Decision Trees & Random-Forests using Lending Club Data ¶

Brief

> For this project we will be exploring publicly available data from [www.LendingClub.com (http://www.LendingClub.com)](http://www.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, this data is from them before they even went public. The lending data is from 2007-2010 and trying to classify and predict whether or not the borrower paid back their loan in full.

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).

In [4]:
```
import numpy as np
import pandas as pd
```

In [5]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [6]:
```
loans= pd.read_csv('loan_data.csv')
```

In [17]:
```
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 [7]:
```
loans.head(10)
```

Out[7]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1
5	1	credit_card	0.0788	125.13	11.904968	16.98	727	6120.041667	50807	51.0	0	0
6	1	debt_consolidation	0.1496	194.02	10.714418	4.00	667	3180.041667	3839	76.8	0	0
7	1	all_other	0.1114	131.22	11.002100	11.08	722	5116.000000	24220	68.6	0	0
8	1	home_improvement	0.1134	87.19	11.407565	17.25	682	3989.000000	69909	51.1	1	0
9	1	debt_consolidation	0.1221	84.12	10.203592	10.00	707	2730.041667	5630	23.0	1	0

In [10]:
```
loans.describe()
```

Out[10]:

nt.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	r
00000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	9578.000000	9578.000000	9578.000000	9578.000000	
22640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	46.799236	1.577469	0.163708	0.062122	
26847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	29.014417	2.200245	0.546215	0.262126	
60000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.000000	0.000000	0.000000	0.000000	
03900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	22.600000	0.000000	0.000000	0.000000	
22100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	46.300000	1.000000	0.000000	0.000000	
40700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	70.900000	2.000000	0.000000	0.000000	
16400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.000000	33.000000	13.000000	5.000000	

Exploratory Data Analysis¶ (EDA)

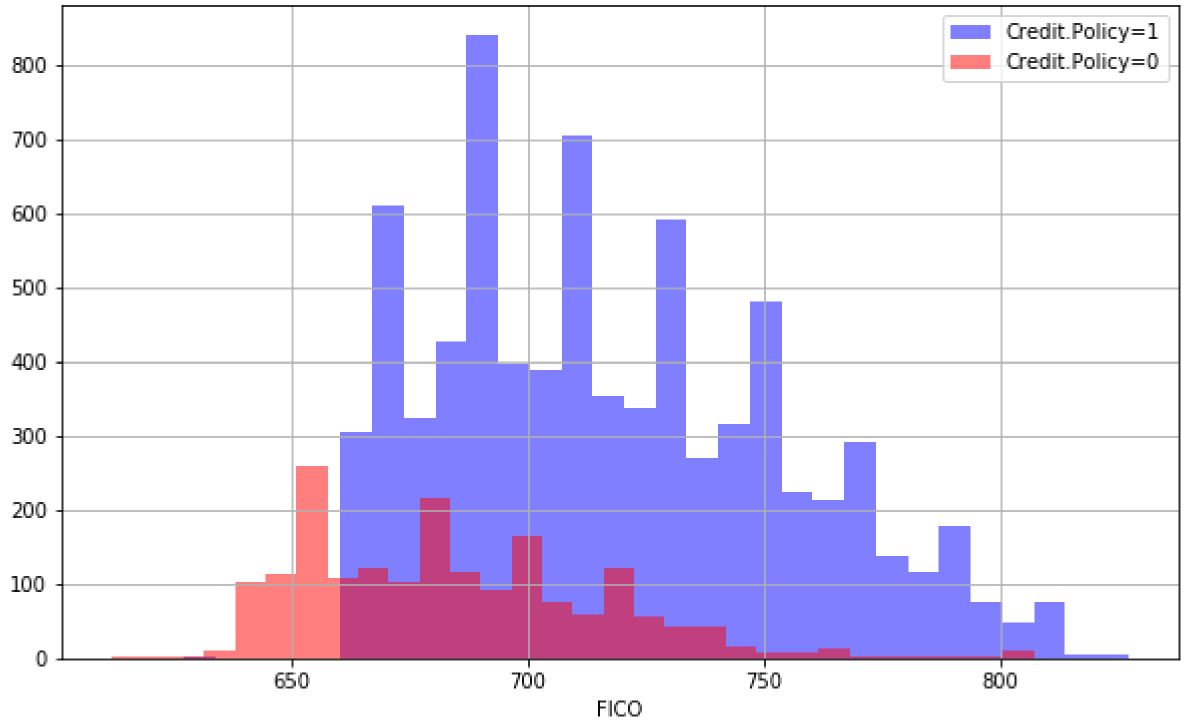
- CreateING a histogram of two FICO distributions on top of each other, one for each credit.policy outcome.

In [11]:

```
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')
```

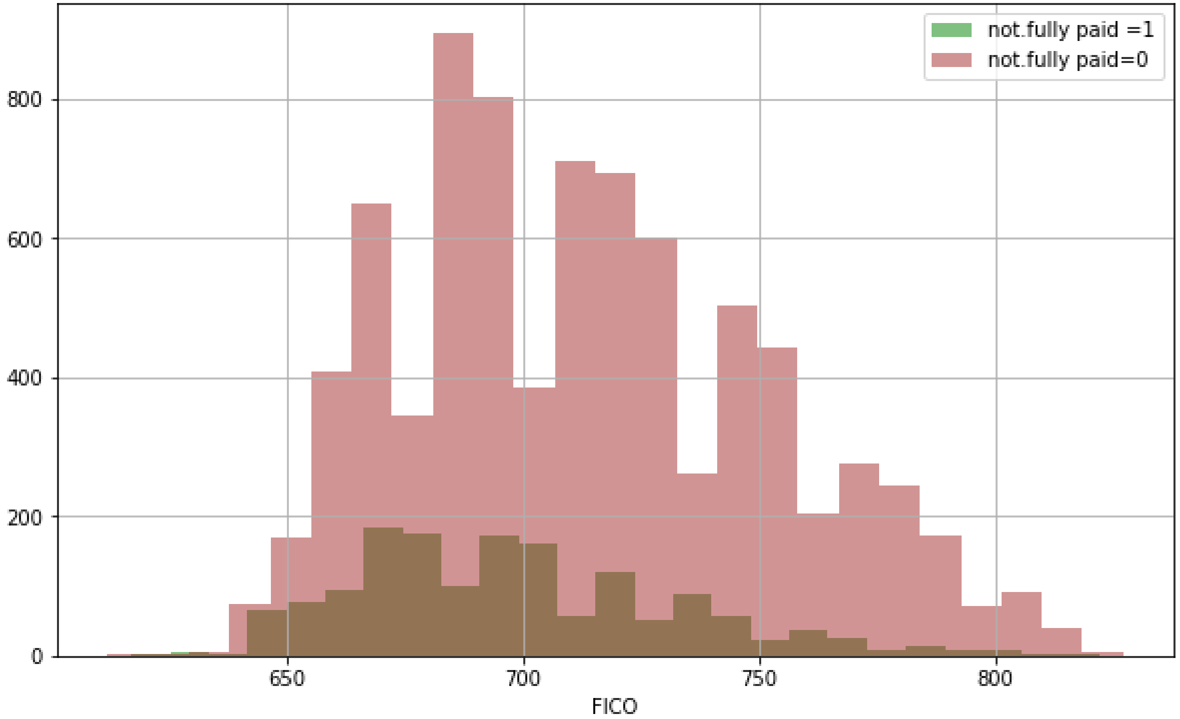
Out[11]:

```
Text(0.5, 0, 'FICO')
```



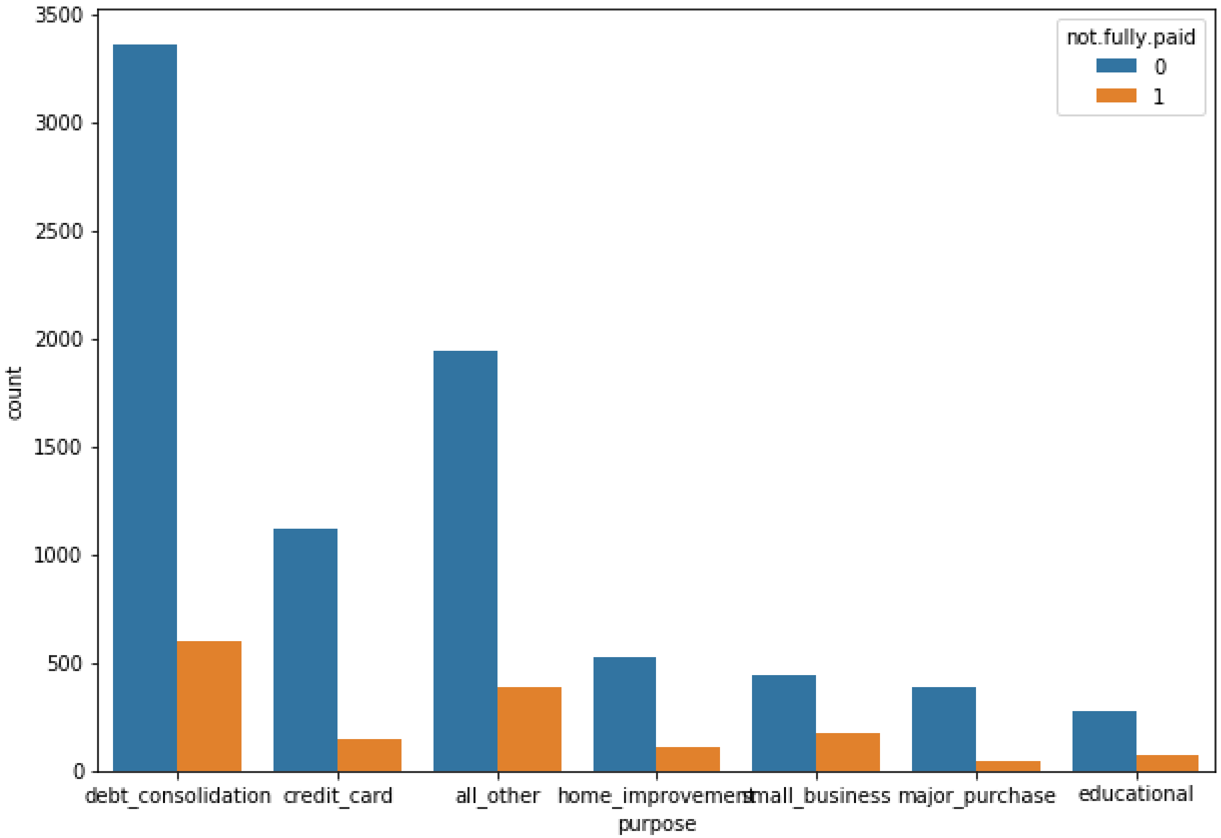
In [8]:
```
plt.figure(figsize=(10,6))
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='green',
                                               bins=25,label='not.fully paid =1')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='brown',
                                               bins=25,label='not.fully paid=0')
plt.legend()
plt.xlabel('FICO')
```

Out[8]:
```
Text(0.5, 0, 'FICO')
```



In [15]:
```
plt.figure(figsize=(10,7))
sns.countplot(x='purpose', hue='not.fully.paid',data=loans,)
```

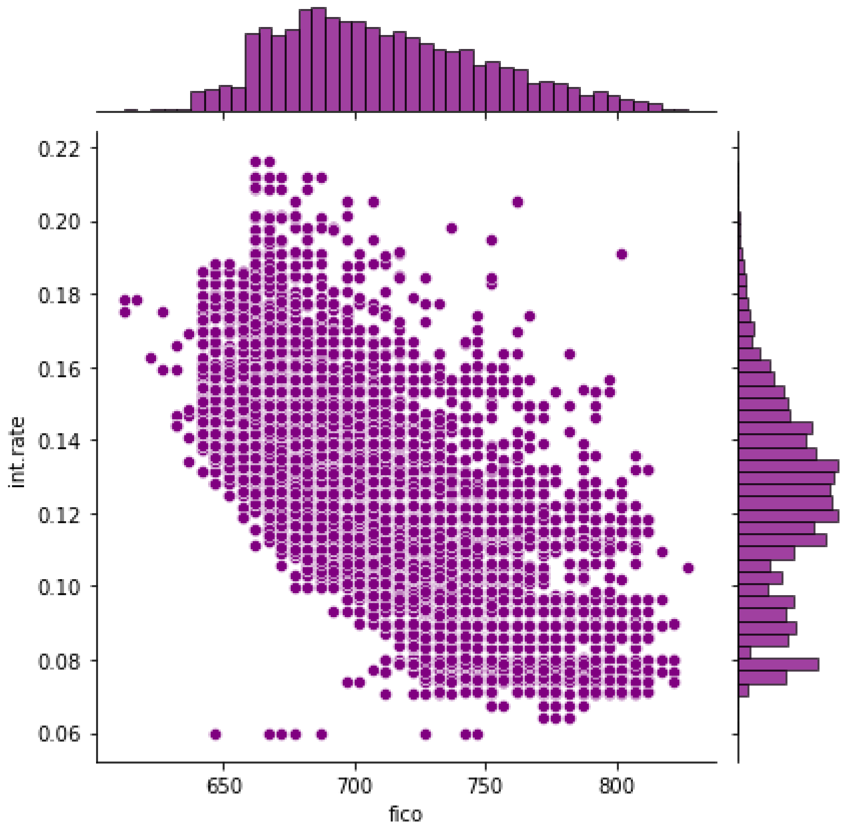
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1e5463b2588>



In [11]:
```
plt.figure(figsize=(10,6))
sns.jointplot(x='fico',y='int.rate',data=loans, color='purple')
```

Out[11]:
```
<seaborn.axisgrid.JointGrid at 0x25017445070>
```

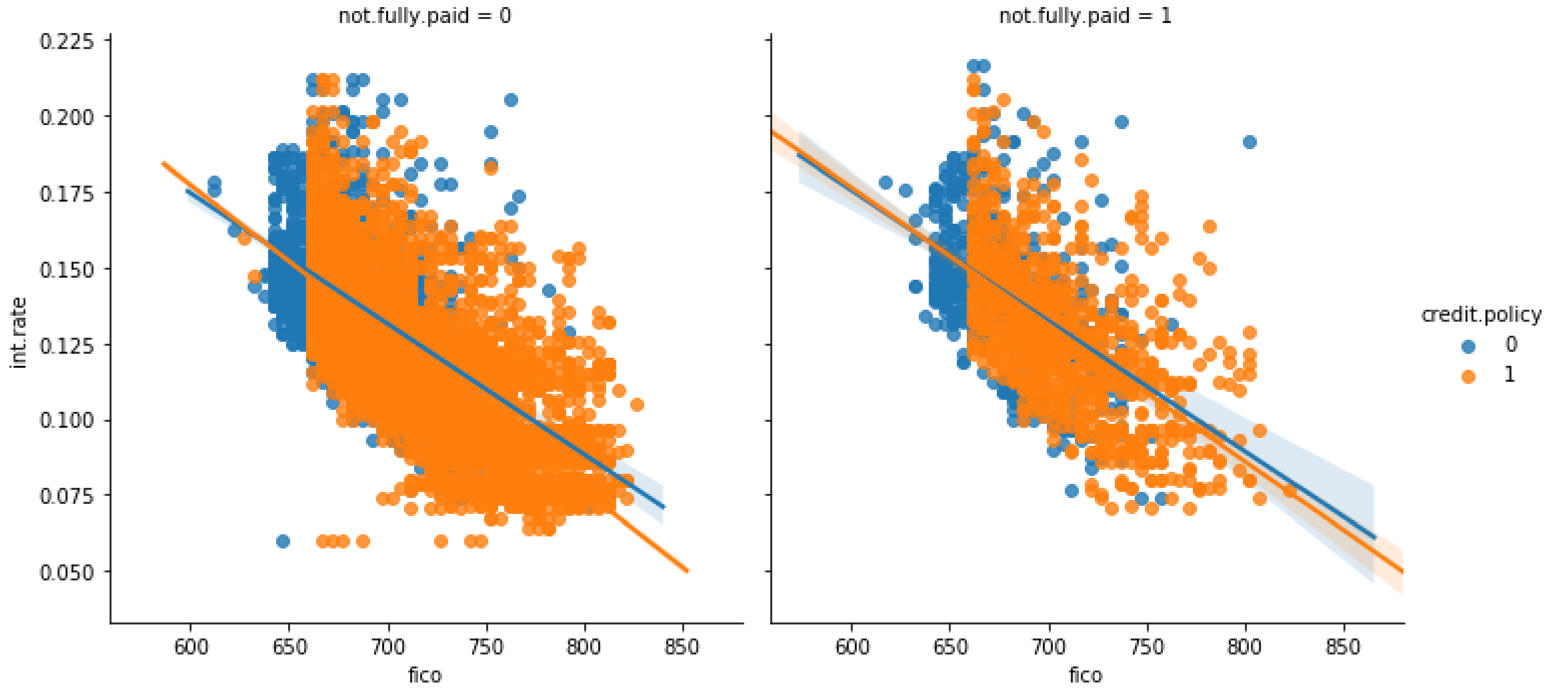
```
<Figure size 720x432 with 0 Axes>
```



In [19]:
```
plt.figure(figsize=(10,7))
sns.lmplot(y='int.rate',x='fico', data=loans,hue='credit.policy', col='not.fully.paid')
```

Out[19]:
```
<seaborn.axisgrid.FacetGrid at 0x256aad58a88>
```

```
<Figure size 720x504 with 0 Axes>
```



Setting up the Data

Random Forest Classification Model!

The loans data has an object columms will use pd.get_ dummies to make it an int

In [113]:
```
cat_feats=['purpose']
```

In [114]:
```
final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True )
```

In [29]:
```
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 uint8
purpose_debt_consolidation    9578 non-null uint8
purpose_educational           9578 non-null uint8
purpose_home_improvement      9578 non-null uint8
purpose_major_purchase        9578 non-null uint8
purpose_small_business        9578 non-null uint8
dtypes: float64(6), int64(7), uint8(6)
memory usage: 1.0 MB
```

In [28]:
```
final_data.head()
```

Out[28]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

In [115]:
```
from sklearn.model_selection import train_test_split
```

In [116]:
```
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=0.3, random_state=101)
```

Training a Decision Tree Model¶

Let's start by Training a single decision tree first!

Import DecisionTreeClassifier

In [117]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [118]:
```
dtree=DecisionTreeClassifier ()
```

In [119]:
```
dtree.fit(x_train,y_train)
```

Out[119]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

Predictions and Evaluation of Decision Tree

Creating a classification report and a confusion matrix

In [140]:
```
predictions = dtree.predict(x_test)
```

In [141]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [142]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.84      2431
           1       0.19      0.23      0.21       443

    accuracy                           0.73      2874
   macro avg       0.52      0.53      0.52      2874
weighted avg       0.75      0.73      0.74      2874
```

In [146]:
```
print (confusion_matrix(y_test,predictions))
```

```
[[1996  435]
 [ 340  103]]
```

Training the Random Forest model

train our model!

RandomForestClassifier class and fit it to our training data

In [147]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [149]:
```
rfc = RandomForestClassifier(n_estimators=600)
```

In [151]:
```
rfc.fit(x_train,y_train)
```

Out[151]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=600,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Predictions and Evaluation

predicting off the y_test values and evaluate the model

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

In [152]:
```
prediction =rfc.predict(x_test)
```

In [154]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [155]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.84      2431
           1       0.19      0.23      0.21       443

    accuracy                           0.73      2874
   macro avg       0.52      0.53      0.52      2874
weighted avg       0.75      0.73      0.74      2874
```

In [156]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[1996  435]
 [ 340  103]]
```

Conclusion: from the model above, it shows people are paying back their loans after all Criteria of lending Club are met