Space\\test.csv")
```

Code

1. Data Exploration

A. Summary Statistics

The data consists of 8693 records and 14 variables (6 numeric nd 8 character). There are a number of missing values and what appear to be skewed distributions among the numeric variables.

```
summary(dfTrain)
```

```
##  PassengerId         HomePlanet         CryoSleep            Cabin
##  Length:8693        Length:8693        Length:8693        Length:8693
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##  Destination             Age           VIP              RoomService
##  Length:8693        Min.   : 0.00   Length:8693        Min.   :    0.0
##  Class :character   1st Qu.:19.00   Class :character   1st Qu.:    0.0
##  Mode  :character   Median :27.00   Mode  :character   Median :    0.0
##                     Mean   :28.83                      Mean   :  224.7
##                     3rd Qu.:38.00                      3rd Qu.:   47.0
##                     Max.   :79.00                      Max.   :14327.0
##                     NA's   :179                        NA's   :181
##    FoodCourt        ShoppingMall          Spa               VRDeck
##  Min.   :    0.0   Min.   :    0.0   Min.   :    0.0   Min.   :    0.0
##  1st Qu.:    0.0   1st Qu.:    0.0   1st Qu.:    0.0   1st Qu.:    0.0
##  Median :    0.0   Median :    0.0   Median :    0.0   Median :    0.0
##  Mean   :  458.1   Mean   :  173.7   Mean   :  311.1   Mean   :  304.9
##  3rd Qu.:   76.0   3rd Qu.:   27.0   3rd Qu.:   59.0   3rd Qu.:   46.0
##  Max.   :29813.0   Max.   :23492.0   Max.   :22408.0   Max.   :24133.0
##  NA's   :183       NA's   :208       NA's   :183       NA's   :188
##      Name            Transported
##  Length:8693        Min.   :0.0000
##  Class :character   1st Qu.:0.0000
##  Mode  :character   Median :1.0000
##                     Mean   :0.5036
##                     3rd Qu.:1.0000
##                     Max.   :1.0000
##
```

```
str(dfTrain)
```

```
## 'data.frame':    8693 obs. of  14 variables:
##  $ PassengerId : chr  "0001_01" "0002_01" "0003_01" "0003_02" ...
##  $ HomePlanet  : chr  "Europa" "Earth" "Europa" "Europa" ...
##  $ CryoSleep   : chr  "False" "False" "False" "False" ...
##  $ Cabin       : chr  "B/0/P" "F/0/S" "A/0/S" "A/0/S" ...
##  $ Destination : chr  "TRAPPIST-1e" "TRAPPIST-1e" "TRAPPIST-1e" "TRAPPIST-1e" ...
```

```
##  $ Age         : num  39 24 58 33 16 44 26 28 35 14 ...
##  $ VIP         : chr  "False" "False" "True" "False" ...
##  $ RoomService : num  0 109 43 0 303 0 42 0 0 0 ...
##  $ FoodCourt   : num  0 9 3576 1283 70 ...
##  $ ShoppingMall: num  0 25 0 371 151 0 3 0 17 0 ...
##  $ Spa         : num  0 549 6715 3329 565 ...
##  $ VRDeck      : num  0 44 49 193 2 0 0 NA 0 0 ...
##  $ Name        : chr  "Maham Ofracculy" "Juanna Vines" "Altark Susent" "Solam Susent" ...
##  $ Transported : num  0 1 0 0 1 1 1 1 1 1 ...
```

```
dfTrain %>%
  count(dfTrain$Transported)
```

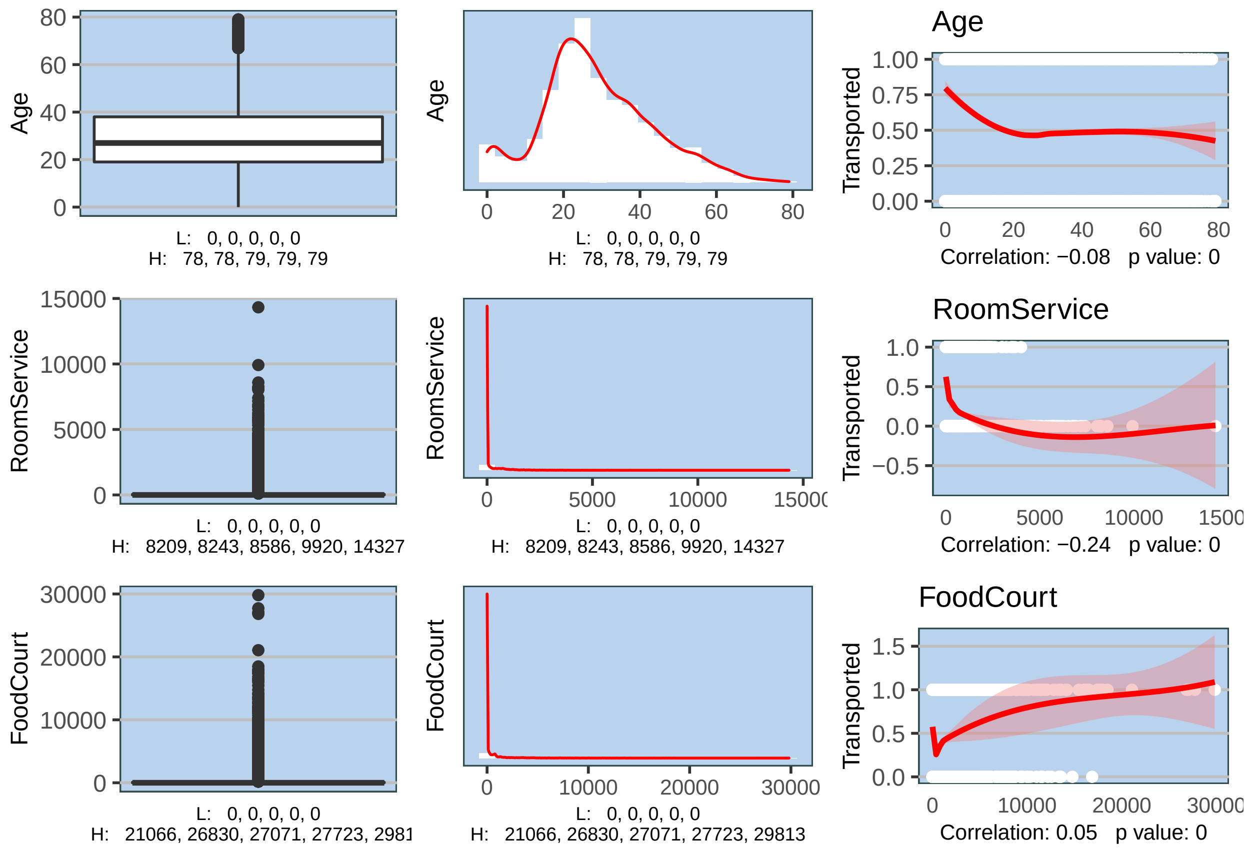
```
##   dfTrain$Transported    n
## 1                   0 4315
## 2                   1 4378
```

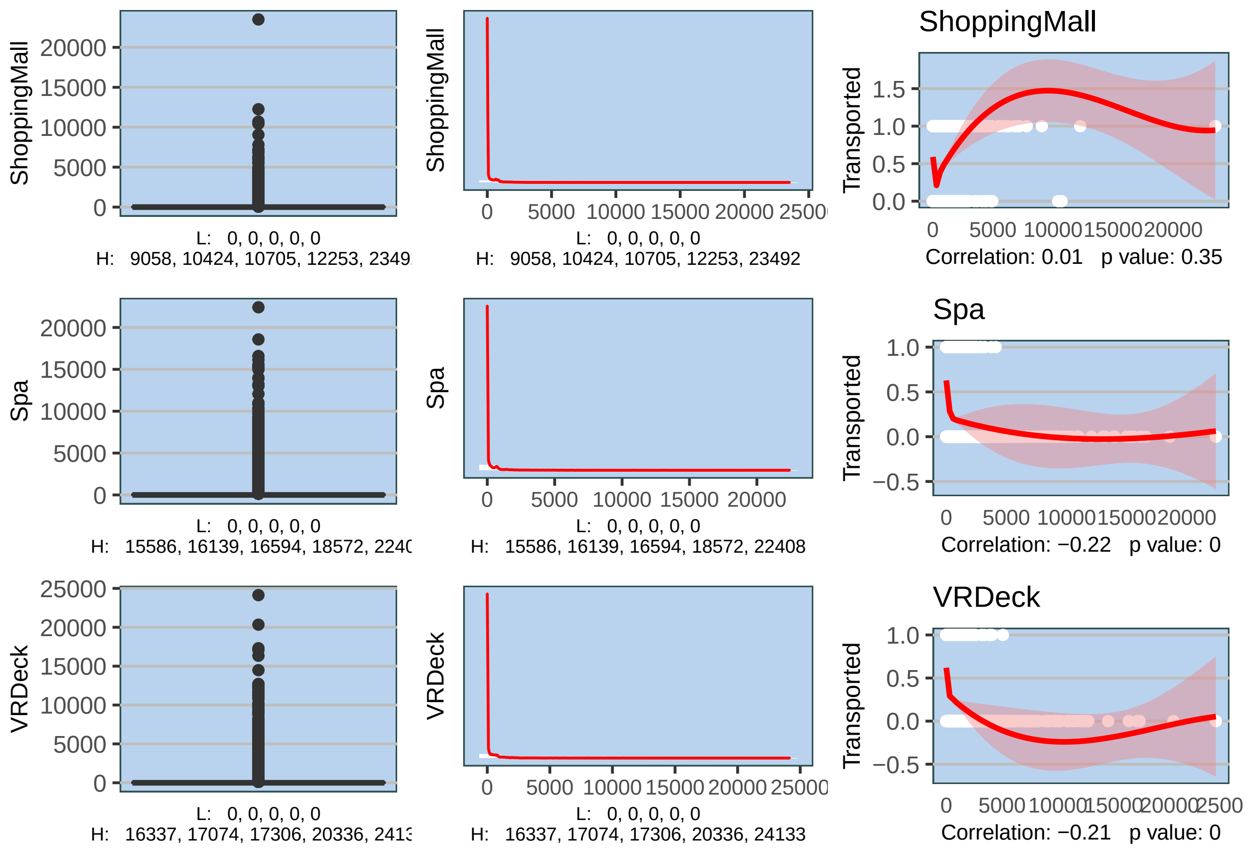
B. Distributions: Skewedness and Outliers

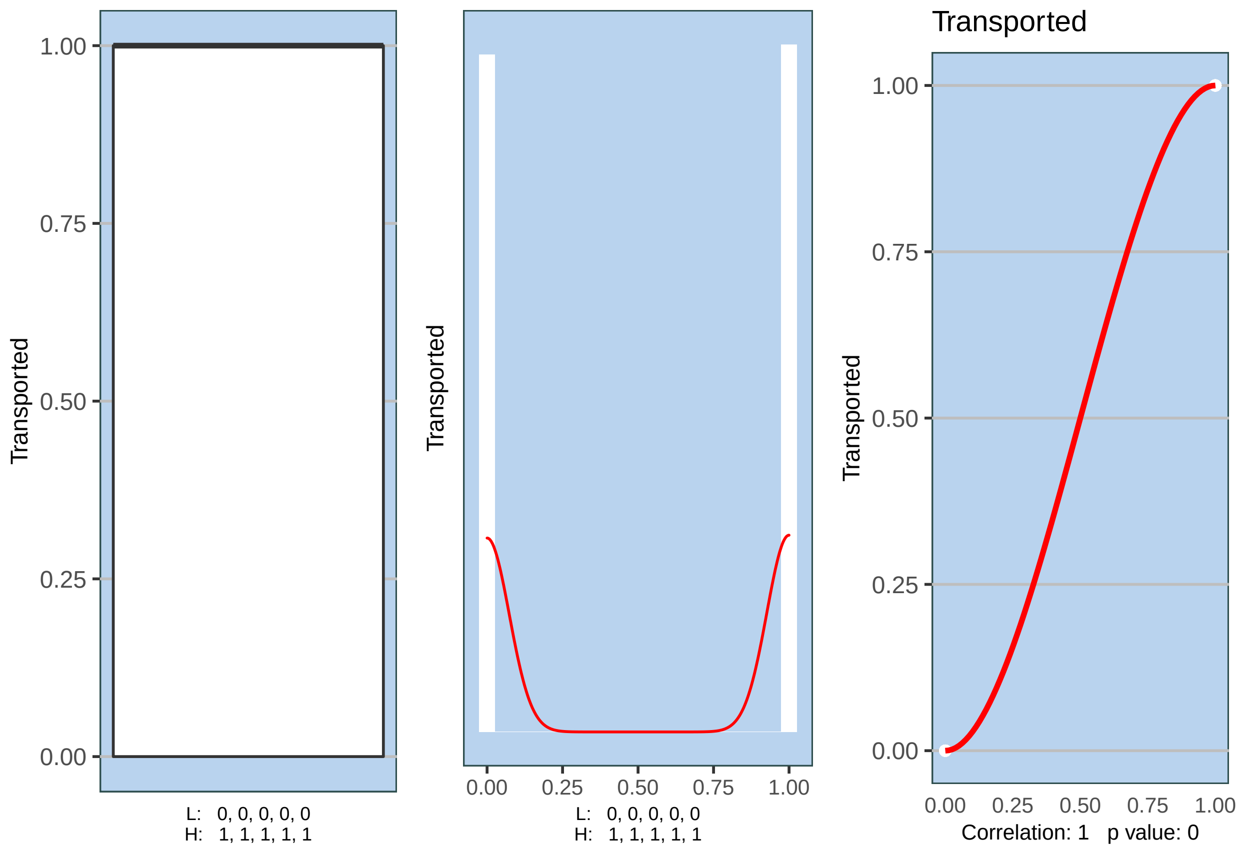
We look at the distribution of all numeric variables. Spending variables are highly skewed - most passengers spend no money while a few spend a great deal. We can see that spending on luxuries (the spa, room service, etc.) is strongly negatively correlated with being transported - this supports the supposition that the rich were spared. Spending on more popular amenities like the food court and shopping mall are also negatively correlated but less so. Age has a small negative correlation as well.

We decide not to log transform the numeric variables as normal distributions for predictors are not required by our models and interpretability suffers.

```
dfTrainD <- dfTrain

EHSummarize_StandardPlots(dfTrainD, "Transported")
```


Age
L: 0, 0, 0, 0, 0
H: 78, 78, 79, 79, 79
L: 0, 0, 0, 0, 0
H: 78, 78, 79, 79, 79
Transported
Correlation: −0.08 p value: 0
RoomService
L: 0, 0, 0, 0, 0
H: 8209, 8243, 8586, 9920, 14327
L: 0, 0, 0, 0, 0
H: 8209, 8243, 8586, 9920, 14327
Correlation: −0.24 p value: 0
FoodCourt
L: 0, 0, 0, 0, 0
H: 21066, 26830, 27071, 27723, 2981
L: 0, 0, 0, 0, 0
H: 21066, 26830, 27071, 27723, 29813
Correlation: 0.05 p value: 0

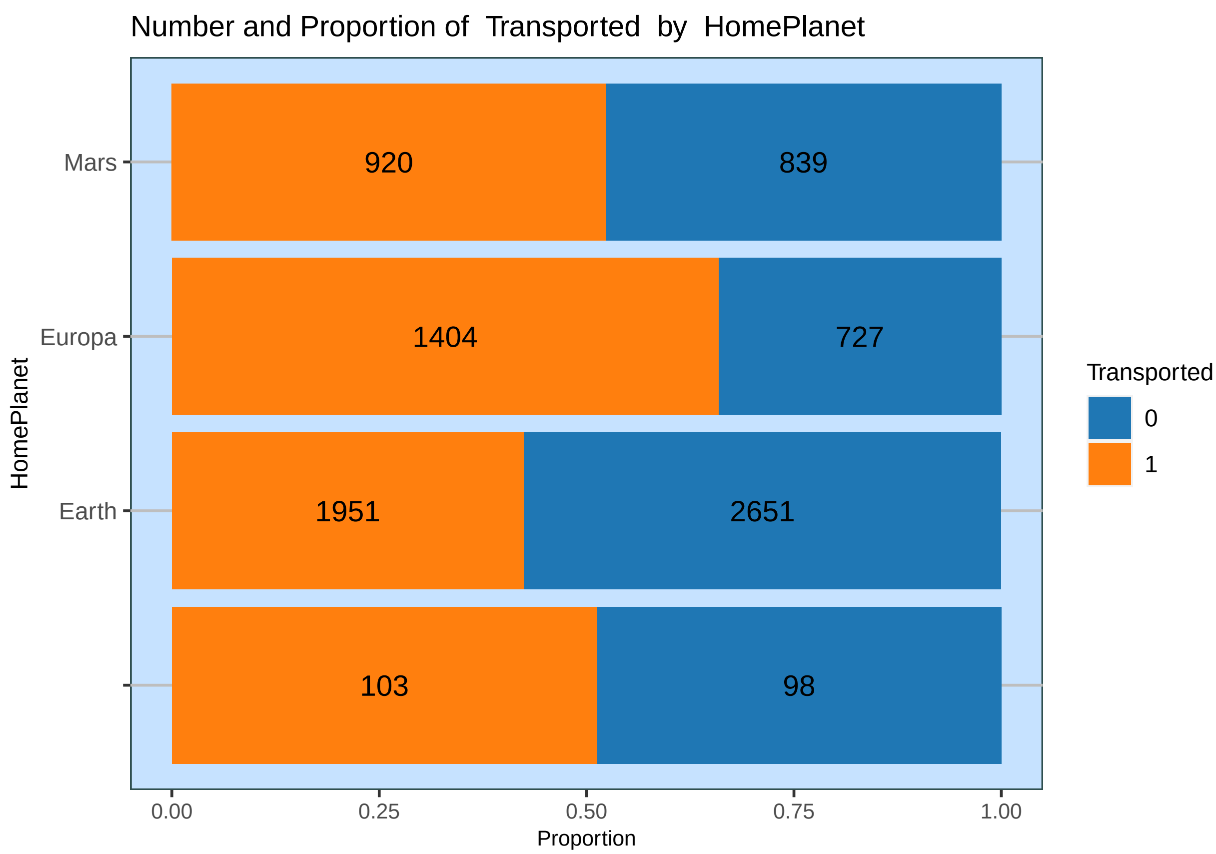

ShoppingMall
Transported
L: 0, 0, 0, 0, 0
H: 9058, 10424, 10705, 12253, 2349
H: 9058, 10424, 10705, 12253, 23492
Correlation: 0.01 p value: 0.35
Spa
L: 0, 0, 0, 0, 0
H: 15586, 16139, 16594, 18572, 2240
H: 15586, 16139, 16594, 18572, 22408
Correlation: −0.22 p value: 0
VRDeck
L: 0, 0, 0, 0, 0
H: 16337, 17074, 17306, 20336, 2413
H: 16337, 17074, 17306, 20336, 24133
Correlation: −0.21 p value: 0



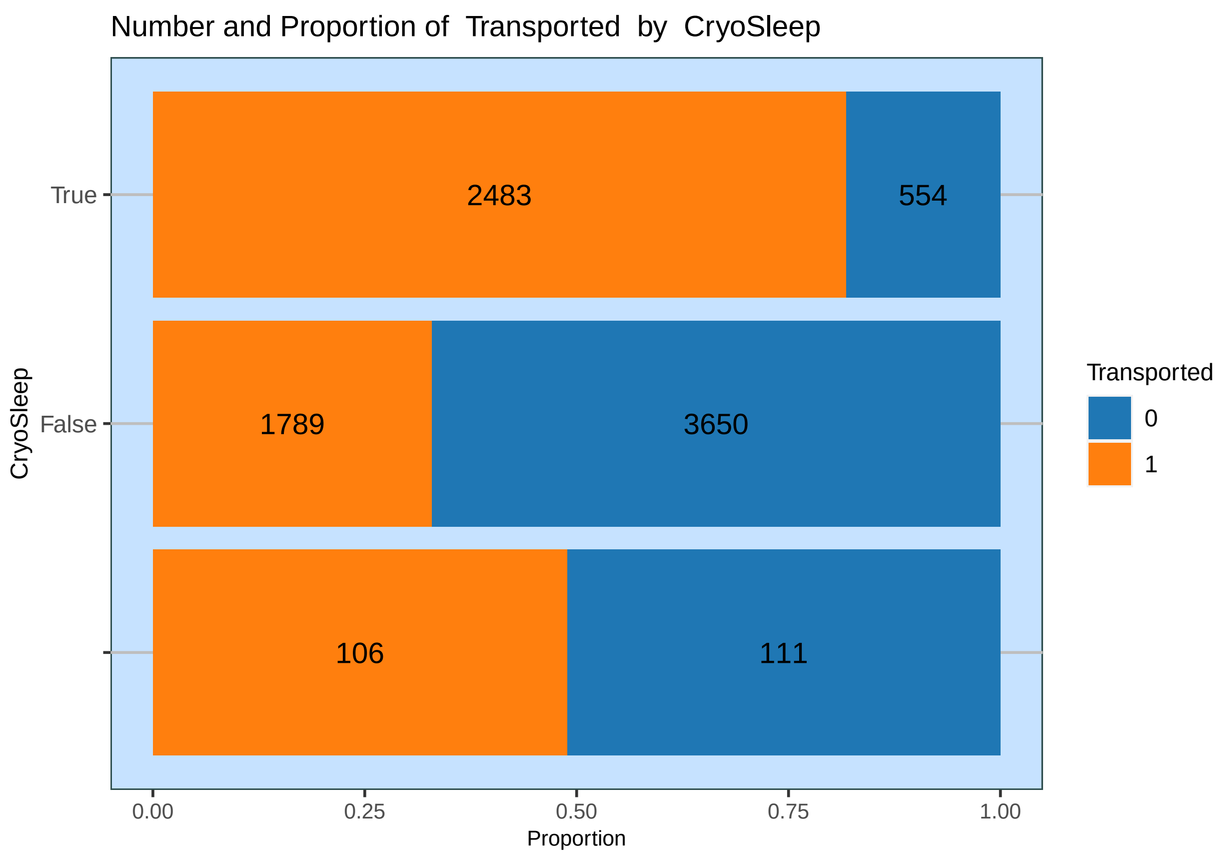
Here we look at count plots for character variables. Home and destination have an association with transported, but cryoSleep is especially important - over 75% of those in cryosleep (presumably a low budget option as food, drink and space are limited) were transported.

```
dfCount <- dfTrainD %>%
  dplyr::select(HomePlanet, CryoSleep, VIP, Destination, Transported)

dfCount$Transported <- as.character(dfCount$Transported)
EHExplore_TwoCategoricalColumns_Barcharts(dfCount, "Transported")
```

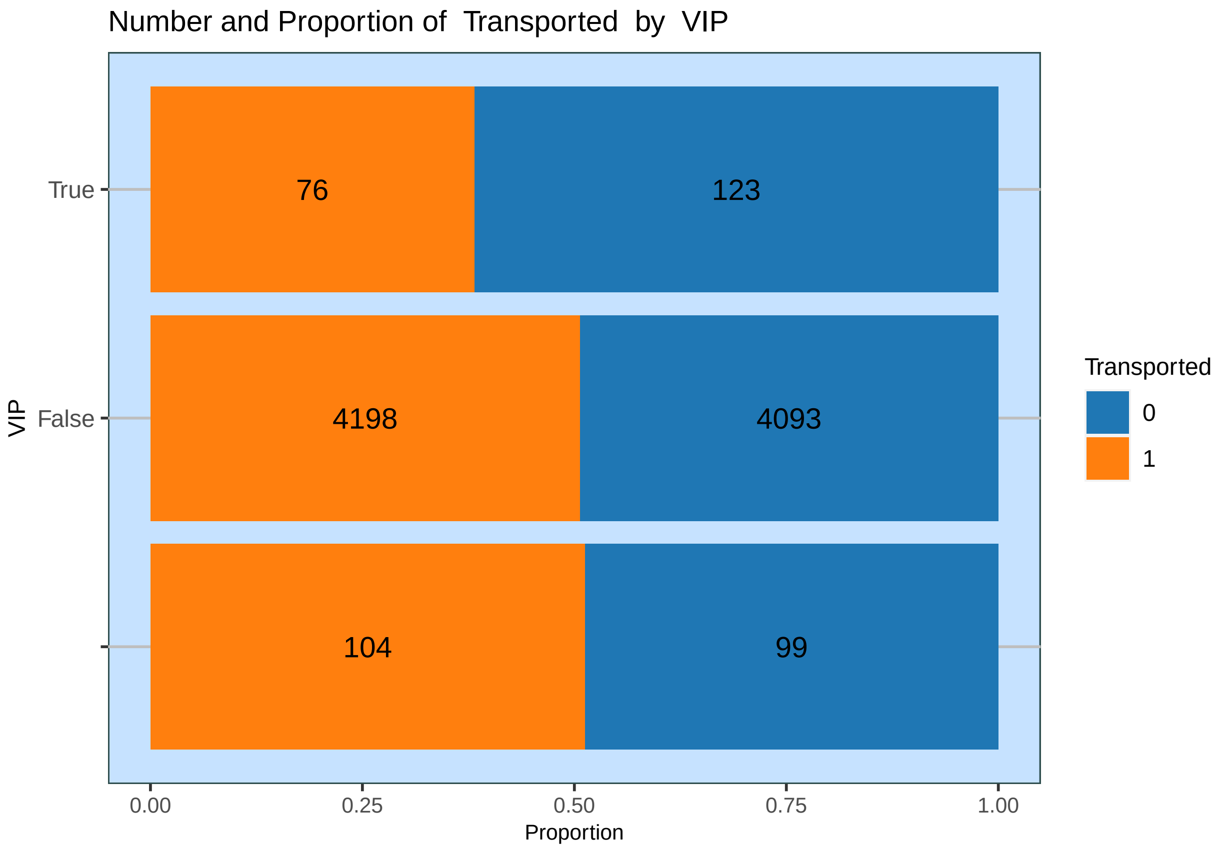
```
## [[1]]
```


Number and Proportion of Transported by HomePlanet
Mars
Europa
Earth
920
839
1404
727
1951
2651
103
98
HomePlanet
0.00
0.25
0.50
0.75
1.00
Proportion
Transported
0
1

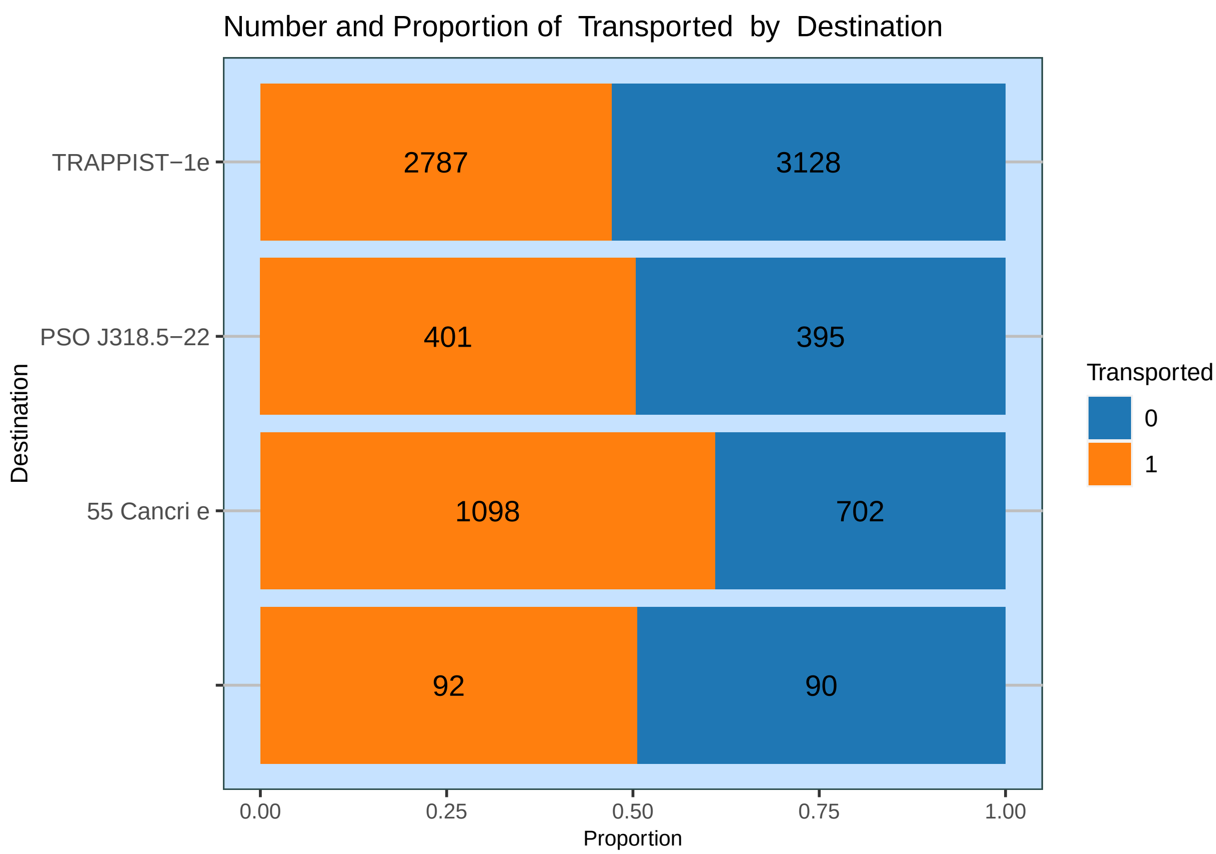
```
##
## [[2]]
```


Number and Proportion of Transported by CryoSleep
True
False
CryoSleep
2483
554
1789
3650
106
111
0.00
0.25
0.50
0.75
1.00
Proportion
Transported
0
1

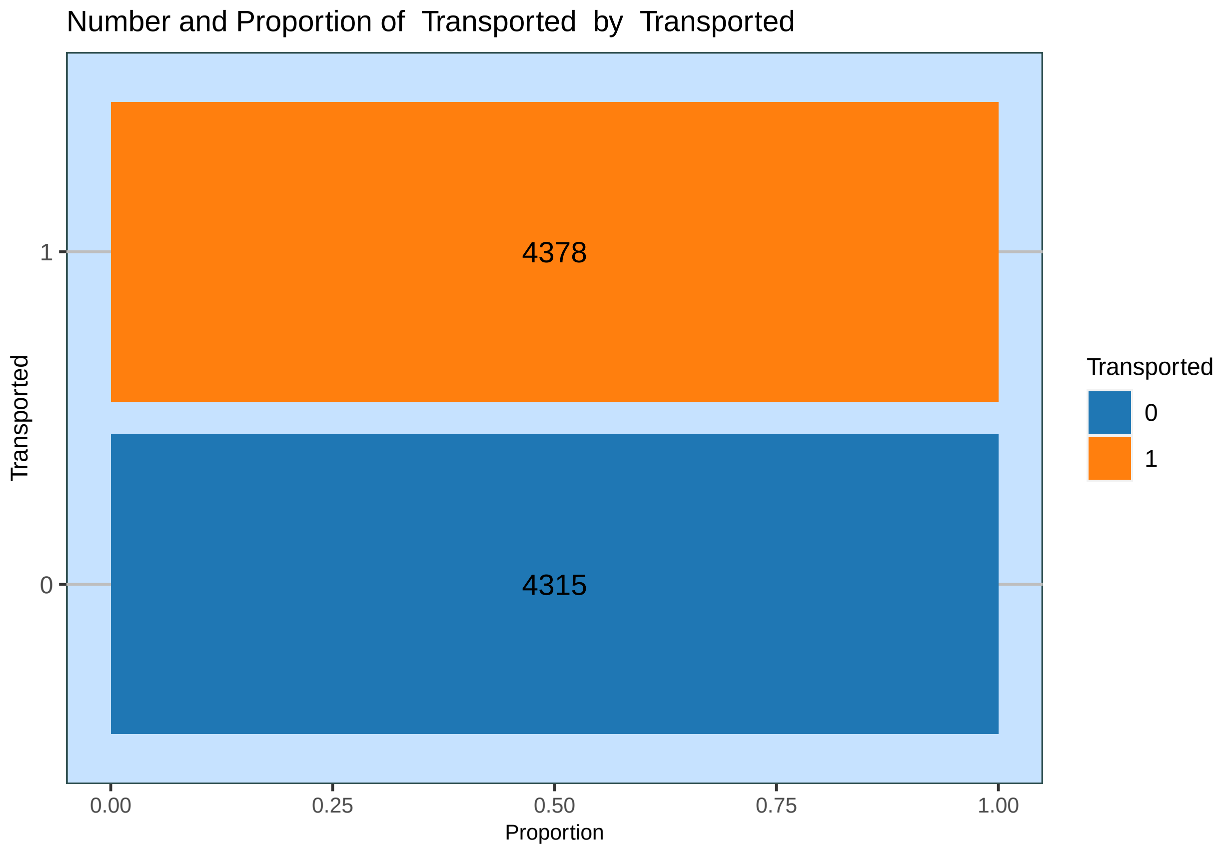
```
##
## [[3]]
```


Number and Proportion of Transported by VIP
True
False
VIP
76
123
4198
4093
104
99
0.00
0.25
0.50
0.75
1.00
Proportion
Transported
0
1

```
##
## [[4]]
```


Number and Proportion of Transported by Destination
TRAPPIST−1e
PSO J318.5−22
55 Cancri e
Destination
2787
3128
401
395
1098
702
92
90
0.00
0.25
0.50
0.75
1.00
Proportion
Transported
0
1

```
##
## [[5]]
```

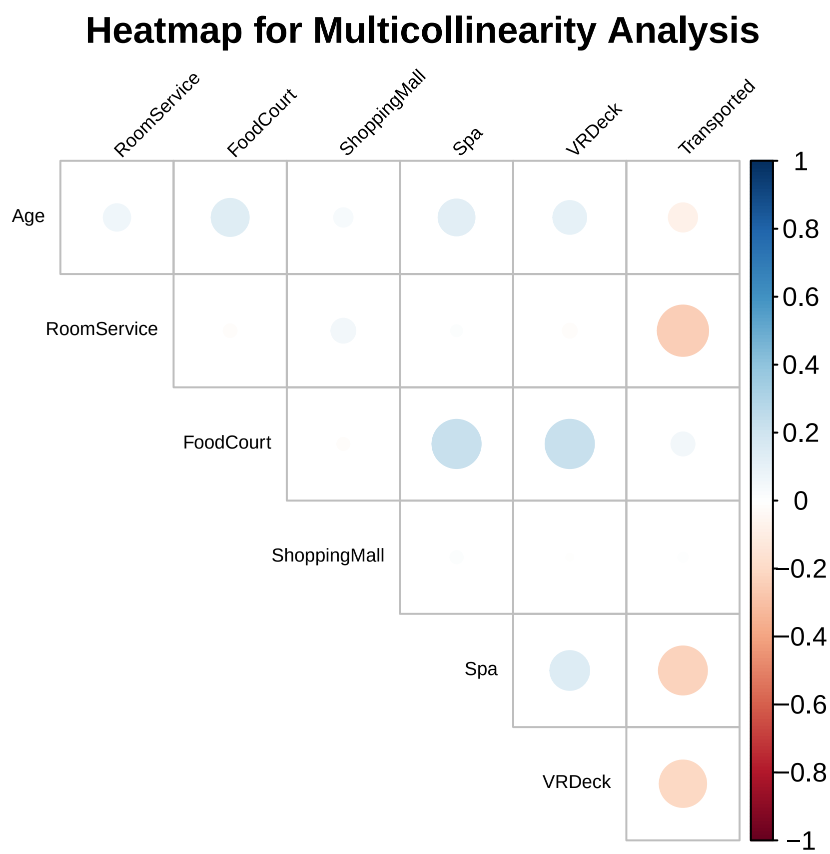


Because we have no reason to think that outliers are errors in data entry or significant data anomalies, and because our algorithms are relatively resistant to outliers, we do not remove outliers from the dataset.

C. Multicollinearity

While we were aware of the correlations with Transported, it is interesting to note that the correlations among various forms of spending are actually quite mild. We do not need to address multicollinearity in any systematic way.

```
dfNum <- dfTrain %>%
  dplyr::select(where(is.numeric))
dfNum <- na.omit(dfNum)

a <- EHExplore_Multicollinearity(dfNum)
```

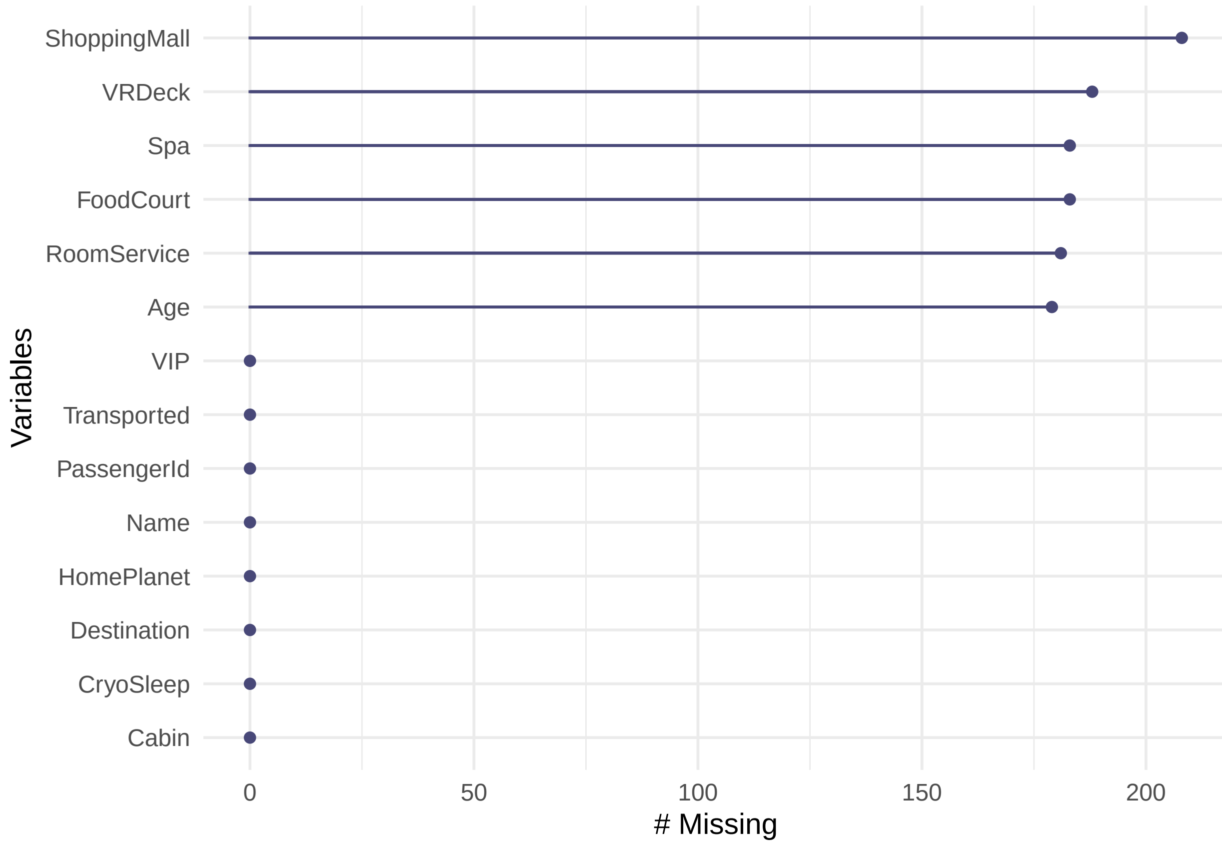


D. Missing Values

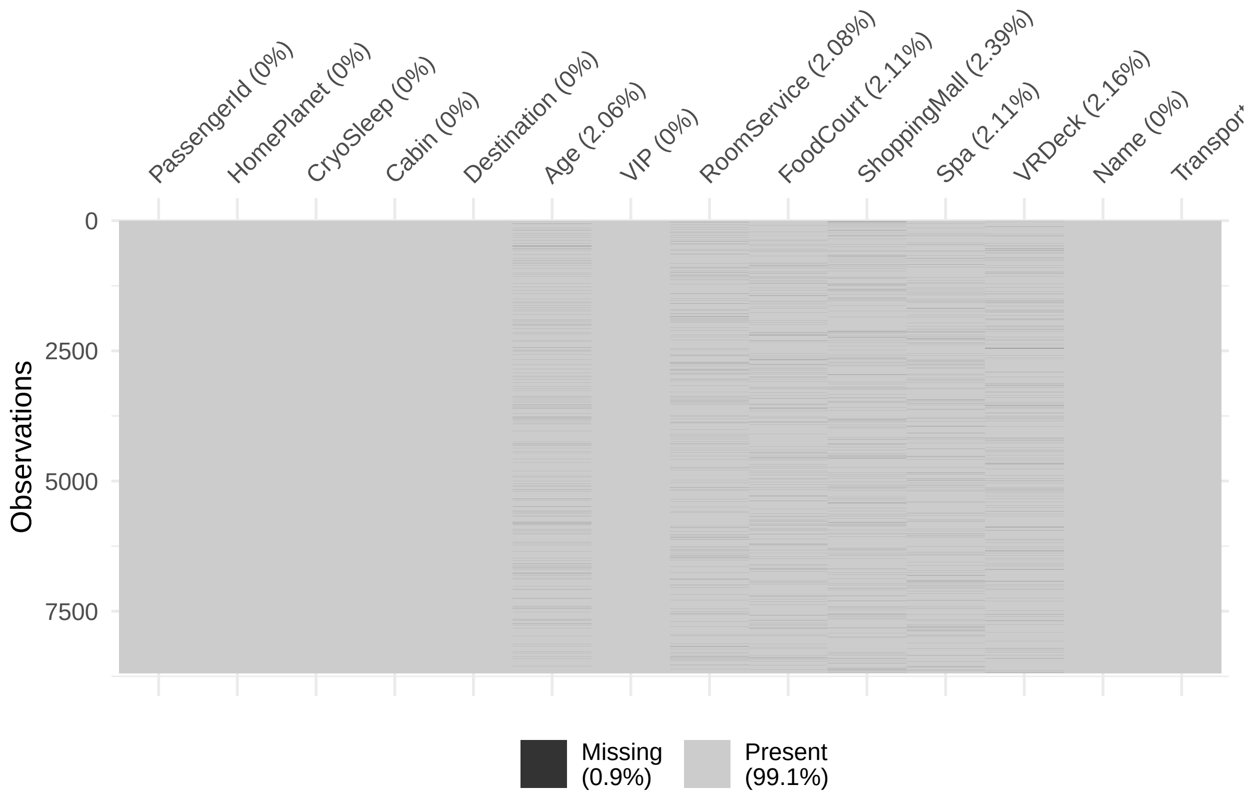
Missing values mainly appear for the amenities spending variables in the dataset. There are over 1,000 (12% of the database).

```
EHSummarize_MissingValues(dfTrain)
```

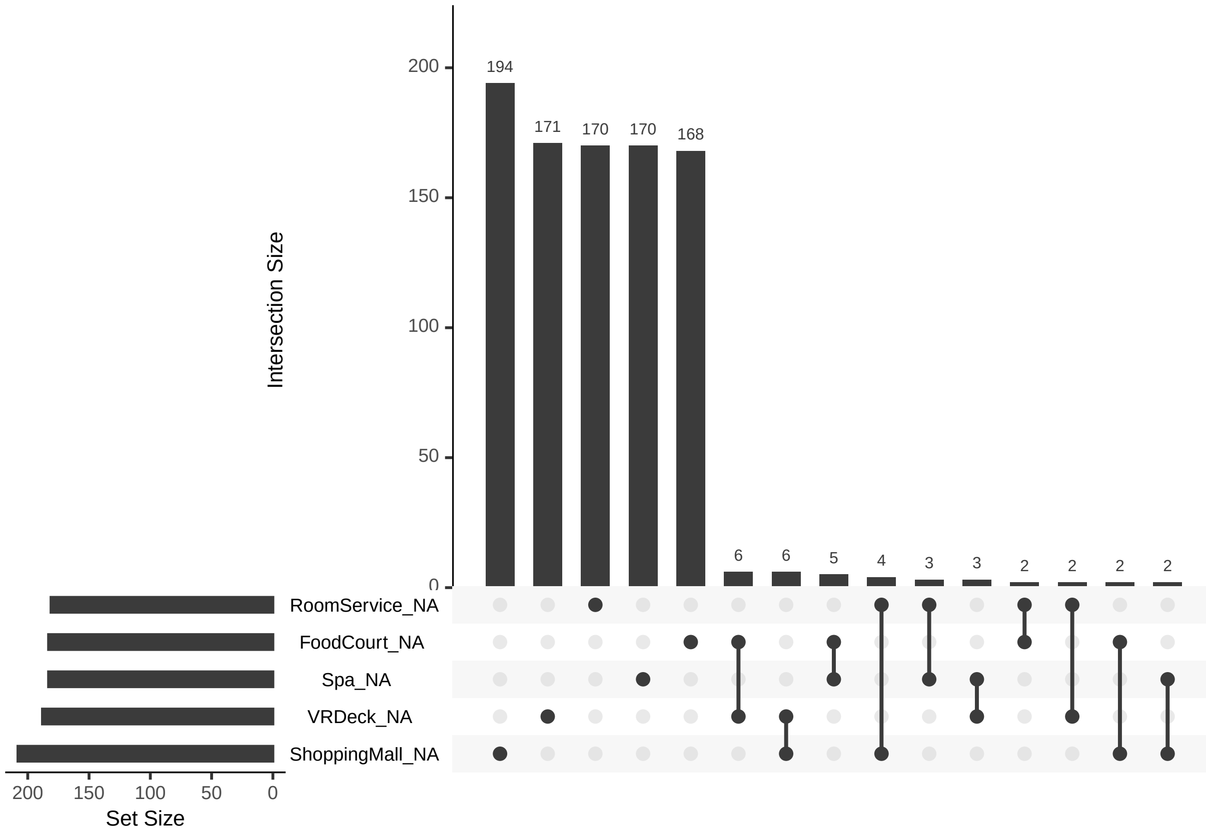
```
## [[1]]
```


ShoppingMall
VRDeck
Spa
FoodCourt
RoomService
Age
VIP
Transported
PassengerId
Name
HomePlanet
Destination
CryoSleep
Cabin
Variables
0
50
100
150
200
Missing

```
##
## [[2]]
```


PassengerId (0%)
HomePlanet (0%)
CryoSleep (0%)
Cabin (0%)
Destination (0%)
Age (2.06%)
VIP (0%)
RoomService (2.08%)
FoodCourt (2.11%)
ShoppingMall (2.39%)
Spa (2.11%)
VRDeck (2.16%)
Name (0%)
Observations
0
2500
5000
7500
Missing (0.9%)
Present (99.1%)

```
##
## [[3]]
```



The missing values are not correlated with each other, suggesting they are probably missing at random. To further support this hypothesis we create flags for missing values and perform Chi Square tests against the target variable. None of the flags are significant. We will therefore remove the records with missing values from the training set (we will do this after some feature engineering), and impute the median for the test set.

```
dfMissingRecordsFlagAny <- dfTrain %>%
    mutate(Flag=ifelse(rowSums(is.na(dfTrain)) > 0, 1, 0)) %>%
    dplyr::select(Transported, Flag)

dfMissingRecordsFlag_RoomService <- dfTrain %>%
    mutate(RoomServiceFlag=ifelse(is.na(dfTrain$RoomService) > 0, 1, 0)) %>%
      dplyr::select(Transported, RoomServiceFlag)


dfMissingRecordsFlag_VRDeck <- dfTrain %>%
    mutate(VRDeckFlag=ifelse(is.na(dfTrain$VRDeck) > 0, 1, 0)) %>%
      dplyr::select(Transported, VRDeckFlag)


dfMissingRecordsFlag_SPA <- dfTrain %>%
    mutate(SpaFlag=ifelse(is.na(dfTrain$Spa) > 0, 1, 0)) %>%
      dplyr::select(Transported, SpaFlag)


dfMissingRecordsFlag_ShoppingMall <- dfTrain %>%
```

```
    mutate(ShoppingMallFlag=ifelse(is.na(dfTrain$ShoppingMall) > 0, 1, 0)) %>%
      dplyr::select(Transported, ShoppingMallFlag)


dfMissingRecordsFlag_FoodCourt <- dfTrain %>%
    mutate(FoodCourtFlag=ifelse(is.na(dfTrain$FoodCourt) > 0, 1, 0)) %>%
      dplyr::select(Transported, FoodCourtFlag)

print(chisq.test(table(dfMissingRecordsFlagAny)))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(dfMissingRecordsFlagAny)
## X-squared = 0.15887, df = 1, p-value = 0.6902
```

```
print(chisq.test(table(dfMissingRecordsFlag_SPA)))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(dfMissingRecordsFlag_SPA)
## X-squared = 0.0098187, df = 1, p-value = 0.9211
```

```
print(chisq.test(table(dfMissingRecordsFlag_FoodCourt)))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(dfMissingRecordsFlag_FoodCourt)
## X-squared = 0.89665, df = 1, p-value = 0.3437
```

```
print(chisq.test(table(dfMissingRecordsFlag_VRDeck)))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(dfMissingRecordsFlag_VRDeck)
## X-squared = 0.1728, df = 1, p-value = 0.6776
```

```
print(chisq.test(table(dfMissingRecordsFlag_ShoppingMall)))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(dfMissingRecordsFlag_ShoppingMall)
## X-squared = 1.5072, df = 1, p-value = 0.2196
```

```
print(chisq.test(table(dfMissingRecordsFlag_RoomService)))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(dfMissingRecordsFlag_RoomService)
## X-squared = 1.3229, df = 1, p-value = 0.2501
```

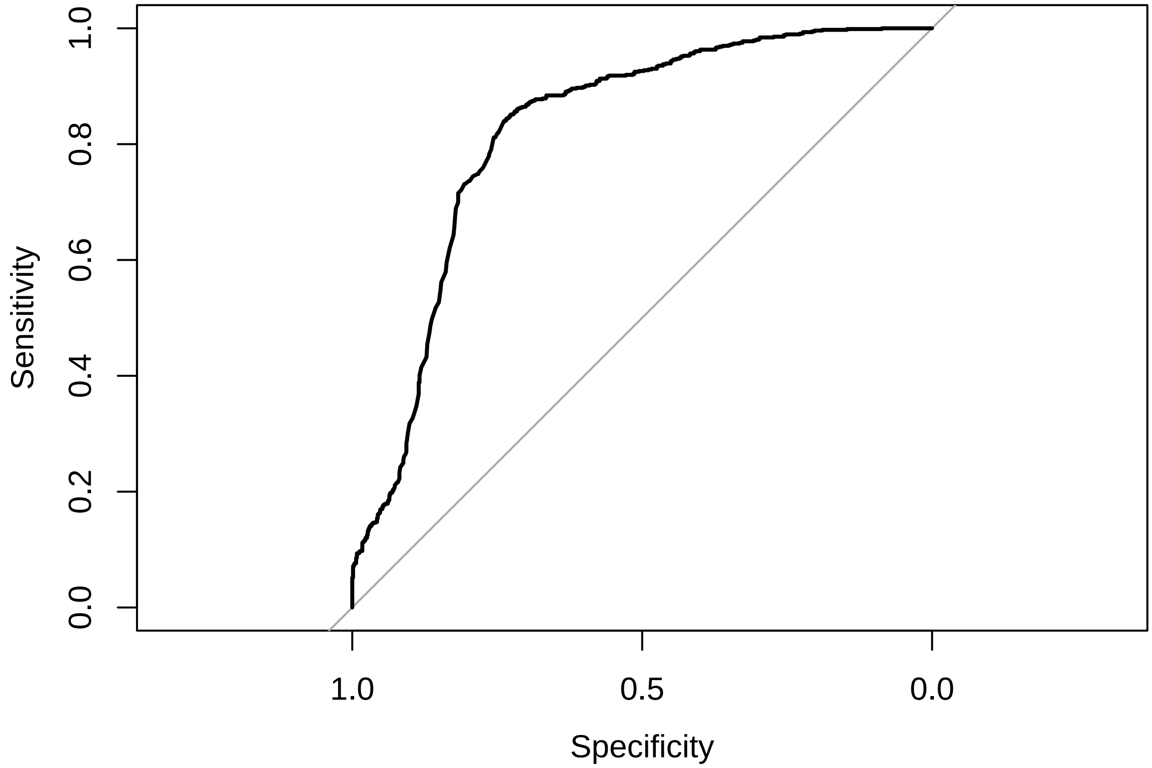
E. First Pass Logistic Regression: 9th percentile

We perform a logistic regression with what we have and post to Kaggle just to get a baseline. Accuracy on training is 77%, significantly better than the 51% no information rate, but gives us only 69% on the Kaggle set (which puts us at the 9th percentile).

```
dfTrainNum <- dfTrain %>%
  dplyr::select(where(is.numeric))

dfTestNum <- dfTest %>%
  dplyr::select(where(is.numeric))

dfTestNum <- cbind(dfTestNum, "PassengerID" = dfTest$PassengerId)

q <- EHModel_Regression_Logistic(dfTrainNum, "Transported", returnLM = TRUE)
```

```
##
## Call:
## glm(formula = fla, family = "binomial", data = train_reg)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4474  -0.8358   0.0166   0.8739   4.9443
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   7.125e-01  6.607e-02  10.785  < 2e-16 ***
## Age           2.085e-03  2.123e-03   0.982  0.32593
## RoomService  -2.216e-03  1.108e-04 -20.008  < 2e-16 ***
## FoodCourt     7.701e-04  4.622e-05  16.662  < 2e-16 ***
## ShoppingMall  1.940e-04  6.630e-05   2.926  0.00343 **
## Spa          -2.488e-03  1.342e-04 -18.536  < 2e-16 ***
## VRDeck       -2.078e-03  1.165e-04 -17.837  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8471.5  on 6110  degrees of freedom
## Residual deviance: 6288.6  on 6104  degrees of freedom
##   (844 observations deleted due to missingness)
## AIC: 6302.6
##
## Number of Fisher Scoring iterations: 7
```

```
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 499  89
##          1 251 670
##
##                Accuracy : 0.7747
##                  95% CI : (0.7528, 0.7955)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5488
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6653
##             Specificity : 0.8827
##          Pos Pred Value : 0.8486
##          Neg Pred Value : 0.7275
##              Prevalence : 0.4970
##          Detection Rate : 0.3307
##    Detection Prevalence : 0.3897
##       Balanced Accuracy : 0.7740
##
##        'Positive' Class : 0
##
```




```
##
## Call:
## roc.default(response = dfPred_raw$class, predictor = dfPred_raw$predict_reg,     plot = TRUE)
##
## Data: dfPred_raw$predict_reg in 750 controls (dfPred_raw$class 0) < 759 cases (dfPred_raw$class 1).
## Area under the curve: 0.822
```

```
predictions_logistic <- EHModel_Predict(q, dfTestNum, threshold=.5, testData_IDColumn = "PassengerID", p

predictions_logistic$Transported <- ifelse(predictions_logistic$Transported==1,"True","False")
write_csv(predictions_logistic, "C://Users//erico//Desktop//LR_FirstPass.csv")
```

Data Preparation and Feature Engineering

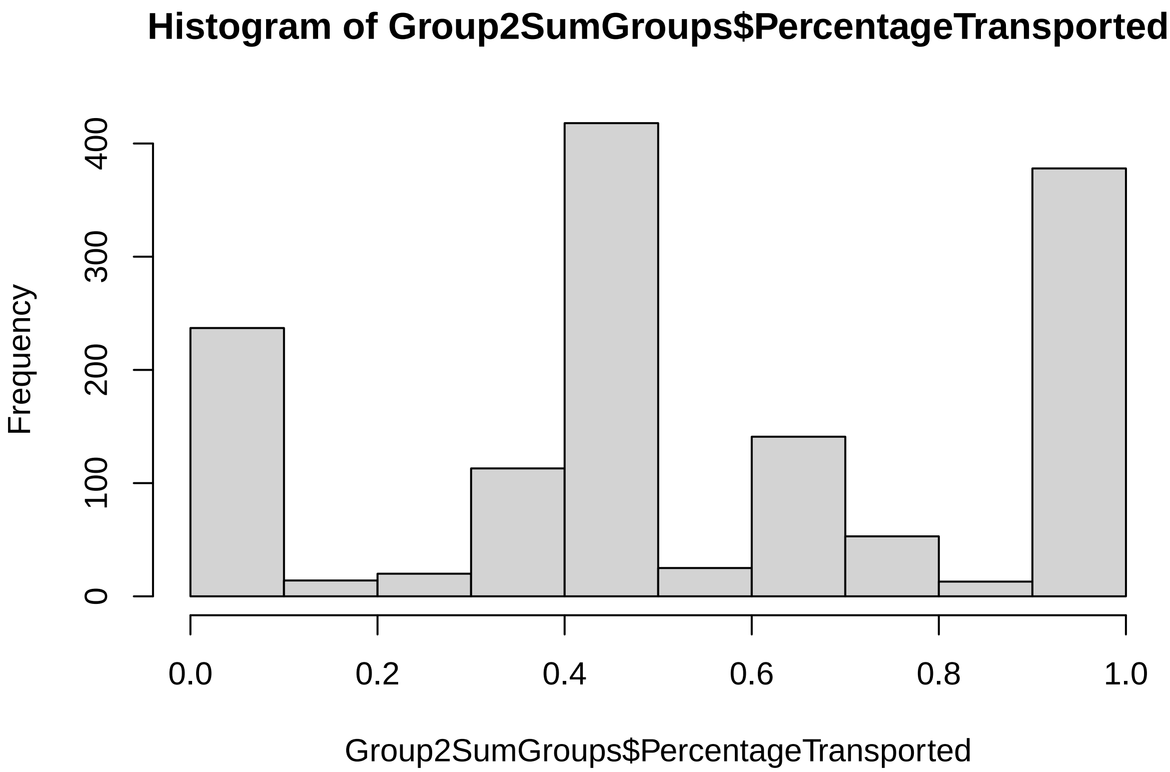
1. Create groups based on the Passenger ID

Passenger IDs are constructed to identify passengers travelling in groups. We create groupings from the ID.

How likely is it that if the majority of members of a group were transported, then they all were transported? Only somewhat likely. A histogram shows the distribution of percentages of transported within groups. Most often, half the members transported and half did not.

```
Group2 <- dfTrain %>% dplyr::select(PassengerId, Transported)
```

```
Group2$Group <- sub("\\_.*", "", Group2$PassengerId)

Group2Sum <- Group2 %>%
  dplyr::group_by(Group) %>%
  dplyr::summarize(PercentageTransported= sum(Transported)/dplyr::n(), count=dplyr::n())

Group2SumGroups <- Group2Sum %>%
  filter(count>1)

hist(Group2SumGroups$PercentageTransported)
```




```
dfTrainA <- dfTrain

dfTrainA$Group <- Group2$Group

dfTrainAA <- inner_join(dfTrainA, Group2Sum, by="Group") %>% dplyr::select(-Group, -PercentageTransporte

dfTrainBB <- dfTrainAA

#boxplot(as.numeric(dfTrainBB$count), as.numeric(dfTrainBB$Transported))

Group2a <- dfTest %>% dplyr::select(PassengerId)

Group2a$Group <- sub("\\_.*", "", Group2a$PassengerId)
```

```
Group2aSum <- Group2a %>%
  dplyr::group_by(Group) %>%
  dplyr::summarize(count=dplyr::n())

dfTestA <- dfTest

dfTestA$Group <- Group2a$Group

dfTestAA <- inner_join(dfTestA, Group2aSum, by="Group") %>% dplyr::select(-Group)

dfTrainAA$InAGroup = ifelse(dfTrainAA$count>1,1,0)
dfTestAA$InAGroup = ifelse(dfTestAA$count>1,1,0)

dfTrain1 <- na.omit(dfTrainAA)
dfTest1 <- EHPrepare_MissingValues_Imputation(dfTestAA, impute="median")
```

2. Create Cabin Variables

Cabin variables consist of 3 parts in the form of a/b/c which indicate the location of the cabin on the ship. Here we extract out parts "a" and "c" - b appears to have no influence on the target.

```
library(stringr)
dfTrain1$Cabin1 <- substr(dfTrain1$Cabin,1,1)
dfTrain1$Cabin2 <- str_sub(dfTrain1$Cabin, - 1, - 1)

dfTrain2 <- dfTrain1 %>%
  dplyr::select(-Cabin)

dfTest1$Cabin1 <- substr(dfTest1$Cabin,1,1)
dfTest1$Cabin2 <- str_sub(dfTest1$Cabin, - 1, - 1)

dfTest2 <- dfTest1 %>%
  dplyr::select(-Cabin)

dfTrain3 <- dfTrain2
dfTest3 <- dfTest2
```

3. Create Dummy Variables

Now that we have engineered Cabin, we create dummy variables to handle category variables throughout the dataset.

```
dfTrain4 <- dfTrain3 %>%
  dplyr::select(-Name, -PassengerId)

PassengerID <- dfTest3$PassengerId
dfTest4 <- dfTest3 %>%
  dplyr::select(-Name, -PassengerId)

dfTrain5v <- EHPrepare_CreateDummies(dfTrain4, "Transported")
dfTrain5 <- dfTrain5v %>%
  dplyr::select(-Cabin2_)
```

```r
dfTest5v <- EHPrepare_CreateDummies(dfTest4, "Transported")
dfTest5 <- dfTest5v %>%
  dplyr::select(-Cabin2_)

dfTest5 <- cbind(PassengerID, dfTest5)
```

4. Implement Interaction Features

There were a number of interactions among variables which appeared when the variables were examined in isolation. We only retained one (group passengers were more likely to be transported if they shopped at the mall) because the others did not significantly increase accuracy when running the overall model - but there would be more to explore here.

```r
dfTrain5$Inter_CountShop = dfTrain5$InAGroup*dfTrain5$ShoppingMall
dfTest5$Inter_CountShop = dfTest5$InAGroup*dfTest5$ShoppingMall
```

5. Perform Logistic Regression With Engineered Features: 26th Percentile

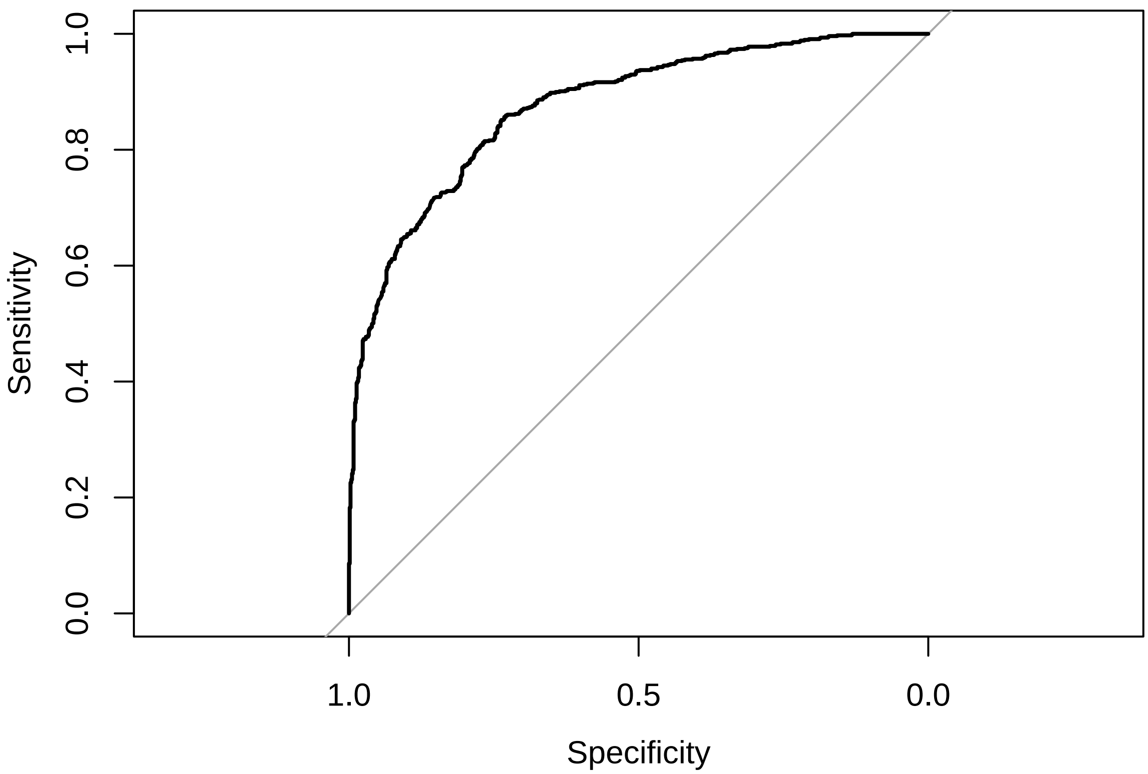
We perform a logistic regression with what we have and post to Kaggle just to get a baseline. Accuracy on the holdout set improves training to 79% (compared to 77% on the untransformed regression), and gives us 78% on the Kaggle set, which puts us at the 26th percentile.

```r
dfTrain5z <- dfTrain5

dfTrain5zz <- dfTrain5z %>%
  dplyr::select(-count)

dfTest5z <- dfTest5

dfTest5zz <- dfTest5z %>%
  dplyr::select(-count)

set.seed(042758)

q <- EHModel_Regression_Logistic(dfTrain5zz, "Transported", returnLM = TRUE)
```

```
##
## Call:
## glm(formula = fla, family = "binomial", data = train_reg)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9252  -0.6568   0.0149   0.6997   3.3536
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)               0.2955844  0.1180833   2.503 0.012308 *
## Age                      -0.0079509  0.0025348  -3.137 0.001709 **
## RoomService              -0.0015488  0.0001113 -13.920  < 2e-16 ***
## FoodCourt                 0.0006140  0.0000530  11.585  < 2e-16 ***
## ShoppingMall              0.0006873  0.0001026   6.701 2.07e-11 ***
```

```
## Spa                        -0.0020068  0.0001222 -16.427  < 2e-16 ***
## VRDeck                     -0.0019359  0.0001217 -15.910  < 2e-16 ***
## InAGroup                    0.1753772  0.0802973   2.184 0.028955 *
## HomePlanet_                 0.4533562  0.2195945   2.065 0.038969 *
## HomePlanet_Europa           1.7502630  0.2681931   6.526 6.75e-11 ***
## HomePlanet_Mars             0.5704315  0.1141614   4.997 5.83e-07 ***
## CryoSleep_                  0.2728606  0.2061154   1.324 0.185561
## CryoSleep_True              1.3798768  0.0978384  14.104  < 2e-16 ***
## Destination_                0.4335945  0.2398617   1.808 0.070655 .
## Destination_55.Cancri.e     0.4794132  0.0980725   4.888 1.02e-06 ***
## Destination_PSO.J318.5.22  0.0326575  0.1102523   0.296 0.767072
## VIP_                        0.2137313  0.2274328   0.940 0.347343
## VIP_True                   -0.1144653  0.3008491  -0.380 0.703593
## Cabin1_                    -0.4472731  0.2413786  -1.853 0.063883 .
## Cabin1_A                   -0.9518680  0.3502360  -2.718 0.006572 **
## Cabin1_B                    0.2368897  0.3155204   0.751 0.452779
## Cabin1_C                    1.3388313  0.3520418   3.803 0.000143 ***
## Cabin1_D                   -0.1615527  0.2077058  -0.778 0.436689
## Cabin1_E                   -0.6671090  0.1240673  -5.377 7.57e-08 ***
## Cabin1_G                   -0.4013448  0.0994084  -4.037 5.41e-05 ***
## Cabin1_T                   -1.3950991  1.8765514  -0.743 0.457217
## Cabin2_P                   -0.6597095  0.0704040  -9.370  < 2e-16 ***
## Inter_CountShop            -0.0003796  0.0001440  -2.637 0.008368 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8450.7  on 6095  degrees of freedom
## Residual deviance: 5172.8  on 6068  degrees of freedom
## AIC: 5228.8
##
## Number of Fisher Scoring iterations: 7
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 580 143
##          1 177 624
##
##                Accuracy : 0.79
##                  95% CI : (0.7687, 0.8102)
##     No Information Rate : 0.5033
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5799
##
##  Mcnemar's Test P-Value : 0.06507
##
##             Sensitivity : 0.7662
##             Specificity : 0.8136
##          Pos Pred Value : 0.8022
##          Neg Pred Value : 0.7790
```

```
##                Prevalence : 0.4967
##            Detection Rate : 0.3806
##      Detection Prevalence : 0.4744
##         Balanced Accuracy : 0.7899
##
##          'Positive' Class : 0
##
```




```
##
## Call:
## roc.default(response = dfPred_raw$class, predictor = dfPred_raw$predict_reg,     plot = TRUE)
##
## Data: dfPred_raw$predict_reg in 757 controls (dfPred_raw$class 0) < 767 cases (dfPred_raw$class 1).
## Area under the curve: 0.8751
```

```
predictions_logistic <- EHModel_Predict(q, dfTest5zz, threshold=.5, testData_IDColumn = "PassengerID", 

predictions_logistic$Transported <- ifelse(predictions_logistic$Transported==1,"True","False")
write_csv(predictions_logistic, "C://Users//erico//Desktop//LR_WithFeatures.csv")
```

At this point, a picture of the transported passengers begins to emerge - they are the poorer passengers, most likely to shop on a budget or, even cheaper, spend the trip in cryosleep. They tend to enter the ship in groups and inhabit lower class cabins.

More Complex Models

Given the apparent complexity of the data shape, we turn to more complex nonparametric models to improve our predictions.

1. Perform SVM: 70th Percentile

We begin with Support Vector Machines and try three kernels - linear, poly and radial. We perform ten-fold cross validation and optimal hypertuning of C based on the caret package. Radial performs the best (accuracy=80.1% on the kaggle set) and boosts us to the 70th percentile.

```
library(doParallel)
library(caret)

cl<-makePSOCKcluster(7)

registerDoParallel(cl)

print("Linear: --------------------------------------")
```

```
## [1] "Linear: --------------------------------------"
```

```
lin <- EHModel_SVM(dfTrain5zz, "Transported", method="linear")
```

```
## Support Vector Machines with Linear Kernel
##
## 6097 samples
##   27 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (27), scaled (27)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 5487, 5487, 5487, 5488, 5488, 5488, ...
## Resampling results across tuning parameters:
##
##   C          Accuracy   Kappa
##   0.0100000  0.7760621  0.5524086
##   0.1147368  0.7987524  0.5974004
##   0.2194737  0.8002288  0.6003060
##   0.3242105  0.8000653  0.5999630
##   0.4289474  0.8004484  0.6007181
##   0.5336842  0.8001749  0.6001643
##   0.6384211  0.8001748  0.6001610
##   0.7431579  0.8002845  0.6003763
##   0.8478947  0.8003938  0.6005952
##   0.9526316  0.8004481  0.6007033
##   1.0573684  0.8001202  0.6000454
##   1.1621053  0.8004484  0.6007013
##   1.2668421  0.8002297  0.6002636
##   1.3715789  0.8002845  0.6003721
##   1.4763158  0.8002845  0.6003719
##   1.5810526  0.8002844  0.6003714
```

```
##   1.6857895  0.8002845  0.6003698
##   1.7905263  0.8005032  0.6008077
##   1.8952632  0.8003390  0.6004790
##   2.0000000  0.8003937  0.6005881
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.790526.
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 562 133
##          1 195 633
##
##                Accuracy : 0.7846
##                  95% CI : (0.7631, 0.805)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.569
##
##  Mcnemar's Test P-Value : 0.0007567
##
##             Sensitivity : 0.7424
##             Specificity : 0.8264
##          Pos Pred Value : 0.8086
##          Neg Pred Value : 0.7645
##              Prevalence : 0.4970
##          Detection Rate : 0.3690
##    Detection Prevalence : 0.4563
##       Balanced Accuracy : 0.7844
##
##        'Positive' Class : 0
##
```

```r
print("Poly: --------------------------------------")
```

```
## [1] "Poly: --------------------------------------"
```

```r
poly <- EHModel_SVM(dfTrain5zz, "Transported", method="poly")
```

```
## Support Vector Machines with Polynomial Kernel
##
## 6097 samples
##   27 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (27), scaled (27)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 5487, 5487, 5487, 5488, 5488, 5488, ...
## Resampling results across tuning parameters:
##
##   degree  scale  C     Accuracy   Kappa
```

```
##   1       0.001  0.25  0.7551225  0.5108272
##   1       0.001  0.50  0.7498187  0.5002867
##   1       0.001  1.00  0.7554498  0.5114740
##   1       0.010  0.25  0.7633219  0.5271118
##   1       0.010  0.50  0.7703213  0.5410180
##   1       0.010  1.00  0.7759529  0.5521904
##   1       0.100  0.25  0.7869427  0.5740405
##   1       0.100  0.50  0.7943245  0.5886851
##   1       0.100  1.00  0.7984247  0.5967629
##   2       0.001  0.25  0.7497093  0.5000669
##   2       0.001  0.50  0.7552857  0.5111456
##   2       0.001  1.00  0.7615722  0.5236397
##   2       0.010  0.25  0.7717418  0.5438533
##   2       0.010  0.50  0.7814206  0.5631022
##   2       0.010  1.00  0.7927376  0.5855954
##   2       0.100  0.25  0.8023051  0.6045642
##   2       0.100  0.50  0.8050391  0.6099812
##   2       0.100  1.00  0.8044376  0.6087612
##   3       0.001  0.25  0.7527161  0.5060429
##   3       0.001  0.50  0.7588931  0.5183145
##   3       0.001  1.00  0.7650716  0.5305831
##   3       0.010  0.25  0.7802723  0.5608270
##   3       0.010  0.50  0.7912077  0.5825486
##   3       0.010  1.00  0.7976053  0.5952331
##   3       0.100  0.25  0.7935057  0.5869583
##   3       0.100  0.50  0.7950370  0.5900214
##   3       0.100  1.00  0.7952015  0.5903619
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 2, scale = 0.1 and C = 0.5.
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 570 148
##          1 187 618
##
##                Accuracy : 0.78
##                  95% CI : (0.7584, 0.8006)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5599
##
##  Mcnemar's Test P-Value : 0.03788
##
##             Sensitivity : 0.7530
##             Specificity : 0.8068
##          Pos Pred Value : 0.7939
##          Neg Pred Value : 0.7677
##              Prevalence : 0.4970
##          Detection Rate : 0.3743
##    Detection Prevalence : 0.4714
##       Balanced Accuracy : 0.7799
```

```
##
##        'Positive' Class : 0
##
```

```r
print("Radial: --------------------------------------")
```

```
## [1] "Radial: --------------------------------------"
```

```r
rad <- EHModel_SVM(dfTrain5zz, "Transported", method="radial")
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 6097 samples
##   27 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (27), scaled (27)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 5487, 5487, 5487, 5488, 5488, 5488, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7841545  0.5685024
##   0.50  0.7899499  0.5799809
##   1.00  0.7951430  0.5902743
##
## Tuning parameter 'sigma' was held constant at a value of 0.04367297
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.04367297 and C = 1.
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 574 165
##          1 183 601
##
##                Accuracy : 0.7715
##                  95% CI : (0.7496, 0.7924)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5429
##
##  Mcnemar's Test P-Value : 0.3621
##
##             Sensitivity : 0.7583
##             Specificity : 0.7846
##          Pos Pred Value : 0.7767
##          Neg Pred Value : 0.7666
##              Prevalence : 0.4970
##          Detection Rate : 0.3769
##    Detection Prevalence : 0.4852
##       Balanced Accuracy : 0.7714
```

```
##
##        'Positive' Class : 0
##
```

```
predictions_rad <- EHModel_Predict(rad$svm, dfTest5zz, predictionsColumnName = "Transported", testData_I

predictions_rad$Transported <- ifelse(predictions_rad$Transported==1,"True","False")
write_csv(predictions_rad, "C://Users//erico//Desktop//SVM_Rad.csv")


stopCluster(cl)
```

2. Perform limited Neural Networks

Neural Networks require a great deal of computer resources and time. A simple first pass with two hidden layers took a great deal of time, needed a high stepmax to converge and provided poor results (15th percentile). The algorithm was therefore difficult to hypertune.

```
library(neuralnet)
set.seed(42760)

dfTrainN <- dfTrain5zz
dfTrainN$Transported = as.factor(dfTrainN$Transported)
dfScale <- EHPrepare_ScaleAllButTarget(dfTrainN, "Transported")

#n <- neuralnet(Transported ~ .,
               #data = dfScale,
              # hidden = 2,
               #err.fct = "ce",
              # linear.output = F,
              # lifesign = 'full',
              # rep = 2,
             #  algorithm = "rprop+",
             #  stepmax = 100000, likelihood=TRUE, )
```

```
#plot(n, rep = 2)
```

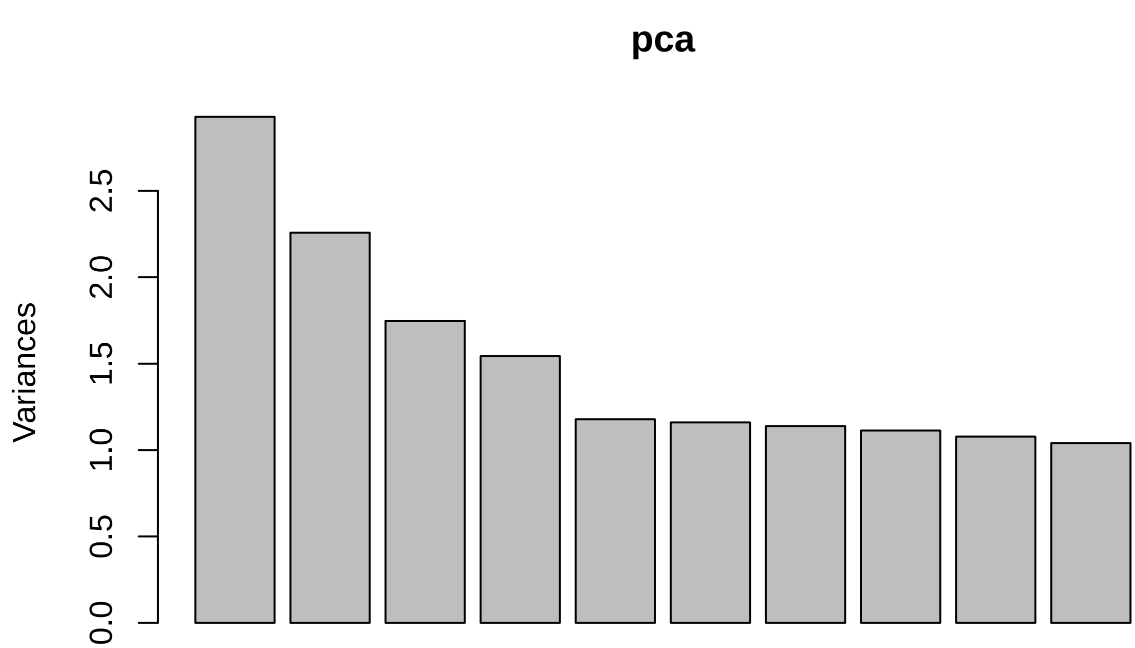
```
#predicts <- predict(n, dfTest5zz)
#predict_nn <- as.data.frame(predict(n, dfTest5zz)) %>%
#  dplyr::select(V1) %>%
#  dplyr::rename("Transported" = V1) %>%
#  mutate(Transported=as.numeric(Transported)) %>%
#  mutate(Transported=ifelse(Transported>.5,"True","False"))

#predictions_nn <- EHModel_Predict(n, dfTest5zz, predictionsColumnName = "Transported", #testData_IDCol

#write_csv(predictions_nn, "C://Users//erico//Desktop//nn.csv")
```

In order to address the long time until convergence (many hours), we experimented with dimensionality reduction, but this was ineffective. PCA, e.g., did not result in a small number of components taking the largest share of variance. Taking a sample of records ormanually eliminating columns allowed for faster run times but hurt performance. Below is the result of PCA analysis:

```r
pca <- prcomp(dfTrain5zz, center = TRUE, scale. = TRUE)

summary(pca)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     1.7113 1.50279 1.32219 1.24229 1.08522 1.07691 1.06709
## Proportion of Variance 0.1046 0.08066 0.06243 0.05512 0.04206 0.04142 0.04067
## Cumulative Proportion  0.1046 0.18524 0.24768 0.30279 0.34486 0.38627 0.42694
##                            PC8    PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     1.05502 1.0382 1.02008 1.01195 1.00481 0.99789 0.98678
## Proportion of Variance 0.03975 0.0385 0.03716 0.03657 0.03606 0.03556 0.03478
## Cumulative Proportion  0.46669 0.5052 0.54236 0.57893 0.61499 0.65055 0.68533
##                           PC15    PC16    PC17   PC18    PC19    PC20    PC21
## Standard deviation     0.98295 0.93825 0.93370 0.9150 0.90835 0.88785 0.87515
## Proportion of Variance 0.03451 0.03144 0.03114 0.0299 0.02947 0.02815 0.02735
## Cumulative Proportion  0.71983 0.75127 0.78241 0.8123 0.84178 0.86993 0.89729
##                           PC22   PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.81848 0.8111 0.76818 0.64984 0.50335 0.45711 0.27096
## Proportion of Variance 0.02393 0.0235 0.02108 0.01508 0.00905 0.00746 0.00262
## Cumulative Proportion  0.92121 0.9447 0.96578 0.98087 0.98992 0.99738 1.00000
```

```r
plot(pca)
```



3. Tree Algorithms 1: Perform Random Forest: 34th Percentile

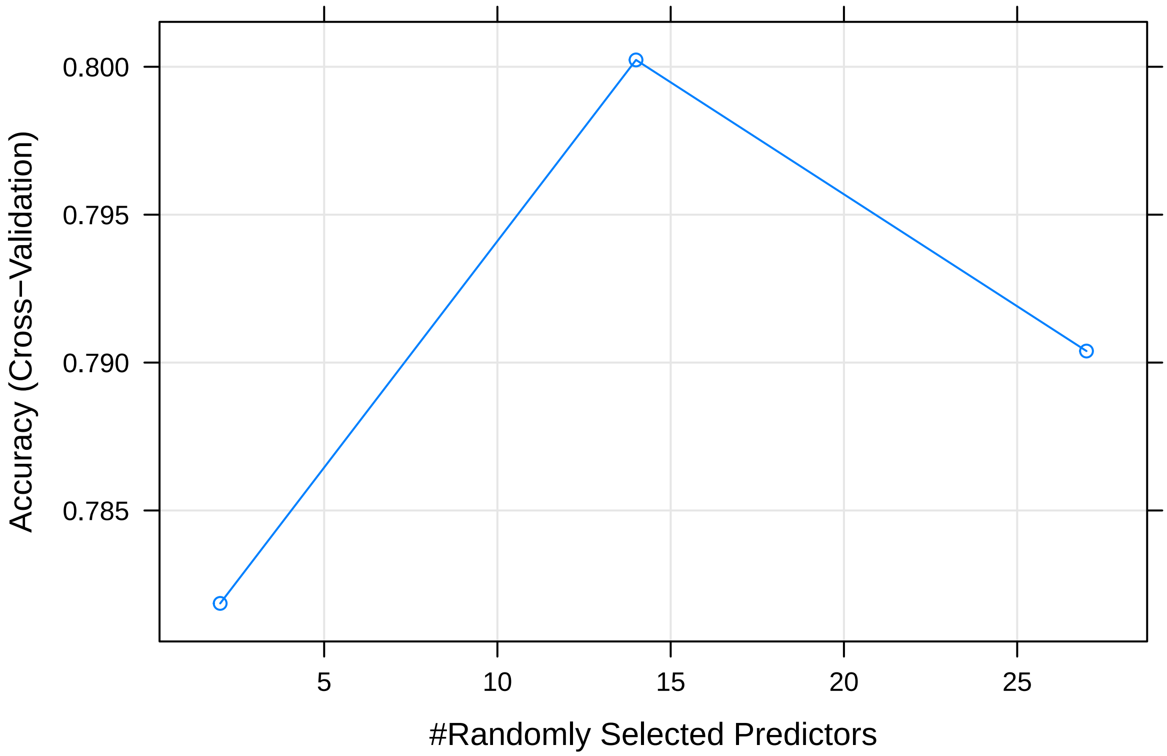
Random forest is our first tree. We use the parallel library to run the model on multiple cores.

We perform ten-fold cross validation and optimal hypertuning of sigma and c based on the caret package.

This gives us 79% on the holdout set and 78.8% on the kaggle set, which puts us in the 34th percentile. This is low compared to SVM.

```
library(doParallel)

cl<-makePSOCKcluster(7)

registerDoParallel(cl)


a <- EHModel_RandomForest(dfTrain5zz, "Transported")
```

```
## Random Forest
##
## 6097 samples
##   27 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5487, 5487, 5487, 5488, 5488, 5488, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7818587  0.5639198
##   14    0.8002320  0.6005062
##   27    0.7903895  0.5809306
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 14.
```




```
## rf variable importance
##
##   only 20 most important variables shown (out of 27)
##
##                             Overall
## CryoSleep_True              100.000
## Spa                          85.968
## Age                          85.931
## VRDeck                       80.377
## RoomService                  76.410
## FoodCourt                    67.359
## ShoppingMall                 45.336
## Cabin1_G                     20.058
## Cabin2_P                     15.122
## Cabin1_E                     13.814
## HomePlanet_Europa            12.745
## Inter_CountShop              10.717
## Destination_55.Cancri.e       9.720
## InAGroup                      9.616
## HomePlanet_Mars               8.855
## Destination_PSO.J318.5.22     8.058
## HomePlanet_                   4.233
## CryoSleep_                    3.845
## VIP_                          3.684
## Cabin1_C                      3.332
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   0   1
##          0 606 169
##          1 151 597
##
##                Accuracy : 0.7899
##                  95% CI : (0.7686, 0.8101)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5798
##
##  Mcnemar's Test P-Value : 0.3419
##
##             Sensitivity : 0.8005
##             Specificity : 0.7794
##          Pos Pred Value : 0.7819
##          Neg Pred Value : 0.7981
##              Prevalence : 0.4970
##          Detection Rate : 0.3979
##    Detection Prevalence : 0.5089
##       Balanced Accuracy : 0.7900
##
##        'Positive' Class : 0
##
## [1] "Parameters:   mtry =  14 , ntree =  500 , nrnodes =  1605"
```

```
predictions_rf <- EHModel_Predict(a$rf, dfTest5zz, predictionsColumnName = "Transported", testData_IDCol

predictions_rf$Transported <- ifelse(predictions_rf$Transported==1,"True","False")
write_csv(predictions_rf, "C://Users//erico//Desktop//RF1.csv")


stopCluster(cl)
```

4. Tree Algorithms 2: Perform XGBoost Untuned: 73rd Percentile

At the 34th percentile we need a more powerful model. As an active learner, XGBoost is likely to fit our training model better than random forest, though it may overfit the data.

Our untuned model, with 55 rounds (chosen randomly), achieves 79% accuracy on the holdout set and 80.2% accuracy on the kaggle set, and reaches the 74th percentile.

```
library(xgboost)
library(caret)

set.seed(0)

dfTrain5q <-dfTrain5
dfTest5q <- dfTest5

dfTrain5q$Transported = as.character(dfTrain5q$Transported)
```

```r
dfTrain5$Transported = as.character(dfTrain5$Transported)


train_x <- dfTrain5q %>%
  dplyr::select(-Transported, -count)
train_y <- dfTrain5q %>%
  dplyr::select(Transported)

dfTest6 <- dfTest5q
dfTest6$Transported = "1"

test_x <- dfTest6 %>%
  dplyr::select(-Transported,-PassengerID, -count)
test_y <- dfTest6 %>%
  dplyr::select(Transported)


xgb_train <- xgb.DMatrix(data = as.matrix(train_x), label = as.matrix(train_y))
xgb_test <- xgb.DMatrix(data = as.matrix(test_x), label = as.matrix(test_y))

#depth of 1000 and nrounds 100000 showed no double descent

model2 <- xgb.train(data = xgb_train, max.depth = 3, nrounds = 50)
```

```r
predictions <- predict(model2,newdata=xgb_test)
predictions <- data.frame(as.vector(predictions))
predictions$PassengerId <- dfTest5$PassengerID
predictions[,c(1,2)] <- predictions[,c(2,1)]
colnames(predictions) <- c("PassengerID", "Transported")

predictions <- predictions %>%
  mutate(Transported=ifelse(Transported>.5,'True','False'))

write_csv(predictions, "C:\\Users\\erico\\Desktop\\XGBpredictions_Rounds50.csv")
```

Hypertuning

Now we try some experiments with hypertuning. First we choose a more optimal nrounds for xgboost. Then we try manually adjusting sigma both up and down on our radial SVM model to see if that improves performance.

1. Tune XGBoost: 78th Percentile

We perform ten-fold cross validation on 1-100 nrounds and find the optimal nrounds for accuracy (14). Our model, with 14 rounds, achieves 80.3% accuracy and reaches the 74th percentile.

```r
set.seed(100)
cv <- xgb.cv(data = xgb_train, nrounds = 100, nthread = 2, nfold = 10, metrics = list("rmse","auc", "er
                  max_depth = 3, eta = 1, objective = "binary:logistic")
```

```
## [1]  train-rmse:0.415047+0.000841    train-auc:0.813023+0.001987 train-error:0.282094+0.001992   tes
```

```
## [2]  train-rmse:0.385845+0.001185    train-auc:0.860516+0.002116 train-error:0.222164+0.007927   tes
## [3]  train-rmse:0.378852+0.001398    train-auc:0.870728+0.001990 train-error:0.209274+0.003398   tes
## [4]  train-rmse:0.372729+0.001637    train-auc:0.880756+0.002273 train-error:0.203893+0.003009   tes
## [5]  train-rmse:0.368863+0.001861    train-auc:0.886316+0.002073 train-error:0.201575+0.002140   tes
## [6]  train-rmse:0.366441+0.001607    train-auc:0.889350+0.001927 train-error:0.199446+0.002347   tes
## [7]  train-rmse:0.364508+0.001586    train-auc:0.891987+0.001954 train-error:0.196544+0.003051   tes
## [8]  train-rmse:0.361741+0.002500    train-auc:0.895200+0.002572 train-error:0.192797+0.004358   tes
## [9]  train-rmse:0.359367+0.002674    train-auc:0.897881+0.002820 train-error:0.190580+0.005296   tes
## [10] train-rmse:0.357505+0.002598    train-auc:0.900123+0.002623 train-error:0.188655+0.005272   tes
## [11] train-rmse:0.356330+0.002516    train-auc:0.901504+0.002411 train-error:0.187474+0.005112   tes
## [12] train-rmse:0.354930+0.002560    train-auc:0.903052+0.002540 train-error:0.185681+0.004593   tes
## [13] train-rmse:0.353317+0.002560    train-auc:0.905031+0.002491 train-error:0.183129+0.003636   tes
## [14] train-rmse:0.352609+0.002423    train-auc:0.905840+0.002382 train-error:0.181846+0.003486   tes
## [15] train-rmse:0.351316+0.002204    train-auc:0.907134+0.002231 train-error:0.180782+0.003400   tes
## [16] train-rmse:0.350333+0.002146    train-auc:0.908162+0.002161 train-error:0.179280+0.002695   tes
## [17] train-rmse:0.349126+0.002166    train-auc:0.909447+0.002254 train-error:0.177924+0.003161   tes
## [18] train-rmse:0.348295+0.002153    train-auc:0.910399+0.002297 train-error:0.177705+0.002517   tes
## [19] train-rmse:0.347184+0.002109    train-auc:0.911515+0.002141 train-error:0.176363+0.002503   tes
## [20] train-rmse:0.346005+0.001498    train-auc:0.912769+0.001518 train-error:0.175168+0.002447   tes
## [21] train-rmse:0.345138+0.001769    train-auc:0.913644+0.001712 train-error:0.174161+0.003187   tes
## [22] train-rmse:0.344329+0.001754    train-auc:0.914464+0.001689 train-error:0.173637+0.003224   tes
## [23] train-rmse:0.343350+0.001915    train-auc:0.915459+0.001870 train-error:0.172907+0.003426   tes
## [24] train-rmse:0.342475+0.001964    train-auc:0.916324+0.001901 train-error:0.171945+0.003266   tes
## [25] train-rmse:0.341509+0.002072    train-auc:0.917216+0.002013 train-error:0.171289+0.003922   tes
## [26] train-rmse:0.340689+0.001933    train-auc:0.918042+0.001888 train-error:0.170516+0.003508   tes
## [27] train-rmse:0.339792+0.002083    train-auc:0.918946+0.002021 train-error:0.169714+0.004183   tes
## [28] train-rmse:0.339044+0.002041    train-auc:0.919682+0.001946 train-error:0.169175+0.003845   tes
## [29] train-rmse:0.338046+0.001954    train-auc:0.920647+0.001815 train-error:0.167760+0.003082   tes
## [30] train-rmse:0.337128+0.001795    train-auc:0.921598+0.001656 train-error:0.166900+0.002514   tes
## [31] train-rmse:0.336358+0.001755    train-auc:0.922325+0.001597 train-error:0.166229+0.002539   tes
## [32] train-rmse:0.335645+0.001621    train-auc:0.922911+0.001497 train-error:0.165150+0.002449   tes
## [33] train-rmse:0.334922+0.001529    train-auc:0.923596+0.001316 train-error:0.164071+0.002469   tes
## [34] train-rmse:0.333932+0.001624    train-auc:0.924532+0.001408 train-error:0.163459+0.002936   tes
## [35] train-rmse:0.332912+0.001554    train-auc:0.925443+0.001347 train-error:0.162321+0.002618   tes
## [36] train-rmse:0.332262+0.001438    train-auc:0.926035+0.001194 train-error:0.161738+0.002974   tes
## [37] train-rmse:0.331246+0.001364    train-auc:0.926959+0.001162 train-error:0.161534+0.002805   tes
## [38] train-rmse:0.330345+0.001437    train-auc:0.927820+0.001221 train-error:0.159740+0.003067   tes
## [39] train-rmse:0.329495+0.001612    train-auc:0.928612+0.001336 train-error:0.158953+0.003136   tes
## [40] train-rmse:0.328807+0.001581    train-auc:0.929218+0.001320 train-error:0.158093+0.003108   tes
## [41] train-rmse:0.327937+0.001694    train-auc:0.930023+0.001379 train-error:0.157480+0.003106   tes
## [42] train-rmse:0.327184+0.001656    train-auc:0.930684+0.001377 train-error:0.156416+0.003253   tes
## [43] train-rmse:0.326425+0.001521    train-auc:0.931303+0.001260 train-error:0.155497+0.002907   tes
## [44] train-rmse:0.325603+0.001620    train-auc:0.931990+0.001330 train-error:0.154666+0.002618   tes
## [45] train-rmse:0.324805+0.001692    train-auc:0.932658+0.001348 train-error:0.153733+0.002719   tes
## [46] train-rmse:0.324063+0.001575    train-auc:0.933289+0.001216 train-error:0.153223+0.003100   tes
## [47] train-rmse:0.323439+0.001762    train-auc:0.933793+0.001388 train-error:0.153135+0.003544   tes
## [48] train-rmse:0.322854+0.001788    train-auc:0.934262+0.001377 train-error:0.152391+0.003528   tes
## [49] train-rmse:0.322252+0.001745    train-auc:0.934807+0.001355 train-error:0.151750+0.003334   tes
## [50] train-rmse:0.321600+0.001629    train-auc:0.935366+0.001298 train-error:0.151035+0.002842   tes
## [51] train-rmse:0.320875+0.001747    train-auc:0.935970+0.001400 train-error:0.150423+0.003509   tes
## [52] train-rmse:0.320162+0.001893    train-auc:0.936542+0.001521 train-error:0.149329+0.003485   tes
## [53] train-rmse:0.319647+0.001986    train-auc:0.936932+0.001590 train-error:0.149081+0.003185   tes
## [54] train-rmse:0.319180+0.002051    train-auc:0.937360+0.001665 train-error:0.148921+0.003133   tes
## [55] train-rmse:0.318495+0.002023    train-auc:0.937930+0.001619 train-error:0.148046+0.002944   tes
```

```
## [56] train-rmse:0.317847+0.001816    train-auc:0.938420+0.001408 train-error:0.147171+0.002895   tes
## [57] train-rmse:0.317274+0.001728    train-auc:0.938898+0.001332 train-error:0.146530+0.002161   tes
## [58] train-rmse:0.316655+0.001743    train-auc:0.939373+0.001312 train-error:0.145669+0.002031   tes
## [59] train-rmse:0.316001+0.001710    train-auc:0.939908+0.001291 train-error:0.145217+0.002054   tes
## [60] train-rmse:0.315394+0.001733    train-auc:0.940395+0.001311 train-error:0.144736+0.002320   tes
## [61] train-rmse:0.314790+0.001697    train-auc:0.940824+0.001289 train-error:0.144430+0.002118   tes
## [62] train-rmse:0.314159+0.001587    train-auc:0.941314+0.001225 train-error:0.143992+0.001886   tes
## [63] train-rmse:0.313643+0.001755    train-auc:0.941697+0.001368 train-error:0.143336+0.002088   tes
## [64] train-rmse:0.313051+0.001694    train-auc:0.942164+0.001256 train-error:0.142695+0.002148   tes
## [65] train-rmse:0.312558+0.001828    train-auc:0.942522+0.001354 train-error:0.142695+0.002514   tes
## [66] train-rmse:0.312013+0.001876    train-auc:0.942916+0.001425 train-error:0.141951+0.002486   tes
## [67] train-rmse:0.311434+0.001698    train-auc:0.943358+0.001266 train-error:0.141441+0.002384   tes
## [68] train-rmse:0.310875+0.001640    train-auc:0.943777+0.001186 train-error:0.141134+0.002037   tes
## [69] train-rmse:0.310278+0.001617    train-auc:0.944223+0.001195 train-error:0.140245+0.001969   tes
## [70] train-rmse:0.309696+0.001594    train-auc:0.944673+0.001148 train-error:0.139531+0.002061   tes
## [71] train-rmse:0.309181+0.001532    train-auc:0.945069+0.001093 train-error:0.138947+0.001563   tes
## [72] train-rmse:0.308746+0.001576    train-auc:0.945353+0.001151 train-error:0.138320+0.001831   tes
## [73] train-rmse:0.308254+0.001624    train-auc:0.945715+0.001192 train-error:0.138087+0.001517   tes
## [74] train-rmse:0.307611+0.001518    train-auc:0.946229+0.001126 train-error:0.137270+0.001555   tes
## [75] train-rmse:0.307122+0.001547    train-auc:0.946569+0.001098 train-error:0.136468+0.001889   tes
## [76] train-rmse:0.306571+0.001594    train-auc:0.946961+0.001127 train-error:0.135944+0.001965   tes
## [77] train-rmse:0.305946+0.001564    train-auc:0.947416+0.001071 train-error:0.135185+0.001789   tes
## [78] train-rmse:0.305473+0.001577    train-auc:0.947754+0.001094 train-error:0.134835+0.002163   tes
## [79] train-rmse:0.305042+0.001672    train-auc:0.948028+0.001165 train-error:0.134354+0.002092   tes
## [80] train-rmse:0.304559+0.001754    train-auc:0.948360+0.001245 train-error:0.133596+0.002231   tes
## [81] train-rmse:0.304144+0.001801    train-auc:0.948627+0.001221 train-error:0.133231+0.002122   tes
## [82] train-rmse:0.303739+0.001734    train-auc:0.948914+0.001159 train-error:0.133348+0.002223   tes
## [83] train-rmse:0.303312+0.001882    train-auc:0.949232+0.001253 train-error:0.132458+0.002218   tes
## [84] train-rmse:0.302778+0.001926    train-auc:0.949588+0.001289 train-error:0.131715+0.002569   tes
## [85] train-rmse:0.302109+0.001988    train-auc:0.950044+0.001358 train-error:0.131030+0.002782   tes
## [86] train-rmse:0.301626+0.002012    train-auc:0.950331+0.001356 train-error:0.130767+0.002845   tes
## [87] train-rmse:0.300992+0.001877    train-auc:0.950830+0.001264 train-error:0.130300+0.002366   tes
## [88] train-rmse:0.300468+0.001871    train-auc:0.951155+0.001258 train-error:0.129863+0.002225   tes
## [89] train-rmse:0.299921+0.001991    train-auc:0.951500+0.001340 train-error:0.129251+0.002430   tes
## [90] train-rmse:0.299382+0.002110    train-auc:0.951856+0.001388 train-error:0.128653+0.002764   tes
## [91] train-rmse:0.298813+0.002120    train-auc:0.952226+0.001396 train-error:0.128055+0.002812   tes
## [92] train-rmse:0.298246+0.002226    train-auc:0.952622+0.001474 train-error:0.127647+0.002876   tes
## [93] train-rmse:0.297793+0.002160    train-auc:0.952924+0.001428 train-error:0.127238+0.002568   tes
## [94] train-rmse:0.297216+0.002172    train-auc:0.953288+0.001439 train-error:0.126947+0.002558   tes
## [95] train-rmse:0.296801+0.002170    train-auc:0.953534+0.001385 train-error:0.126465+0.002678   tes
## [96] train-rmse:0.296367+0.002323    train-auc:0.953836+0.001481 train-error:0.125722+0.002673   tes
## [97] train-rmse:0.295864+0.002369    train-auc:0.954187+0.001540 train-error:0.125882+0.002883   tes
## [98] train-rmse:0.295462+0.002359    train-auc:0.954423+0.001510 train-error:0.125299+0.002879   tes
## [99] train-rmse:0.295035+0.002460    train-auc:0.954663+0.001593 train-error:0.124993+0.002777   tes
## [100]    train-rmse:0.294489+0.002418    train-auc:0.955032+0.001500 train-error:0.124555+0.002784
```

```r
trained_model <- xgb.train(data = xgb_train, max_depth = 3,
              eta = 1, nthread = 4, nrounds = 14,
              watchlist = list(train = xgb_train, eval = xgb_test),
              objective = "binary:logistic")
```

```
## [1]  train-logloss:0.518785  eval-logloss:0.820399
## [2]  train-logloss:0.460336  eval-logloss:0.881454
## [3]  train-logloss:0.443383  eval-logloss:0.952815
```

```
## [4]  train-logloss:0.429115  eval-logloss:1.006953
## [5]  train-logloss:0.421462  eval-logloss:1.068660
## [6]  train-logloss:0.417349  eval-logloss:1.082765
## [7]  train-logloss:0.404430  eval-logloss:1.115529
## [8]  train-logloss:0.401043  eval-logloss:1.130244
## [9]  train-logloss:0.396576  eval-logloss:1.161850
## [10] train-logloss:0.393187  eval-logloss:1.185680
## [11] train-logloss:0.390200  eval-logloss:1.205786
## [12] train-logloss:0.388254  eval-logloss:1.226213
## [13] train-logloss:0.386829  eval-logloss:1.243061
## [14] train-logloss:0.383553  eval-logloss:1.241412
```

```
predictions <- predictions %>%
  mutate(Transported=ifelse(Transported>.5,'True','False'))

write_csv(predictions, "C:\\Users\\erico\\Desktop\\XGBpredictions_Rounds14.csv")
```

2. Tune SVM Radial: No Improvement

We manually hypertune our svm radial model by increasing sigma (loosening the fit) and decreasing sigma (tightening the fit). A decreased sigma increased accuracy on the holdout set but not on the Kaggle.

```
library(doParallel)
library(caret)

cl<-makePSOCKcluster(10)

registerDoParallel(cl)

#rad3 <- EHModel_SVM(dfTrain5zz, "Transported", method="radial", cValue=1, sigmaValue =.05)

rad3 <- EHModel_SVM(dfTrain5zz, "Transported", method="radial", cValue=1, sigmaValue =.04)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 6097 samples
##   27 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (27), scaled (27)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 5487, 5487, 5487, 5488, 5488, 5488, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7959085  0.5918032
##
## Tuning parameter 'sigma' was held constant at a value of 0.04
## Tuning
##  parameter 'C' was held constant at a value of 1
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0 576 163
##          1 181 603
##
##                Accuracy : 0.7741
##                  95% CI : (0.7523, 0.7949)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5482
##
##  Mcnemar's Test P-Value : 0.3594
##
##             Sensitivity : 0.7609
##             Specificity : 0.7872
##          Pos Pred Value : 0.7794
##          Neg Pred Value : 0.7691
##              Prevalence : 0.4970
##          Detection Rate : 0.3782
##    Detection Prevalence : 0.4852
##       Balanced Accuracy : 0.7741
##
##        'Positive' Class : 0
##
```

```
predictions_rad3 <- EHModel_Predict(rad3$svm, dfTest5zz, predictionsColumnName = "Transported", testData

predictions_rad3$Transported <- ifelse(predictions_rad3$Transported==1,"True","False")
write_csv(predictions_rad3, "C://Users//erico//Desktop//SVM_Rad_LHigherSigma.csv")

stopCluster(cl)
```

Discussion

(see above)