Random Forest Project

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). As an investor you would want to invest in people who showed a profile of having a high probability of paying back their loan. The purpose of this project is to create a model that will help predict this.

We will use lending data from 2007-2010 and be trying to classify and predict whether or not the borrower paid back his/her loan in full. Download the data from [https://www.lendingclub.com/info/download-data.action (https://www.lendingclub.com/info/download-data.action)](https://www.lendingclub.com/info/download-data.action)

See csv file in this project as it has been cleaned of NA values.

The columns represent the following:

- 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).

Import Libraries and get the Data

In [32]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
loans = pd.read_csv('loan_data.csv')
```

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

In [33]:
```
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 [34]:
```
loans.describe()
```

Out[34]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.wit
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639

In [35]:
```
loans.head()
```

Out[35]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.lin
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.95833
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.00000
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.00000
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.95833
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.00000

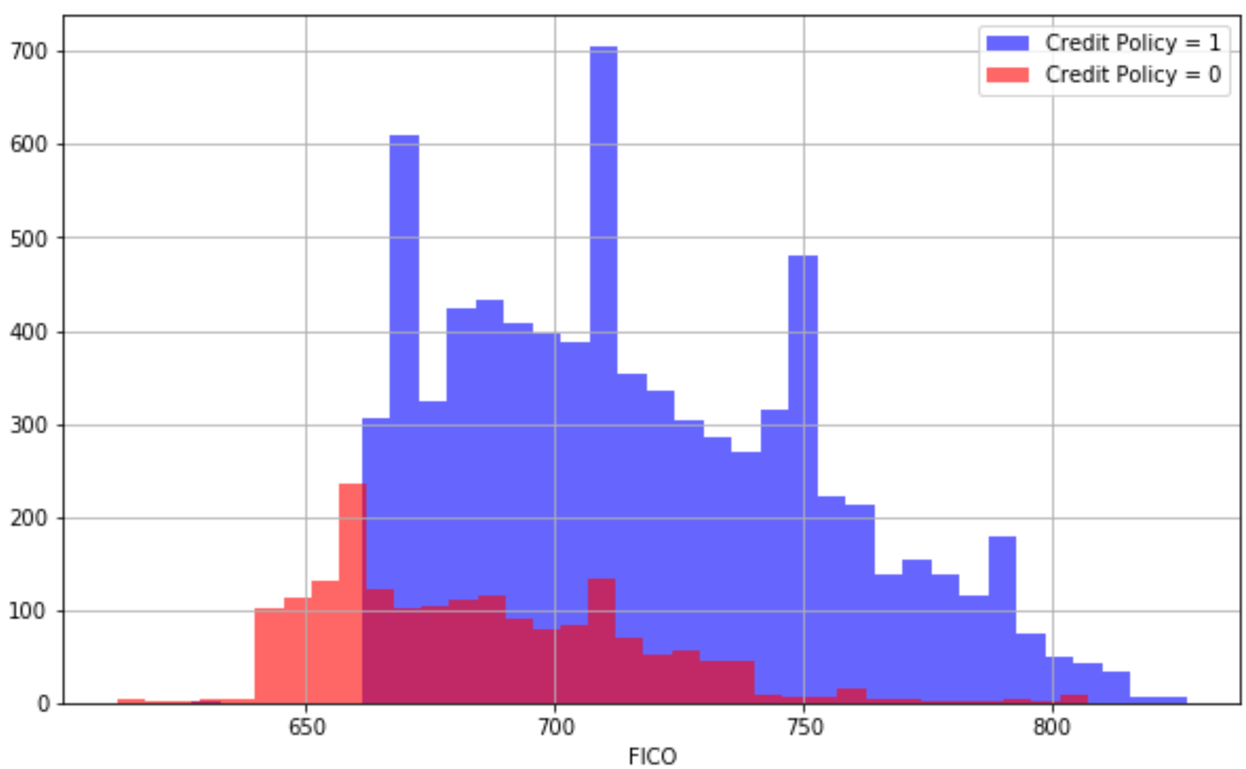
Exploratory Data Analysis

For Data Visualization, we will use seaborn and pandas built-in plotting capabilities.

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

In [36]:
```
plt.figure(figsize = (10, 6))
loans[loans['credit.policy'] == 1]['fico'].hist(bins = 35,
                                                color = 'blue',
                                                label = 'Credit Policy = 1
                                                alpha = 0.6)
loans[loans['credit.policy'] == 0]['fico'].hist(bins = 35,
                                                color = 'red',
                                                label = 'Credit Policy = 0
                                                alpha = 0.6)
plt.xlabel('FICO')
plt.legend()
```

Out[36]:
```
<matplotlib.legend.Legend at 0x1a238283c8>
```

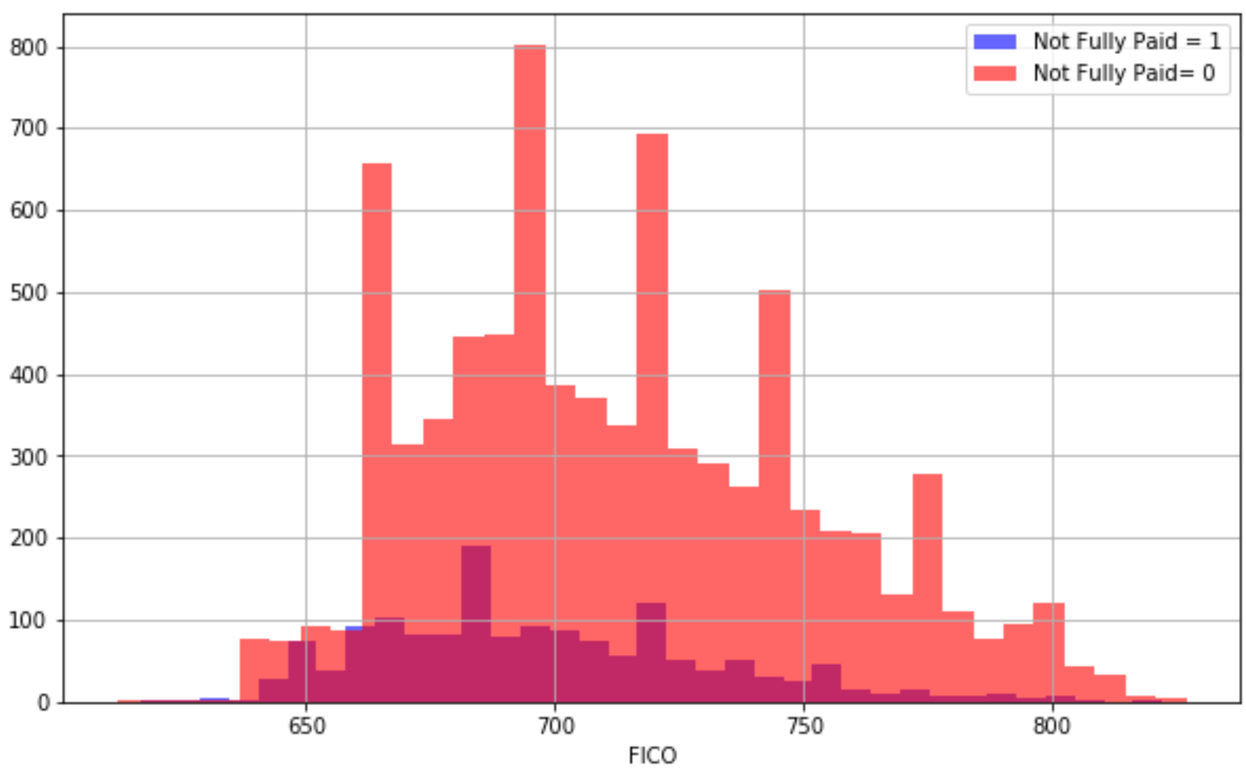


Create a similar figure, except this time select by the not.fully.paid column.

In [37]:

```
plt.figure(figsize = (10, 6))
loans[loans['not.fully.paid'] == 1]['fico'].hist(bins = 35,
                                                 color = 'blue',
                                                 label = 'Not Fully Paid =
                                                 alpha = 0.6)
loans[loans['not.fully.paid'] == 0]['fico'].hist(bins = 35,
                                                 color = 'red',
                                                 label = 'Not Fully Paid=
                                                 alpha = 0.6)
plt.xlabel('FICO')
plt.legend()
```

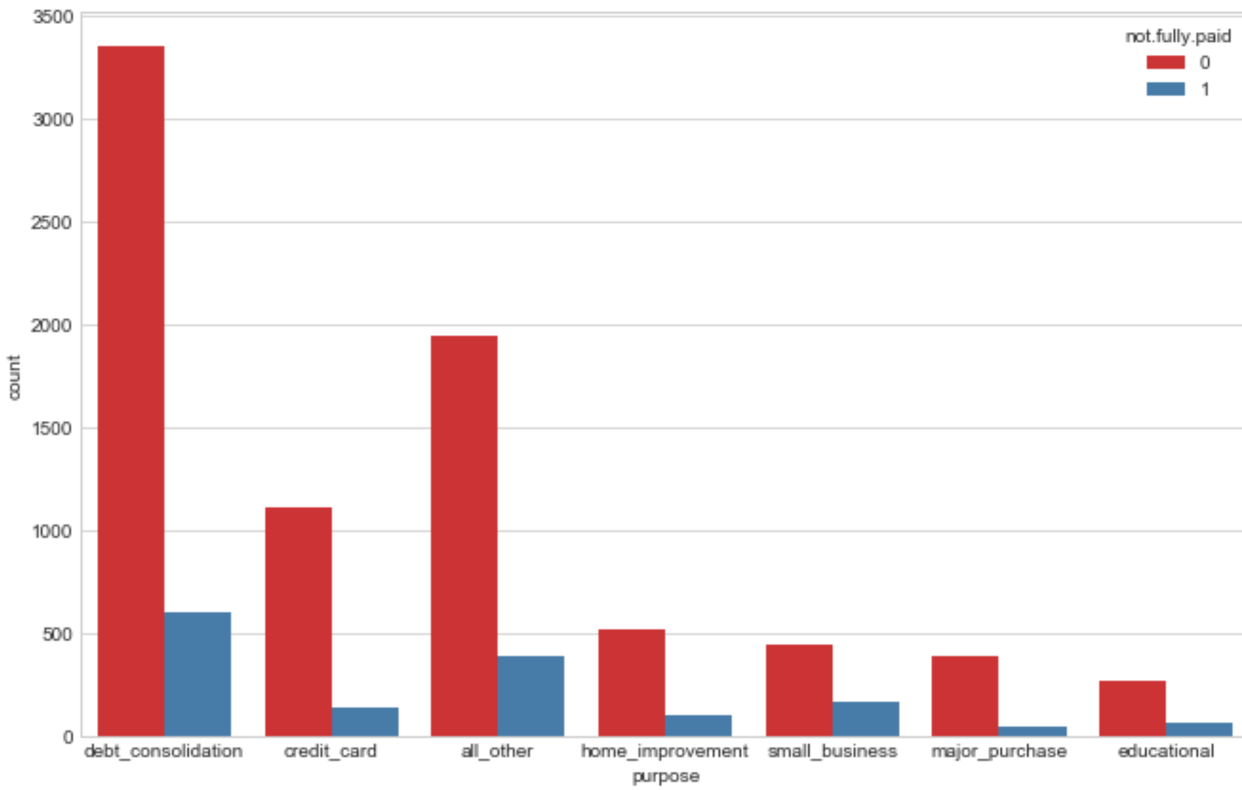
Out[37]: <matplotlib.legend.Legend at 0x1a23ce6fd0>



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

In [38]:
```
sns.set_style('whitegrid')
fig = plt.figure(figsize = (11, 7))
sns.countplot(data=loans, x='purpose', hue='not.fully.paid',
                 palette = 'Set1' )
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1a224a2518>

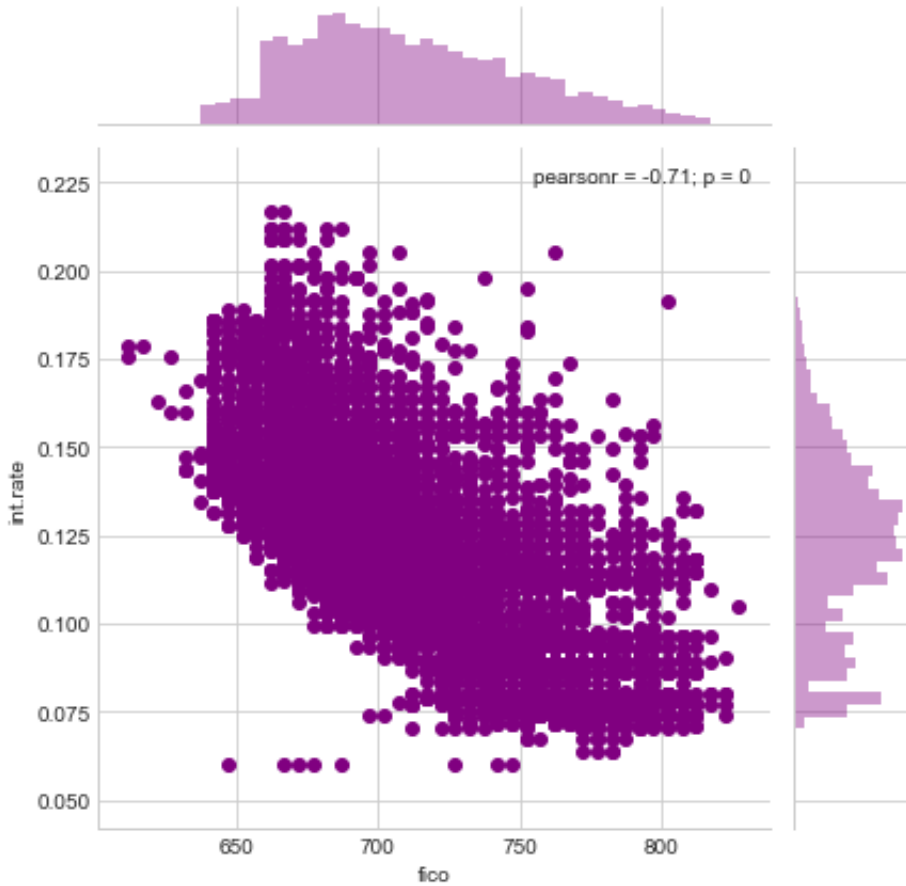


Let us see the trend between FICO score and interest rate by creating a jointplot.

In [39]:
```
sns.jointplot(data = loans, x = 'fico', y='int.rate', color='purple')
```

```
/Users/Jayashri/anaconda/lib/python3.6/site-packages/scipy/stats/stats
.py:1713: FutureWarning: Using a non-tuple sequence for multidimension
al indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`
. In the future this will be interpreted as an array index, `arr[np.ar
ray(seq)]`, which will result either in an error or a different result
.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[39]: <seaborn.axisgrid.JointGrid at 0x10dd69c88>



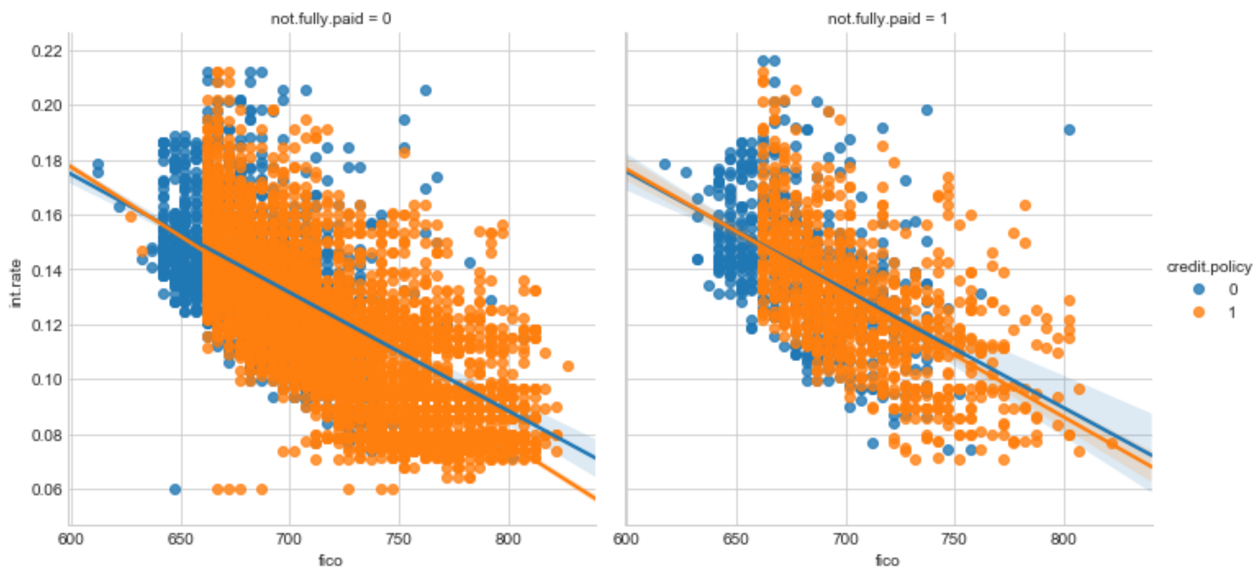
Create the following lmplots to see if the trend differed between not.fully.paid and credit.policy. We will separate them into columns

In [40]:
```python
plt.figure(figsize = (11, 7))
sns.lmplot(data=loans, x='fico', y='int.rate',
           col = 'not.fully.paid', hue = 'credit.policy')
```

```
/Users/Jayashri/anaconda/lib/python3.6/site-packages/scipy/stats/stats
.py:1713: FutureWarning: Using a non-tuple sequence for multidimension
al indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`
. In the future this will be interpreted as an array index, `arr[np.ar
ray(seq)]`, which will result either in an error or a different result
.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[40]:
```
<seaborn.axisgrid.FacetGrid at 0x1a246559b0>

<matplotlib.figure.Figure at 0x1a242f3400>
```



Setting up the Data

We will set up the data for Random Forest Classification Model

Check loans.info() again.

In [41]:
```
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
```

Categorical Features

Observation: The **purpose** column is a categorical column.

So we need to transform it using dummy variables using pd.get_dummies for using it in our quantitative analysis.

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

In [42]:
```
cat_feats = ['purpose']
cat_feats
```

Out[42]: ['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. Note that the first column is dropped to avoid problems with multicollinearity.

In [43]:
```
final_data = pd.get_dummies(loans, columns=cat_feats,
                            drop_first = True)
final_data.head()
```

Out[43]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.u
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39

In [44]:
```
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
```

Train Test Split

We will split the data into a training set and a testing set using sklearn and use test_size of 30%.

In [45]:
```
from sklearn.model_selection import train_test_split
```

In [46]:
```
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 = 10
```

Training a Decision Tree Model

We will start by training a single decision tree first.

Import DecisionTreeClassifier

In [47]:
```
from sklearn.tree import DecisionTreeClassifier
```

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

In [48]:
```
dtree = DecisionTreeClassifier()
```

In [49]:
```
dtree.fit(X_train, y_train)
```

Out[49]:
```
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

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

In [50]:
```
predictions = dtree.predict(X_test)
```

In [51]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [52]:
```
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

avg / total       0.75      0.73      0.74      2874
```

In [53]:
```
print(confusion_matrix(y_test, predictions))
```

```
[[1997  434]
 [ 341  102]]
```

Training the Random Forest model

Now we will train the Random Forest Model

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

In [54]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [55]:
```
rfc = RandomForestClassifier(n_estimators=300)
```

In [61]:
```
rfc
```

Out[61]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            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=300, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [62]:
```
rfc.fit(X_train, y_train)
```

Out[62]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            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=300, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Predictions and Evaluation

Let us predict the y_test values and evaluate our model.

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

In [63]:
```
rfc_pred = rfc.predict(X_test)
```

Show the Classification Report

In [66]:
```
print(classification_report(y_test, rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.85      1.00      0.92      2431
          1       0.53      0.02      0.03       443

avg / total       0.80      0.85      0.78      2874
```

Show the Confusion Matrix for the predictions.

In [65]:
```
print(confusion_matrix(y_test, rfc_pred))
```

```
[[2424    7]
 [ 435    8]]
```

Observation: The Random Forest Model performed better that the Decision Tree Model.