Decision Trees and Random Forest Project

February 5, 2018

1 Random Forest Project

For this project we will be exploring publicly available data from LendingClub.com. Lending Club connects people who need money (borrowers) with people who have money (investors). Hopefully, as an investor you would want to invest in people who showed a profile of having a high probability of paying you back. We will try to create a model that will help predict this.

Lending club had a very interesting year in 2016, so let's check out some of their data and keep the context in mind. This data is from before they even went public.

We will use lending data from 2007-2010 and be trying to classify and predict whether or not the borrower paid back their loan in full. You can download the data from here or just use the csv already provided. It's recommended you use the csv provided as it has been cleaned of NA values.

Here are what the columns represent: * credit.policy: 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 otherwise. * purpose: The purpose of the loan (takes values "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business", and "all_other"). * int.rate: The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher interest rates. * installment: The monthly installments owed by the borrower if the loan is funded. * log.annual.inc: The natural log of the self-reported annual income of the borrower. * dti: The debt-to-income ratio of the borrower (amount of debt divided by annual income). * fico: The FICO credit score of the borrower. * days.with.cr.line: The number of days the borrower has had a credit line. * revol.bal: The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle). * revol.util: The borrower's revolving line utilization rate (the amount of the credit line used relative to total credit available). * inq.last.6mths: The borrower's number of inquiries by creditors in the last 6 months. * delinq.2yrs: The number of times the borrower had been 30+ days past due on a payment in the past 2 years. * pub.rec: The borrower's number of derogatory public records (bankruptcy filings, tax liens, or judgments).

2 Import Libraries

Import the usual libraries for pandas and plotting. You can import sklearn later on.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

2.1 Get the Data

** Use pandas to read loan_data.csv as a dataframe called loans.**

```
In [2]: loans = pd.read_csv('loan_data.csv')
```

** Check out the info(), head(), and describe() methods on loans.**

```
In [3]: loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
credit.policy        9578 non-null int64
purpose              9578 non-null object
int.rate             9578 non-null float64
installment          9578 non-null float64
log.annual.inc       9578 non-null float64
dti                  9578 non-null float64
fico                 9578 non-null int64
days.with.cr.line    9578 non-null float64
revol.bal            9578 non-null int64
revol.util           9578 non-null float64
inq.last.6mths       9578 non-null int64
delinq.2yrs          9578 non-null int64
pub.rec              9578 non-null int64
not.fully.paid       9578 non-null int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

```
In [4]: loans.describe()
```

```
Out[4]:        credit.policy     int.rate  installment  log.annual.inc          dti  \
        count    9578.000000  9578.000000  9578.000000     9578.000000  9578.000000
        mean        0.804970     0.122640   319.089413       10.932117    12.606679
        std         0.396245     0.026847   207.071301        0.614813     6.883970
        min         0.000000     0.060000    15.670000        7.547502     0.000000
        25%         1.000000     0.103900   163.770000       10.558414     7.212500
        50%         1.000000     0.122100   268.950000       10.928884    12.665000
        75%         1.000000     0.140700   432.762500       11.291293    17.950000
        max         1.000000     0.216400   940.140000       14.528354    29.960000

                     fico  days.with.cr.line     revol.bal   revol.util  \
        count  9578.000000        9578.000000  9.578000e+03  9578.000000
        mean    710.846314        4560.767197  1.691396e+04    46.799236
        std      37.970537        2496.930377  3.375619e+04    29.014417
        min     612.000000         178.958333  0.000000e+00     0.000000
        25%     682.000000        2820.000000  3.187000e+03    22.600000
```

```
50%      707.000000         4139.958333  8.596000e+03    46.300000
75%      737.000000         5730.000000  1.824950e+04    70.900000
max      827.000000        17639.958330  1.207359e+06   119.000000

       inq.last.6mths  delinq.2yrs      pub.rec  not.fully.paid
count     9578.000000  9578.000000  9578.000000     9578.000000
mean         1.577469     0.163708     0.062122        0.160054
std          2.200245     0.546215     0.262126        0.366676
min          0.000000     0.000000     0.000000        0.000000
25%          0.000000     0.000000     0.000000        0.000000
50%          1.000000     0.000000     0.000000        0.000000
75%          2.000000     0.000000     0.000000        0.000000
max         33.000000    13.000000     5.000000        1.000000
```

In [5]:
```
loans.head()
```

Out[5]:
```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

3 Exploratory Data Analysis

Let's do some data visualization! We'll use seaborn and pandas built-in plotting capabilities, but feel free to use whatever library you want. Don't worry about the colors matching, just worry about getting the main idea of the plot.

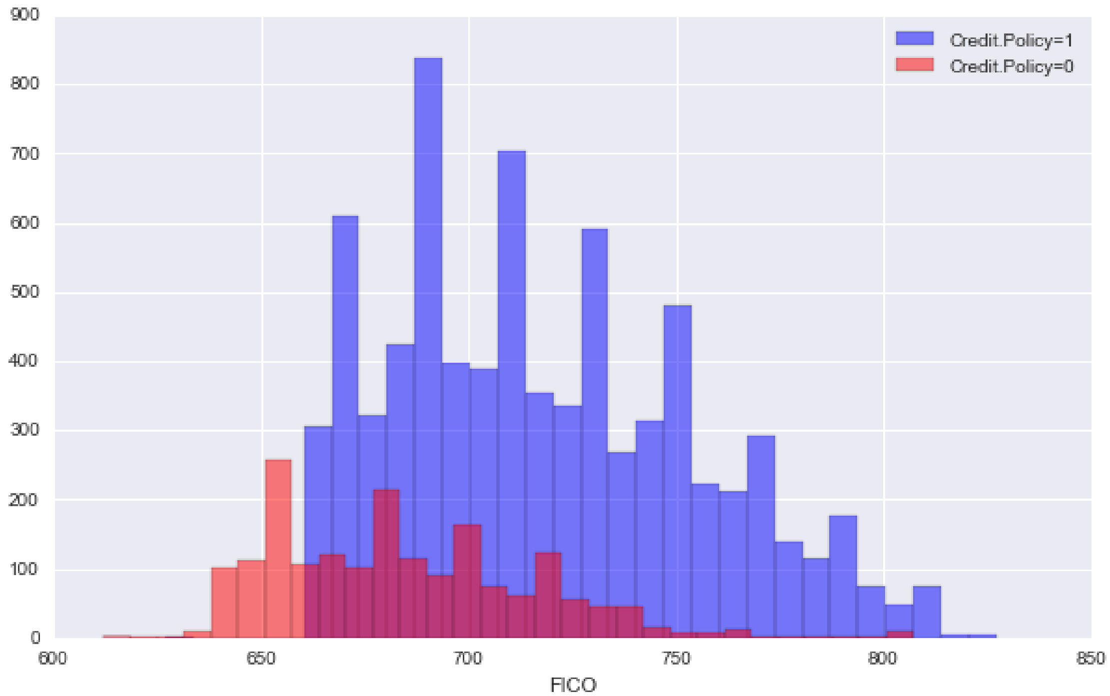
** Create a histogram of two FICO distributions on top of each other, one for each credit.policy outcome.**

Note: This is pretty tricky, feel free to reference the solutions. You'll probably need one line of code for each histogram, I also recommend just using pandas built in .hist()

In [6]:
```
plt.figure(figsize=(10,6))
loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',
```

```
                                               bins=30,label='Credit.Policy=1')
loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5,color='red',
                                               bins=30,label='Credit.Policy=0')
plt.legend()
plt.xlabel('FICO')
```

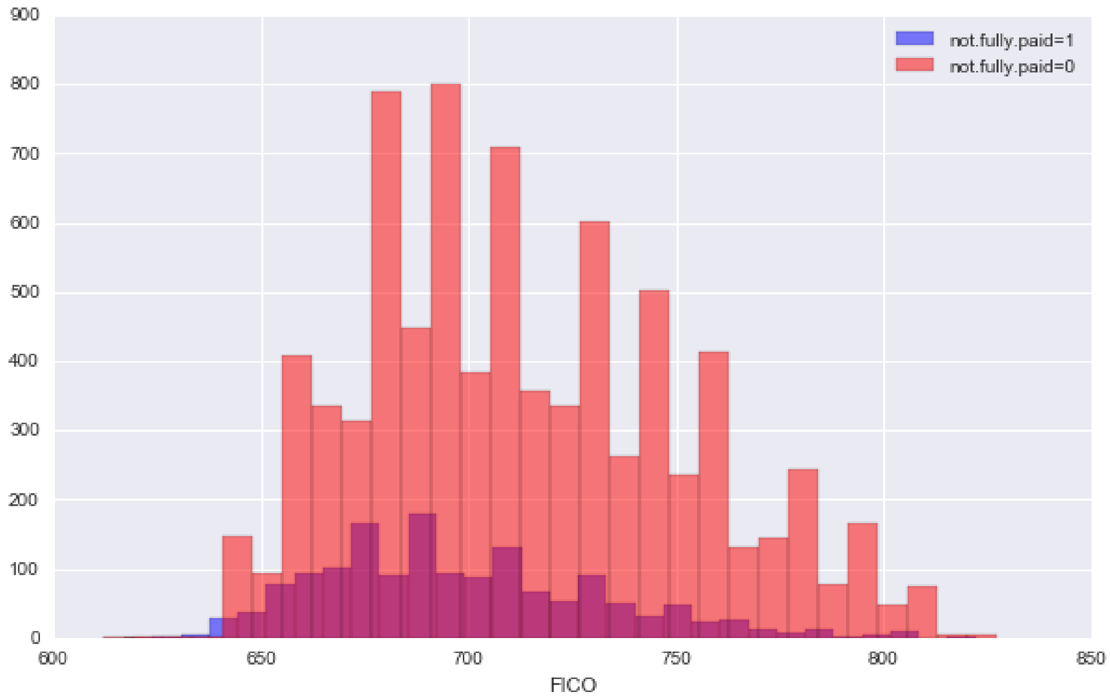
Out[6]: <matplotlib.text.Text at 0x119963f28>



** Create a similar figure, except this time select by the not.fully.paid column.**

In [7]:
```
plt.figure(figsize=(10,6))
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='blue',
                                               bins=30,label='not.fully.paid=1')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='red',
                                               bins=30,label='not.fully.paid=0')
plt.legend()
plt.xlabel('FICO')
```

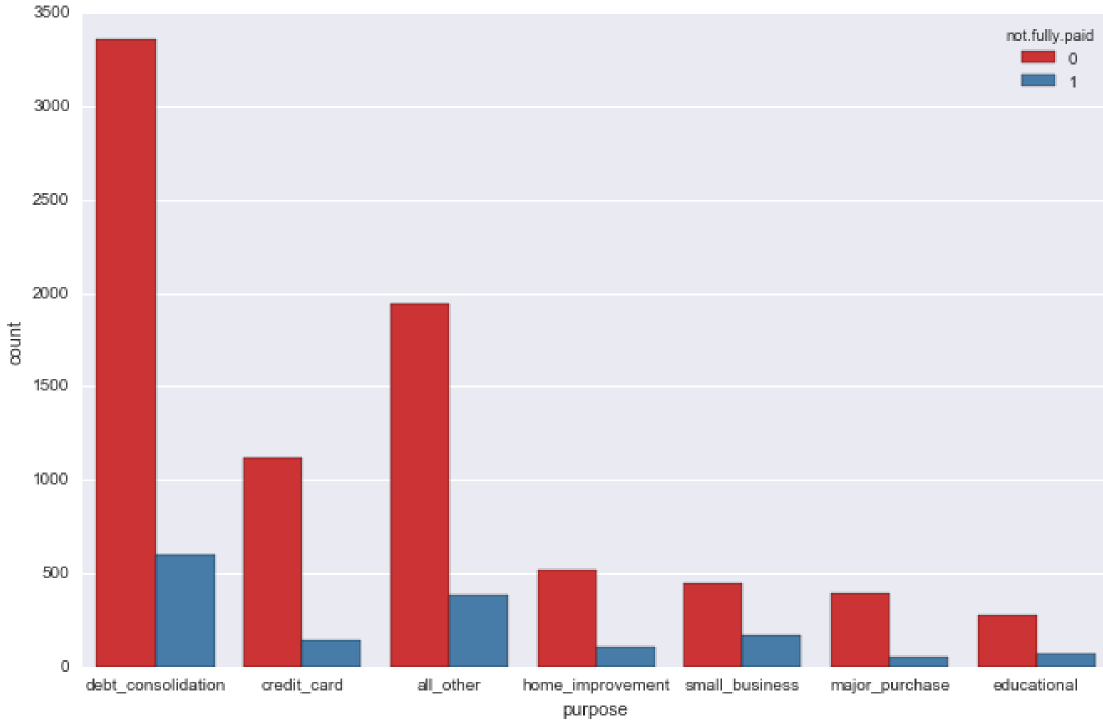
Out[7]: <matplotlib.text.Text at 0x11c47a7f0>



** Create a countplot using seaborn showing the counts of loans by purpose, with the color hue defined by not.fully.paid. **

In [8]:
```
plt.figure(figsize=(11,7))
sns.countplot(x='purpose',hue='not.fully.paid',data=loans,palette='Set1')
```

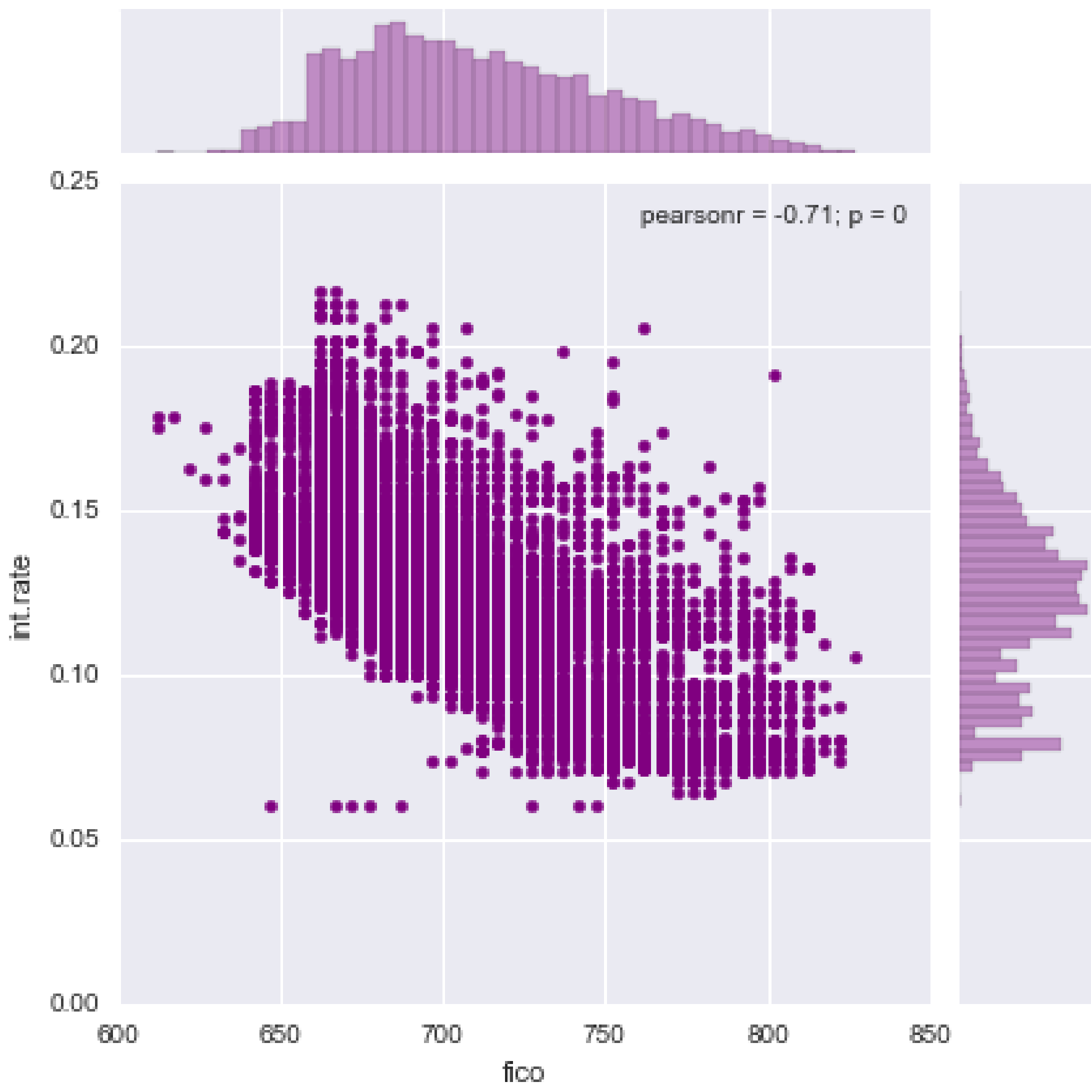
Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x119996828>
```



** Let's see the trend between FICO score and interest rate. Recreate the following jointplot.**

```
In [9]: sns.jointplot(x='fico',y='int.rate',data=loans,color='purple')
```

```
Out[9]: <seaborn.axisgrid.JointGrid at 0x119963320>
```



** Create the following lmplots to see if the trend differed between not.fully.paid and credit.policy. Check the documentation for lmplot() if you can't figure out how to separate it into columns.**

```
In [10]: plt.figure(figsize=(11,7))
         sns.lmplot(y='int.rate',x='fico',data=loans,hue='credit.policy',
                    col='not.fully.paid',palette='Set1')

Out[10]: <seaborn.axisgrid.FacetGrid at 0x11d34b668>

<matplotlib.figure.Figure at 0x11d3094e0>
```

4 Setting up the Data

Let's get ready to set up our data for our Random Forest Classification Model!

Check loans.info() again.

```
In [12]: loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
credit.policy        9578 non-null int64
purpose              9578 non-null object
int.rate             9578 non-null float64
installment          9578 non-null float64
log.annual.inc       9578 non-null float64
dti                  9578 non-null float64
fico                 9578 non-null int64
days.with.cr.line    9578 non-null float64
revol.bal            9578 non-null int64
revol.util           9578 non-null float64
inq.last.6mths       9578 non-null int64
delinq.2yrs          9578 non-null int64
pub.rec              9578 non-null int64
not.fully.paid       9578 non-null int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

4.1 Categorical Features

Notice that the **purpose** column as categorical

That means we need to transform them using dummy variables so sklearn will be able to understand them. Let's do this in one clean step using pd.get_dummies.

Let's show you a way of dealing with these columns that can be expanded to multiple categorical features if necessary.

Create a list of 1 element containing the string 'purpose'. Call this list cat_feats.

```
In [36]: cat_feats = ['purpose']
```

Now use pd.get_dummies(loans,columns=cat_feats,drop_first=True) to create a fixed larger dataframe that has new feature columns with dummy variables. Set this dataframe as final_data.

```
In [37]: final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True)
```

```
In [38]: final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
credit.policy                 9578 non-null int64
int.rate                      9578 non-null float64
installment                   9578 non-null float64
log.annual.inc                9578 non-null float64
dti                           9578 non-null float64
fico                          9578 non-null int64
days.with.cr.line             9578 non-null float64
revol.bal                     9578 non-null int64
revol.util                    9578 non-null float64
inq.last.6mths                9578 non-null int64
delinq.2yrs                   9578 non-null int64
pub.rec                       9578 non-null int64
not.fully.paid                9578 non-null int64
purpose_credit_card           9578 non-null float64
purpose_debt_consolidation    9578 non-null float64
purpose_educational           9578 non-null float64
purpose_home_improvement      9578 non-null float64
purpose_major_purchase        9578 non-null float64
purpose_small_business        9578 non-null float64
dtypes: float64(12), int64(7)
memory usage: 1.4 MB
```

4.2 Train Test Split

Now its time to split our data into a training set and a testing set!

** Use sklearn to split your data into a training set and a testing set as we've done in the past.**

```
In [20]: from sklearn.model_selection import train_test_split
```

```
In [21]: X = final_data.drop('not.fully.paid',axis=1)
         y = final_data['not.fully.paid']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state
```

4.3 Training a Decision Tree Model

Let's start by training a single decision tree first!
** Import DecisionTreeClassifier**

```
In [22]: from sklearn.tree import DecisionTreeClassifier
```

Create an instance of DecisionTreeClassifier() called dtree and fit it to the training data.

```
In [23]: dtree = DecisionTreeClassifier()
```

```
In [24]: dtree.fit(X_train,y_train)
```

```
Out[24]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     presort=False, random_state=None, splitter='best')
```

4.4 Predictions and Evaluation of Decision Tree

Create predictions from the test set and create a classification report and a confusion matrix.

```
In [25]: predictions = dtree.predict(X_test)
```

```
In [26]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [27]: print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.85      0.82      0.84      2431
          1       0.19      0.23      0.20       443

avg / total       0.75      0.73      0.74      2874
```

```
In [28]: print(confusion_matrix(y_test,predictions))
```

```
[[1995  436]
 [ 343  100]]
```

4.5 Training the Random Forest model

Now its time to train our model!

Create an instance of the RandomForestClassifier class and fit it to our training data from the previous step.

```
In [29]: from sklearn.ensemble import RandomForestClassifier
```

```
In [30]: rfc = RandomForestClassifier(n_estimators=600)
```

```
In [31]: rfc.fit(X_train,y_train)
```

```
Out[31]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=600, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

4.6 Predictions and Evaluation

Let's predict off the y_test values and evaluate our model.

** Predict the class of not.fully.paid for the X_test data.**

```
In [32]: predictions = rfc.predict(X_test)
```

Now create a classification report from the results. Do you get anything strange or some sort of warning?

```
In [33]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [34]: print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.85      1.00      0.92      2431
          1       0.57      0.03      0.05       443

avg / total       0.81      0.85      0.78      2874
```

Show the Confusion Matrix for the predictions.

```
In [35]: print(confusion_matrix(y_test,predictions))
```

```
[[2422    9]
 [ 431   12]]
```

What performed better the random forest or the decision tree?

```
In [36]: # Depends what metric we are trying to optimize for.
         # Notice the recall for each class for the models.
         # Neither did very well, more feature engineering is needed.
```

5 Great Job!