```
---
title: "Credit Card Fraud Analysis"
author: "Daniel Starer"
date: "2025-08-12"
output: word_document
upd: "2025-08-18"
---
```

This analysis concerns the Credit Card Transactions dataset compiled by
Kelvin Obiri on Kaggle.  The purpose of this is to test different models'
accuracy when it comes to determining the presence of credit card fraud.

First, we will load the necessary libraries for this problem.
```{r}
library("tree")
library("caret")
```

Next, we will load the dataset.

```{r}
CCardData <- read.csv("/home/dan/Desktop/Personal Data Science Projects/Credit Card Fraud
Check/archive/transactions.csv")
```

Since this is a classification problem, we'll need to get a good understanding of
the types of variables/columns we are working with here.
```{r}
head(CCardData)
```

Description: df [6 × 10]

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest |
|---|---|---|---|---|---|---|---|---|
| | <int> | <chr> | <dbl> | <chr> | <dbl> | <dbl> | <chr> | <dbl> |
| 1 | 8 | CASH_OUT | 158007.12 | C424875646 | 0 | 0 | C1298177219 | 474016.32 |
| 2 | 236 | CASH_OUT | 457948.30 | C1342616552 | 0 | 0 | C1323169990 | 2720411.37 |
| 3 | 37 | CASH_IN | 153602.99 | C900876541 | 11160429 | 11314032 | C608741097 | 3274930.56 |
| 4 | 331 | CASH_OUT | 49555.14 | C177696810 | 10865 | 0 | C462716348 | 0.00 |
| 5 | 250 | CASH_OUT | 29648.02 | C788941490 | 0 | 0 | C1971700992 | 56933.09 |
| 6 | 182 | PAYMENT | 15712.66 | C365217190 | 13981 | 0 | M1108542644 | 0.00 |

| newbalanceDest | isFraud |
|---:|---:|
| <dbl> | <int> |
| 1618631.97 | 0 |
| 3178359.67 | 0 |
| 3121327.56 | 0 |
| 49555.14 | 0 |
| 86581.10 | 0 |

6 rows | 1-9 of 10 columns

We can see there is a step, a type, an amount, nameOrig, oldbalanceOrg, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest, and IsFraud variables.  What we will do here is do a bit of cleaning so we can isolate only the variables that are important to us. Namely, we will not be including nameOrig and nameDest because the account names do not carry much meaning in predicting future credit card fraud.

~~~
```{r}
myCCardData <- CCardData[, c(1, 2, 3, 5, 6, 8, 9, 10)]
```
~~~

Now we are going to do something called Upsampling.  Upsampling effectively increases the number of samples in the minority class (fraud/1) to balance it out with the majority class.  I did try another method called SMOTE, which would have created synthetic samples that reflect already existing samples, but unfortunately SMOTE has a memory limit that it was constantly hitting with this data and the successor to it, SMOTEfamily does not install successfully.  So, rather than use downsampling, which would effectively remove some non-fraudulant cases or use another method called ROSE, which works better for cases in which the minority class can be more accurately estimated.  We are dealing with a problem here where we do not know what future samples we will have, so it is better to add more cases of fraud for the model to train on since that is what we are trying to catch and in the end, it is better to have a misclassification of non-fraud than it is to have one of fraud.

~~~
```{r}
set.seed(123)
myCCardData$isFraud <- as.factor(myCCardData$isFraud)
myCCardData$type <- as.factor(myCCardData$type)
CCardData_UPSam <- upSample(x = myCCardData[, setdiff(names(myCCardData), "isFraud")], y =
myCCardData$isFraud, yname = "isFraud")
table(CCardData_UPSam$isFraud)
```
~~~

```
     0      1
199717 199717
```

Now, as we can see the fraud and non-fraudulant cases are divided equally, so the imbalance in the data is gone.

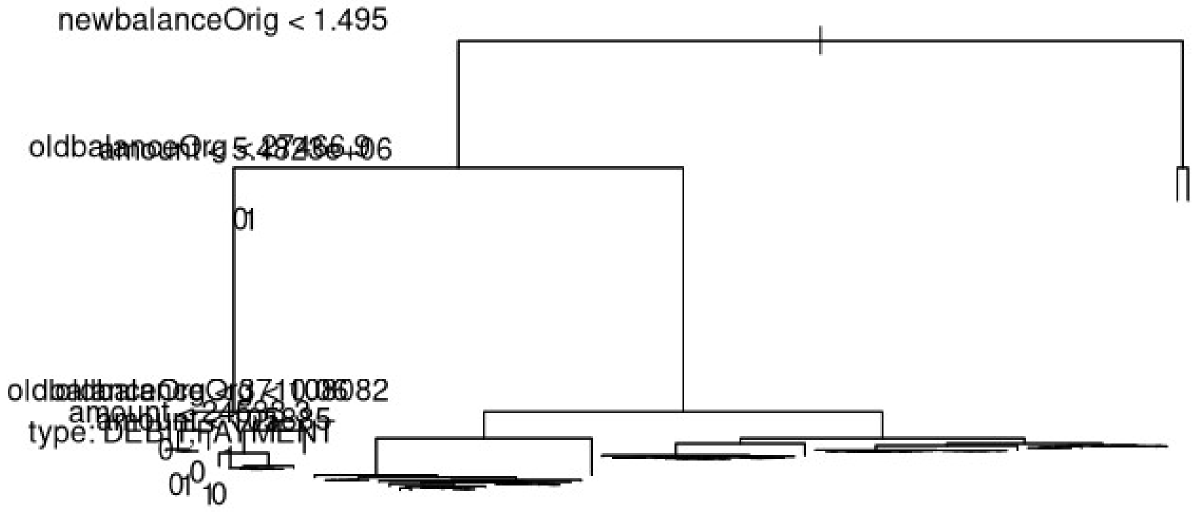
Next, let's separate the data into test and training data.

```{r}
set.seed(123)
indis <- sample(1:nrow(CCardData_UPSam), 2/3*nrow(CCardData_UPSam))
test <- CCardData_UPSam[indis,]
train <- CCardData_UPSam[-indis,]
```

Now, we will fit the dataset, CCardData_UPSam that we just created to a tree.

```{r}
set.seed(123)
fit <- tree(isFraud ~ ., data = train, split = "gini")
summary(fit)
plot(fit)
text(pruned_tree, pretty = 0)
```

```
Classification tree:
tree(formula = isFraud ~ ., data = train, split = "gini")
Number of terminal nodes:  97
Residual mean deviance:  0.004145 = 551.5 / 133000
Misclassification error rate: 0.0003755 = 50 / 133145
```



We can see from this result that there are 97 terminal nodes with a residual mean deviance of 0.004145, which is phenomenal! There are 551.5 degrees of freedom and 133,000 data points being tested here. The misclassification error rate here is 0.0003755 or close to 0%, which is very good as well. However, you can see that this tree is not easy to interpret. For that reason, we will see if we can prune and cross-validate the tree to see if that helps with interpretability. Before that, we will implement something called a confusion matrix to help evaluate the predictive accuracy of this model.

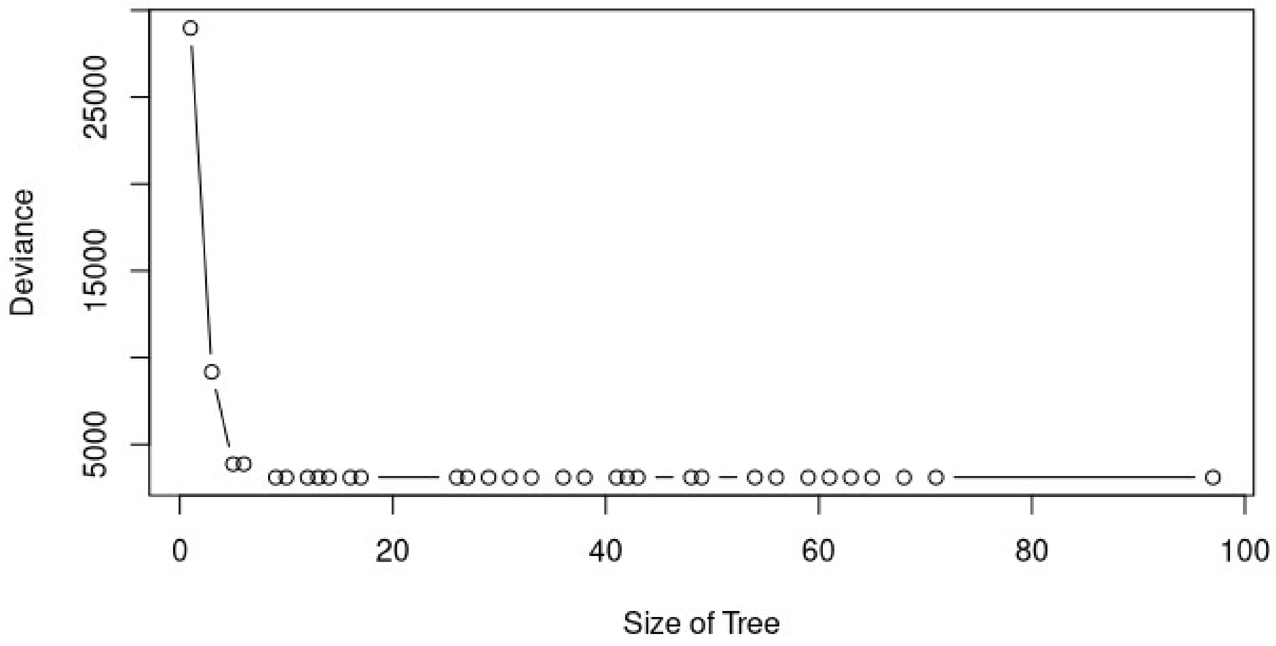
```{r}
treepred <- predict(fit, train, type = "class")
treecm <- table(Predicted = treepred, Actual = train$isFraud)
print(treecm)
```

```
         Actual
Predicted     0     1
        0 66544     0
        1    50 66551
```

This confusion matrix is essentially read like a 2 x 2 cell table. The predicted values are on the left going vertically and the actual values are above, going horizontally. So, for example, we can see that 66,544 values were predicted to not be fraud and 66,544 values were not actually fraud. This means those values are correctly being identified by the model as not being fraud. Below that, we can see 50 cases are being predicted to be fraud, but in actuality, they are not cases of fraud. On the inverse, in the top right, we can see no cases are being misidentified as not being fraud cases, so no fraud cases are going undetected with this model.

Now, let's prune the tree to help with the interpretability problem.

~~~
```{r}
cv_fit <- cv.tree(fit, FUN = prune.misclass)
cv_fit
plot(cv_fit$size, cv_fit$dev, type = "b", xlab = "Size of Tree", ylab = "Deviance")
```
~~~

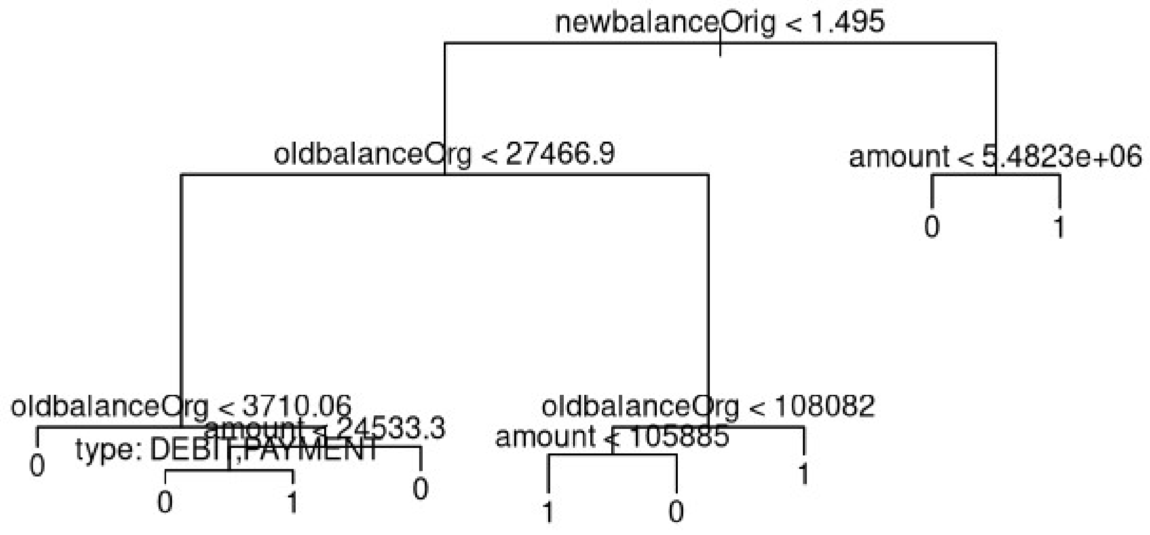


We can see by this visualization that the deviance starts leveling out at around a tree size of 5 with a little less than 5000 as deviance.

Let's see what R thinks the minimum acceptable value for the tree size is and prune the tree.

```{r}
min_dev <- min(cv_fit$dev)
threshold <- min_dev * 1.05
candidate_sizes <- cv_fit$size[cv_fit$dev <= threshold]
best_size <- min(candidate_sizes)
best_size

pruned_tree <- prune.misclass(fit, best = best_size)
summary(pruned_tree)
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```

```
[1] 9

Classification tree:
snip.tree(tree = fit, nodes = c(37L, 20L, 11L, 8L))
Variables actually used in tree construction:
[1] "newbalanceOrig" "oldbalanceOrg"  "amount"         "type"
Number of terminal nodes:  9
Residual mean deviance:  0.1829 = 24350 / 133100
Misclassification error rate: 0.02083 = 2774 / 133145
```



We can see here that the ideal number of leaves for the pruned tree was 9 and
16 branches showing improved interpretability, so we assigned
that to a variable, best-size and pruned the tree accordingly. From the summary
output of the pruned tree, we can see that we have a residual mean deviance of
0.1829, which is slightly worse than the tree before it was pruned and a
misclassification error rate of 0.0003755.

Let's check the confusion matrix for this.

```{r}
prunedtreepred <- predict(pruned_tree, train, type = "class")
prunedtreecm <- table(Predicted = prunedtreepred, Actual = train$isFraud)
print(prunedtreecm)
```

```
         Actual
Predicted     0     1
        0 64508   688
        1  2086 65863
```

We can see that the confusion matrix increases the rate of misclassifications.
688 out of 65,863 cases were misclassified as non-fraudulent or 0.01044592563 (about 1% of cases). 2086 out of 64,508 cases were misclassified as being fraud when they really weren't or 0.03233707447 (about 3% of cases). This isn't substantially large, however, it could create some issues for some account holders having their accounts being flagged as having evidence of credit card fraud.
Also, while 1% of accounts being misclassified as not being fraudulent, the damage this could cause a credit card company is dependent on how much money these fraudulent accounts are moving around. Regardless, for interpretability, this is still the best version of the tree so far.

So, it would appear that the unpruned tree does the best so far, but let us now compare the results of this to some other models we can test.

First, we're just going to perform something called rpart with cp tuning.
Rpart stands for "Recursive PARTitioning and Regression Trees", which is used with cp, the complexity parameter to make sure a classification tree is not overfitting the data. An overfitted tree can lead to bad generalization of the data. Interpretability is also lost with overfitted trees because of the amount of branches they can have. We're going to use the upsampled dataset because rpart doesn't work with datasets that already have the tree classifier.
We'll have to create a special dataset for the rpart tree because it will implement the complexity parameter and uses the rpart dependency.

```{r}
library("rpart")
set.seed(123)
rpartUPSam <- rpart(isFraud ~ ., data = train, cp = 0.001)
```

We keep the target cp as close to 0 as possible to find out exactly where or if the tree becomes overfitted. Now, we will print the cp values in the rpart.

```{r}
printcp(rpartUPSam)
```

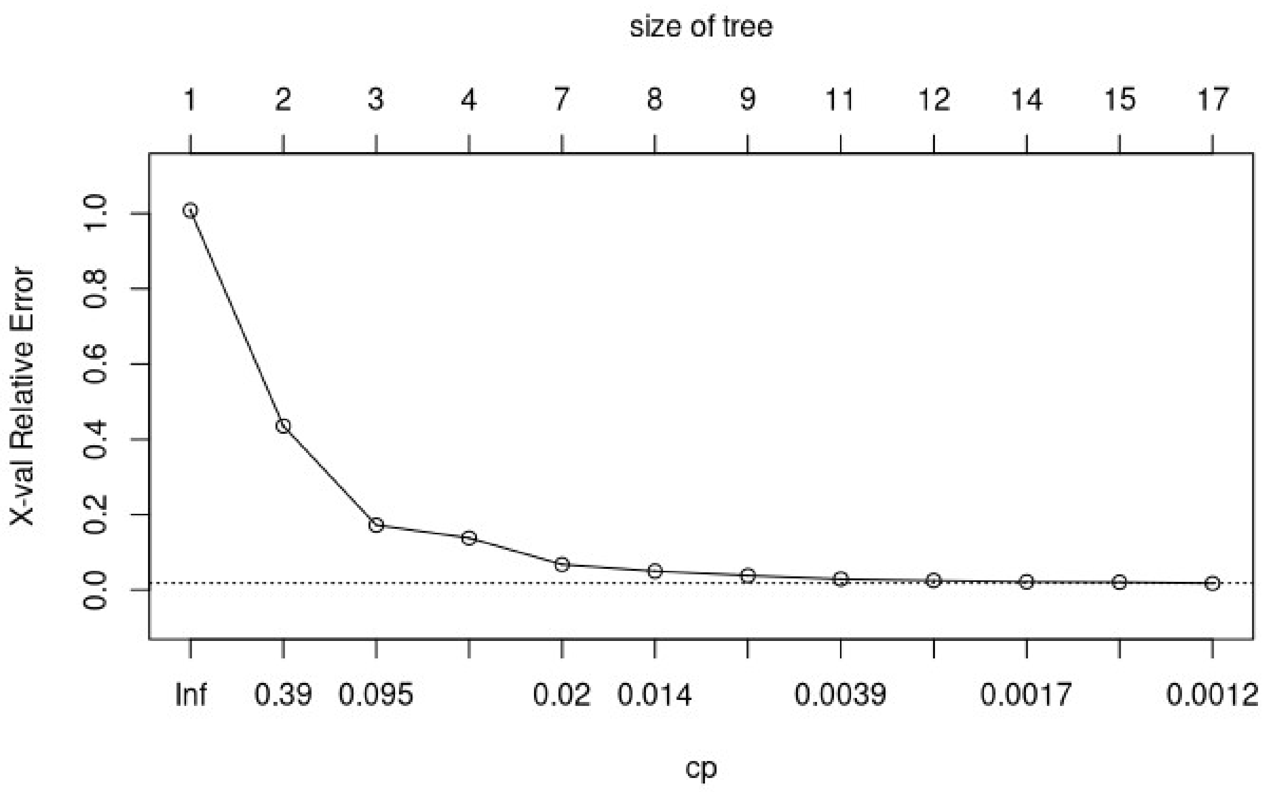
```
Classification tree:

rpart(formula = isFraud ~ ., data = train, cp = 0.001)

Variables actually used in tree construction:
[1] amount         newbalanceDest newbalanceOrig oldbalanceOrg  step           type

Root node error: 66551/133145 = 0.49984
```

```
n= 133145
```

| Tree # | CP | nsplit | rel error | xerror | xstd |
|---|---|---|---|---|---|
| 1 | 0.5644694 | 0 | 1.000000 | 1.008249 | 0.00274135 |
| 2 | 0.2634220 | 1 | 0.435531 | 0.435531 | 0.00226266 |
| 3 | 0.0342444 | 2 | 0.172109 | 0.172124 | 0.00153747 |
| 4 | 0.0234256 | 3 | 0.137864 | 0.137864 | 0.00138881 |
| 5 | 0.0177608 | 6 | 0.067587 | 0.067587 | 0.00099059 |
| 6 | 0.0118105 | 7 | 0.049826 | 0.049826 | 0.00085443 |
| 7 | 0.0046431 | 8 | 0.038016 | 0.038286 | 0.00075119 |
| 8 | 0.0032306 | 10 | 0.028730 | 0.029015 | 0.00065549 |
| 9 | 0.0019609 | 11 | 0.025499 | 0.025063 | 0.00060983 |
| 10 | 0.0015176 | 13 | 0.021577 | 0.021758 | 0.00056866 |
| 11 | 0.0013523 | 14 | 0.020060 | 0.020601 | 0.00055350 |
| 12 | 0.0010000 | 16 | 0.017355 | 0.017385 | 0.00050888 |

Our target CP of 0.001 performed pretty well, resulting in 16 splits and a xerror or cross-validated error of 0.017385 and xstd or standard cross-validated error of 0.00050888. But, considering that the misclassification error of our original tree was 0.0003755, there would appear based off of only that that there is no evidence of significant overfitting and the model generalizes well to the data. However, what is important to keep in mind is that interpretability is lost without some form of pruning. The pruned tree pictured above that had 16 splits/branches looked good because it had the minimum residual mean deviance at the lowest size. Also, based on the visualization, you can see that the pruned tree with 10 branches is much easier to interpret than the tree without pruning. Therefore, the pruned tree, though resulting in a negligible difference in xerror and xstd is still the better choice, visually and performance-wise.

~~~
```{r}
plotcp(rpartUPSam)
```
~~~



Next, we can try a random forest model, which is more computationally expensive, but basically it builds on the idea of classification trees and generates several hundred to several thousand trees at one time.

~~~
```{r}
library("randomForest")
set.seed(123)
CCardrandForest <- randomForest(isFraud ~ ., data = train, ntree = 500)
```
~~~

I'm going to print the output results in a separate markdown box here because as mentioned previously random forests are computationally expensive! Depending on the amount of trees generated by R and the hardware on the computer in question it can take a while to run.

~~~
```{r}
library(devtools)
library(reprtree)
print(CCardrandForest)
RFTree1 <- getTree(CCardrandForest, k = 1, labelVar = TRUE)
~~~

reprtree:::plot.getTree(CCardrandForest)
```

```
Call:
 randomForest(formula = isFraud ~ ., data = train, ntree = 500,      mtry = 2)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 0.01%
Confusion matrix:
      0     1  class.error
0 66576    18 0.0002702946
1     0 66551 0.0000000000
```

amount < 509976.38
step < 411
newbalanceDest < 2835.54
oldbalanceDest < 123729.6
oldbalanceOrg < 37026.275
newbalanceDest < 914252.195
oldbalanceOrg < 54071.63
oldbalanceOrg < 305840.715

This random forest ended up performing even better than the single classification tree.  By printing a random forest's results, we can see that it looks a bit different than the results for a classification tree.  This is because we cannot visualize all 500 trees at once and even larger random forest models are even more difficult to display.  We can visualize one of the trees, though.  We do have some useful
```

information here that will help us tell the performance of this model.  Namely, OOB estimate of error rate refers to the Out of Bag error rate, which estimates the error rate on unseen data in a random forest. This is 0.01%, which is very good. And next we have the confusion matrix. Here we have only 18 cases out of 66,551 or 0.0002702946 (less than 1%) of misclassifications for fraudulent cases and no misclassified non-fraud cases. This is the best performing model for us so far because it minimizes headaches for account holders who do not have fraudulent activity on their accounts.  However, as you can see from a view of the first tree from our random forest, the tree has no interpretability.  This isWe can see with both of these tests for the test set, the results closely
mirror the results we were able to obtain with the training set, confirming our theory that a ridge regression model is not good for deployment.

In conclusion, of the models we tested, the best performing model was the random forest model.  It was computationally expensive, having taken up several gigabytes of data in R's virtual memory, but it produced the most accurate result in fraud detection, which shouldn't really come as a surprise since fraud detection is a statistical classification problem.  Our goal is to identify what is fraud and what isn't, and it does that well.  For interpretability, you could also use the 9 leaf and 16 branch pruned classification tree, which is much easier to run on most computers and is about just as accurate.
We can see with both of these tests for the test set, the results closely mirror the results we were able to obtain with the training set, confirming our theory that a ridge regression model is not good for deployment.

In conclusion, of the models we tested, the best performing model was the random forest model.  It was computationally expensive, having taken up several gigabytes of data in R's virtual memory, but it produced the most accurate result in fraud detection, which shouldn't really come as a surprise since fraud detection is a statistical classification problem.  Our goal is to identify what is fraud and what isn't, and it does that well.  For interpretability, you could also use the 9 leaf and 16 branch pruned classification tree, which is much easier to run on most computers and is about just as accurate.actually one of the main problems with random forest models.

Let's see though, if there is another model that isn't as computationally expensive that achieves a similar result with more interpretability.

Let's try a ridge regression model.  This should be compatible with the dataset we have considering we are coming from a random forest model and that the isFraud variable is already in a binary form.  We're also going to define a new cv_fit for ridge regression because this will be different from the cross validation we performed for the classification tree.  We're going to be introducing something here called lambda.  Lambda is a tuning parameter for ridge regression models that

controls the amount of regularization applied to the modWe can see with both of these tests for the test set, the results closely
mirror the results we were able to obtain with the training set, confirming
our theory that a ridge regression model is not good for deployment.

In conclusion, of the models we tested, the best performing model was the
random forest model. It was computationally expensive, having taken up
several gigabytes of data in R's virtual memory, but it produced the most
accurate result in fraud detection, which shouldn't really come as a surprise
since fraud detection is a statistical classification problem. Our goal is to
identify what is fraud and what isn't, and it does that well. For interpretability,
you could also use the 9 leaf and 16 branch pruned classification tree, which is
much easier to run on most computers and is about just as accurate.el. We want to find the
optimal amount of regularization to use here.

```{r}
library("glmnet")
set.seed(123)
ridgex_train <- model.matrix(isFraud ~ ., train[, -1])
ridgey_train <- train$isFraud
ridgex_test <- model.matrix(isFraud ~ ., test[, -1])
ridgey_test <- test$isFraud
ridgecv_fit <- cv.glmnet(ridgex_train, ridgey_train, family = "binomial", alpha = 0, nfolds = 10)
ridgebest_lambda <- cv_fit$lambda
CCridge_fit <- glmnet(ridgex_train, ridgey_train, family = "binomial", alpha = 0, lambda =
ridgebest_lambda)
```

Now that we have created the model, we will need to now output the accuracy of
this model. We can do that by first calculating two variables, y hat train and
y hat test, which are essentially the predicted values of y or in essence, what
we predict the result will be, fraud or no fraud.

```{r}
ridgeglm.probs.train <- predict(ridgecv_fit, newx = ridgex_train, s = "lambda.min", type = "response")
ridgey_hat_train <- ifelse(ridgeglm.probs.train > 0.5, 1, 0)
ridgeglm.probs.test <- predict(ridgecv_fit, newx = ridgex_test, s = "lambda.min", type = "response")
ridgey_hat_test <- ifelse(ridgeglm.probs.test > 0.5, 1, 0)
ridgetrain_acc <- mean(ridgey_hat_train == ridgey_train)
ridgetest_acc <- mean(ridgey_hat_test == ridgey_hat_test)
ridgetest_acc
ridgetrain_acc
```

```
[1] 1
```

```
[1] 0.8165384
```

So, we can see here from our ridge model, our test accuracy has 1, which is perfect.
Accuracy here is out of 1, so 1/1 is 100%. Our training accuracy is .8165384 or about 82%.
Very accurate model, especially in the testing side of things. This is an important
point because in deployment, it will be the test data that will be used with new data.
However, it is very rare for test accuracy to be 100%, so the model may be overfitted.
We will have to check by performing two other tests to check for model accuracy:
a confusion matrix and AUC (Area Under the Receiver Operating Characteristic (ROC) Curve).
We'll try converting the y-hat test and training data here to the factor data type.

~~~
```{r}
library(pROC)
ridgey_hat_test <- factor(ridgey_hat_test, levels = c(0, 1))
ridgey_hat_train <- factor(ridgey_hat_train, levels = c(0, 1))
ridgecm_train <- confusionMatrix(ridgey_hat_train, ridgey_train, positive = "1")
ridgeauc_train <- auc(ridgey_train, as.numeric(ridgeglm.probs.train))
ridgecm_test <- confusionMatrix(ridgey_hat_test, ridgey_test, positive = "1")
ridgeauc_test <- auc(ridgey_test, as.numeric(ridgeglm.probs.test))
```
~~~

This will produce a long list of data to analyze, so let's first output the ridge
training set confusion matrix.

~~~
```{r}
ridgecm_train
```
~~~

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 46946  4779
         1 19648 61772

               Accuracy : 0.8165
                 95% CI : (0.8144, 0.8186)
    No Information Rate : 0.5002
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6331

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9282
            Specificity : 0.7050
```

```
         Pos Pred Value : 0.7587
         Neg Pred Value : 0.9076
             Prevalence : 0.4998
         Detection Rate : 0.4639
   Detection Prevalence : 0.6115
      Balanced Accuracy : 0.8166

       'Positive' Class : 1
```

We can see that there are actually a lot of misclassifications here. 4779 values have been misclassified as fraud when they weren't out of 61,772 or about 8% of fraud cases and 19,6We can see with both of these tests for the test set, the results closely mirror the results we were able to obtain with the training set, confirming our theory that a ridge regression model is not good for deployment.

In conclusion, of the models we tested, the best performing model was the random forest model. It was computationally expensive, having taken up several gigabytes of data in R's virtual memory, but it produced the most accurate result in fraud detection, which shouldn't really come as a surprise since fraud detection is a statistical classification problem. Our goal is to identify what is fraud and what isn't, and it does that well. For interpretability, you could also use the 9 leaf and 16 branch pruned classification tree, which is much easier to run on most computers and is about just as accurate.48 values out of 46,946 or 42% of non-fraud cases were misclassified.
This would be a completely unacceptable model to use since it would drastically increase the amount of non-fraudulent activity being misidentified as fraud and unfairly flagging account holders' accounts as having fraud. The confusionMatrix code we implemented here also shows the p-value and Mcnemar's p-value. The p-value tells us whether or not we should reject the null hypothesis and whether observed results are statistically significant. Given that the p-value is less than 0.05, we can say this result is statistically significant. In this case, our null-hypothesis would be "this data is not good at detecting fraudulent activity", which the p-value being less than 0.05 would tell us that we would reject that null-hypothesis and state, "yes, this model is good at detecting fraudulent activity". Mcnemar's Test P-value is the same as the p-value test except that it test specifically the relationship between two measurements. The sensitivity here is 0.9282, meaning the model catches 92, almost 93% of all fraud cases. Specificity is how well the model performs at detecting false positives or false negative cases, which is 0.7050 or 70%, which isn't ideal. The positive prediction value or number of correct predictions is only 0.7587 or about 76%, which means a little more than 1/4 of fraudulent cases are still going undetected with the ridge regression model. Now, let's check the AUC for the model's training set.

```{r}
```

ridgeauc_train
```

```
Area under the curve: 0.9218

The AUC result here of 0.9214 alongside the sensitivity of 0.9297 and the specificity of 0.6977 would indicate that there is at least no data leakage and that the model generalizes reasonably well to the data. However, the specificity is a problem in this case because it could create unnecessary problems for account holders who are flagged as having fraudulent activity on their accounts. Additionally, with the positive prediction value being only 0.7559, the company would potentially lose money using this model because 1/4 of their credit card cases would go undetected. There isn't evidence of overfitting, but this model would not be good for deployment. Just to see, let's take a look at the results of the confusionMatrix and AUC for the test set.

```{r}
ridgecm_test
```

```
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0  93531   9456
         1  39592 123710

               Accuracy : 0.8158
                 95% CI : (0.8143, 0.8173)
    No Information Rate : 0.5001
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6316

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9290
            Specificity : 0.7026
         Pos Pred Value : 0.7576
         Neg Pred Value : 0.9082
             Prevalence : 0.5001
         Detection Rate : 0.4646
   Detection Prevalence : 0.6133
      Balanced Accuracy : 0.8158

       'Positive' Class : 1
```

```{r}
ridgeauc_test
```

```
Area under the curve: 0.9213
```

We can see with both of these tests for the test set, the results closely mirror the results we were able to obtain with the training set, confirming our theory that a ridge regression model is not good for deployment.

In conclusion, of the models we tested, the best performing model was the random forest model.  It was computationally expensive, having taken up several gigabytes of data in R's virtual memory, but it produced the most accurate result in fraud detection, which shouldn't really come as a surprise since fraud detection is a statistical classification problem.  Our goal is to identify what is fraud and what isn't, and it does that well.  For interpretability, you could also use the 9 leaf and 16 branch pruned classification tree, which is much easier to run on most computers and is about just as accurate.