# Random Forest Classification

For this application 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 he would want to invest in people who showed a profile of having a high probability of paying hime back. We will try 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 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).

## Import required Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
%matplotlib inline
```

## Get the Data

## Load loan_data.csv data into loans dataframe

In [3]:
```
l = pd.read_csv("loan_data.csv")
```

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

In [4]:
```
l.head()
```

Out[4]:

| | credit.policy | purpose | 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 | debt_consolidation | 0.1189 | 829.10 | 11.350407 | 19.48 | 737 | 5639.958333 | 28854 | 52.1 | 0 | 0 | 0 | 0 |
| **1** | 1 | credit_card | 0.1071 | 228.22 | 11.082143 | 14.29 | 707 | 2760.000000 | 33623 | 76.7 | 0 | 0 | 0 | 0 |
| **2** | 1 | debt_consolidation | 0.1357 | 366.86 | 10.373491 | 11.63 | 682 | 4710.000000 | 3511 | 25.6 | 1 | 0 | 0 | 0 |
| **3** | 1 | debt_consolidation | 0.1008 | 162.34 | 11.350407 | 8.10 | 712 | 2699.958333 | 33667 | 73.2 | 1 | 0 | 0 | 0 |
| **4** | 1 | credit_card | 0.1426 | 102.92 | 11.299732 | 14.97 | 667 | 4066.000000 | 4740 | 39.5 | 0 | 1 | 0 | 0 |

In [5]:
```
l.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 [6]:
```
l.describe()
```

Out[6]:

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 | 9.578000e+03 | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 |
| **mean** | 0.804970 | 0.122640 | 319.089413 | 10.932117 | 12.606679 | 710.846314 | 4560.767197 | 1.691396e+04 | 46.799236 | 1.577469 | 0.163708 | 0.062122 | 0.160054 |
| **std** | 0.396245 | 0.026847 | 207.071301 | 0.614813 | 6.883970 | 37.970537 | 2496.930377 | 3.375619e+04 | 29.014417 | 2.200245 | 0.546215 | 0.262126 | 0.366676 |
| **min** | 0.000000 | 0.060000 | 15.670000 | 7.547502 | 0.000000 | 612.000000 | 178.958333 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 1.000000 | 0.103900 | 163.770000 | 10.558414 | 7.212500 | 682.000000 | 2820.000000 | 3.187000e+03 | 22.600000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **50%** | 1.000000 | 0.122100 | 268.950000 | 10.928884 | 12.665000 | 707.000000 | 4139.958333 | 8.596000e+03 | 46.300000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| **75%** | 1.000000 | 0.140700 | 432.762500 | 11.291293 | 17.950000 | 737.000000 | 5730.000000 | 1.824950e+04 | 70.900000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 |
| **max** | 1.000000 | 0.216400 | 940.140000 | 14.528354 | 29.960000 | 827.000000 | 17639.958330 | 1.207359e+06 | 119.000000 | 33.000000 | 13.000000 | 5.000000 | 1.000000 |

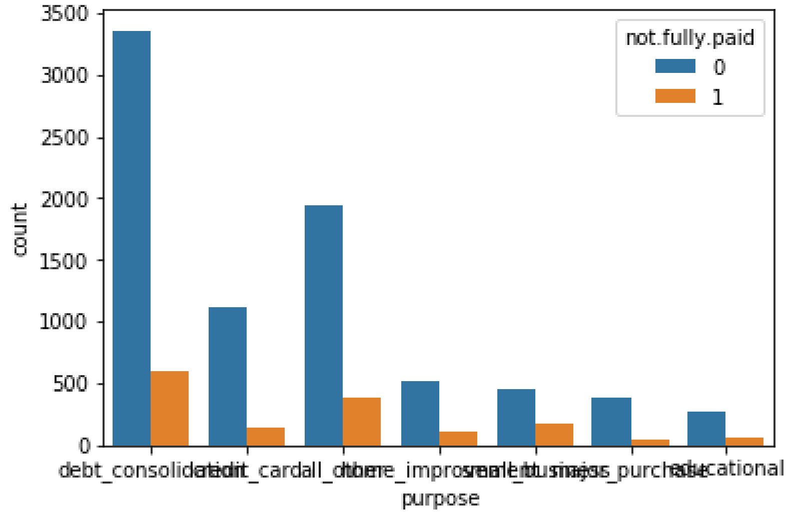
## Exploratory Data Analysis

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

sns.countplot(x = 'col-name1', hue = 'col-name2', data = dataframeName)
where
col-name1=purpose
col_name2=not.fully.paid
dataframename=loan

In [28]:
```
sns.countplot(x ='purpose', hue = "not.fully.paid", data = l)

# Show the plot
plt.show()
```




## Setting up Data

## Categorical Features Preparation

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. Do this in one clean step using pd.get_dummies.

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

1. Create a list of 1 element containing the string 'purpose'. Call this list cat_feats.
2. 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 [7]:
```
cat_feats = ['purpose']
```

In [9]:
```
final_data = pd.get_dummies(l, columns = cat_feats, drop_first = True)
```

## Now print head of final_data. Observe new columns have been added according to category values of purpose column

In [10]:
```
final_data.head()
```

Out[10]:

| | 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 | purpose_credit_card | purpose_debt_consolidation | purpose_educational |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.1189 | 829.10 | 11.350407 | 19.48 | 737 | 5639.958333 | 28854 | 52.1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **1** | 1 | 0.1071 | 228.22 | 11.082143 | 14.29 | 707 | 2760.000000 | 33623 | 76.7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **2** | 1 | 0.1357 | 366.86 | 10.373491 | 11.63 | 682 | 4710.000000 | 3511 | 25.6 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| **3** | 1 | 0.1008 | 162.34 | 11.350407 | 8.10 | 712 | 2699.958333 | 33667 | 73.2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| **4** | 1 | 0.1426 | 102.92 | 11.299732 | 14.97 | 667 | 4066.000000 | 4740 | 39.5 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

## Print the head of only new columns added by get_dummies() from final_data

In [ ]:

## Train Test Split

## Define X input features and y output feature

Drop column 'not.fully.paid' and assign to X
Assign the column 'not.fully.paid' to y

In [21]:
```
X = final_data.drop('not.fully.paid',axis=1)
y = final_data['not.fully.paid']
```

## Split the Dataset into X_train, X_test, y_train, y_test using train_test_split(X, y, test_size = 0.3 , random_state= 101)

In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=101)
```

## Training the Random Forest model

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

In [23]:
```
rfc = RandomForestClassifier(n_estimators=600)
```

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

Out[24]:
```
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=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

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

In [25]:
```
y_pred = rfc.predict(X_test)
```

## Show the Confusion Matrix for the predictions.

In [26]:
```
print(confusion_matrix(y_test, y_pred))
```

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

## Print the classification report

In [27]:
```
print(classification_report(y_test, y_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
```