Loan Default Project

For this project we will be exploring publicly available data from LendingClub.com (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 (https://en.wikipedia.org/wiki/Lending_Club#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 (https://www.lendingclub.com/info/download-data.action) 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).

Import Libraries

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

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns %matplotlib inline

In [2]:
```
plt.rcParams['patch.force_edgecolor']=True
```

Get the Data

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

In [3]:
```
df = pd.read_csv('loan_data.csv')
```

In [4]:
```
df.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
```

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

In [5]:
```
df.describe()
```

Out[5]:

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

In [6]:
```
df.head()
```

Out[6]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.v
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.9
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.9
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.0

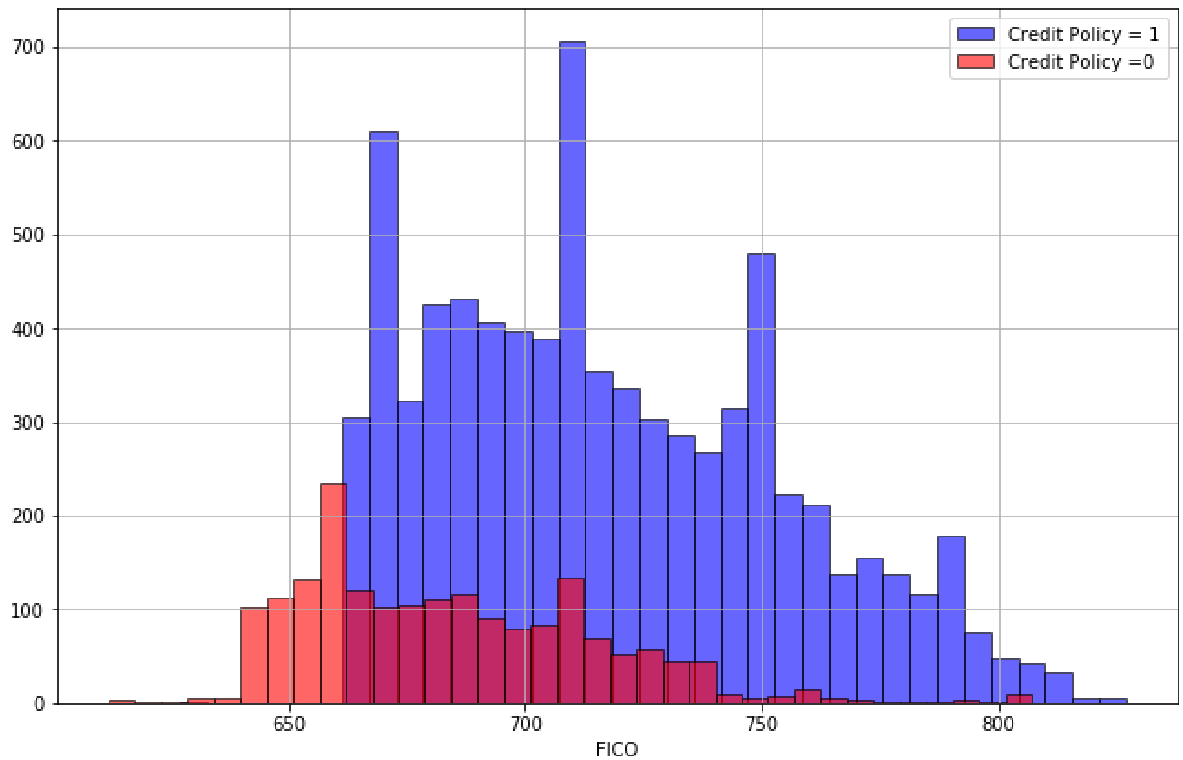
Exploratory Data Analysis

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.

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

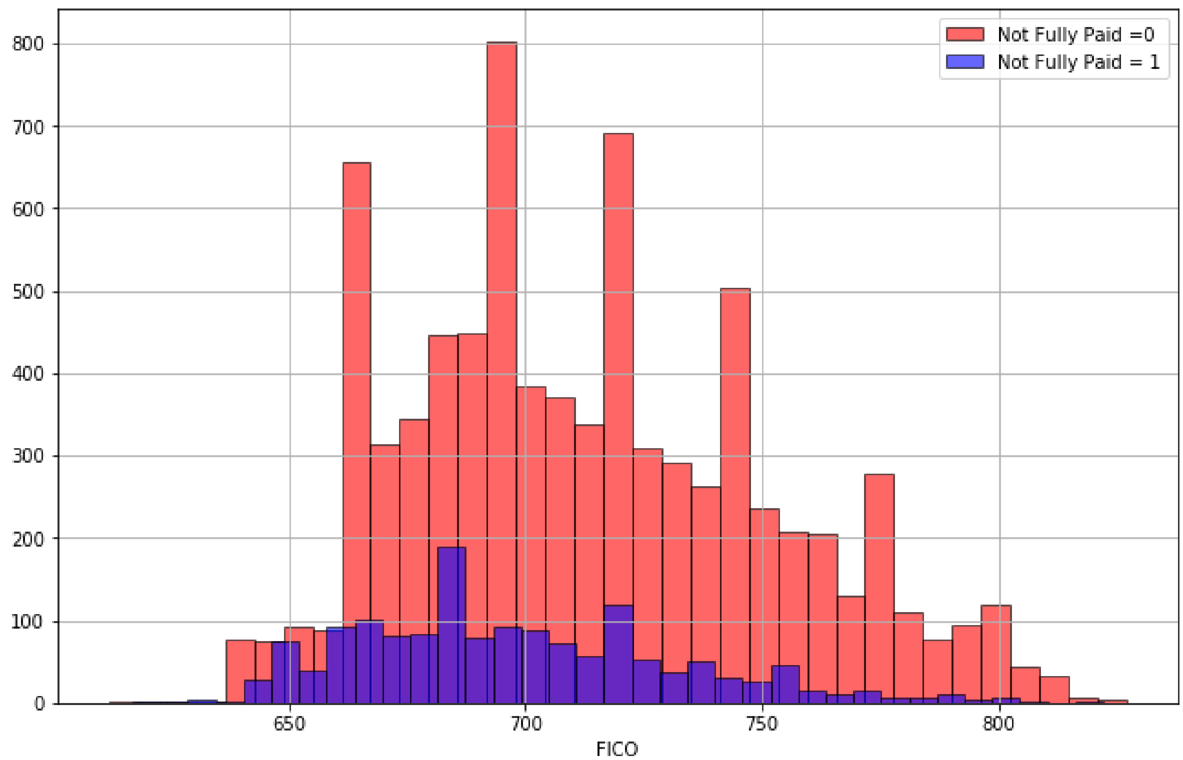
In [7]:
```
plt.figure(figsize=(11,7))
df[df['credit.policy']==1]['fico'].hist(color='blue',
                      alpha=0.6, bins=35, label='Credit Policy = 1')

df[df['credit.policy']==0]['fico'].hist(color='red',
                      alpha=0.6, bins=35,label='Credit Policy =0')
plt.xlabel('FICO')
plt.legend()
```

Out[7]: <matplotlib.legend.Legend at 0xbc0cac8>



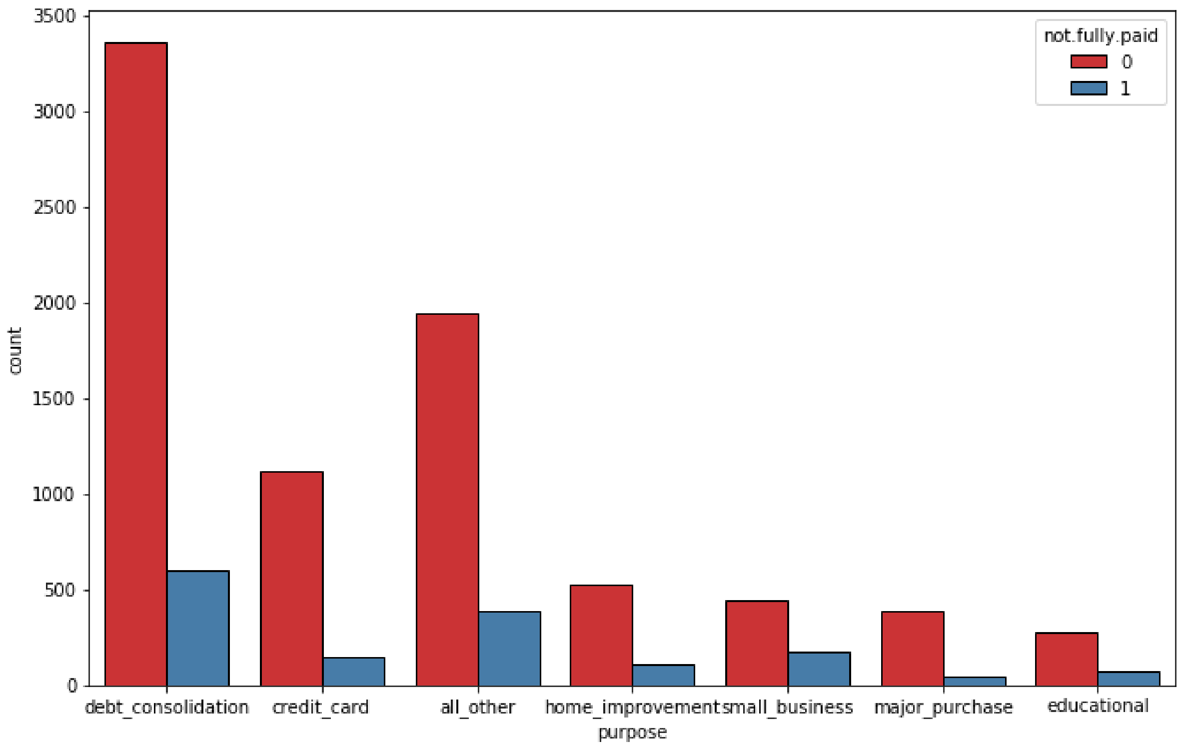
In [8]:
```
plt.figure(figsize=(11,7))
df[df['not.fully.paid']==0]['fico'].hist(color='red',
                    alpha=0.6, bins=35,label='Not Fully Paid =0')

df[df['not.fully.paid']==1]['fico'].hist(color='blue',
                    alpha=0.6, bins=35, label='Not Fully Paid = 1')
plt.xlabel('FICO')
plt.legend()
```

Out[8]: <matplotlib.legend.Legend at 0xbdf0cf8>



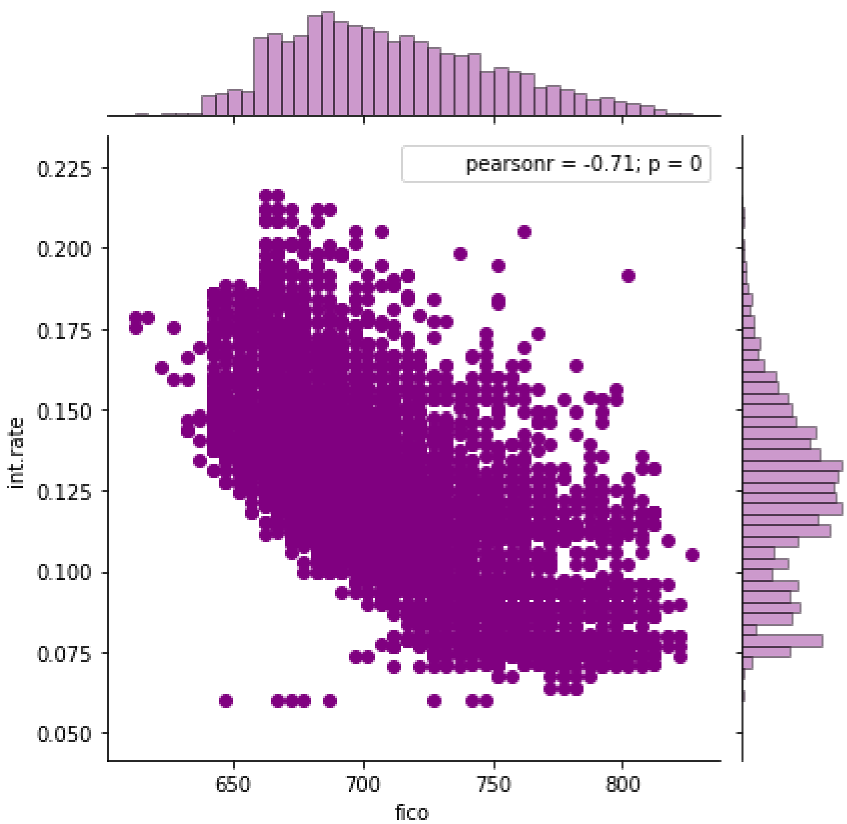
In [9]:
```
plt.figure(figsize=(11,7))
sns.countplot(x='purpose', hue='not.fully.paid', data=df, palette='Set1')
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x6af6a20>



In [10]:
```
sns.jointplot(x='fico', y='int.rate', data=df, color='purple')
```

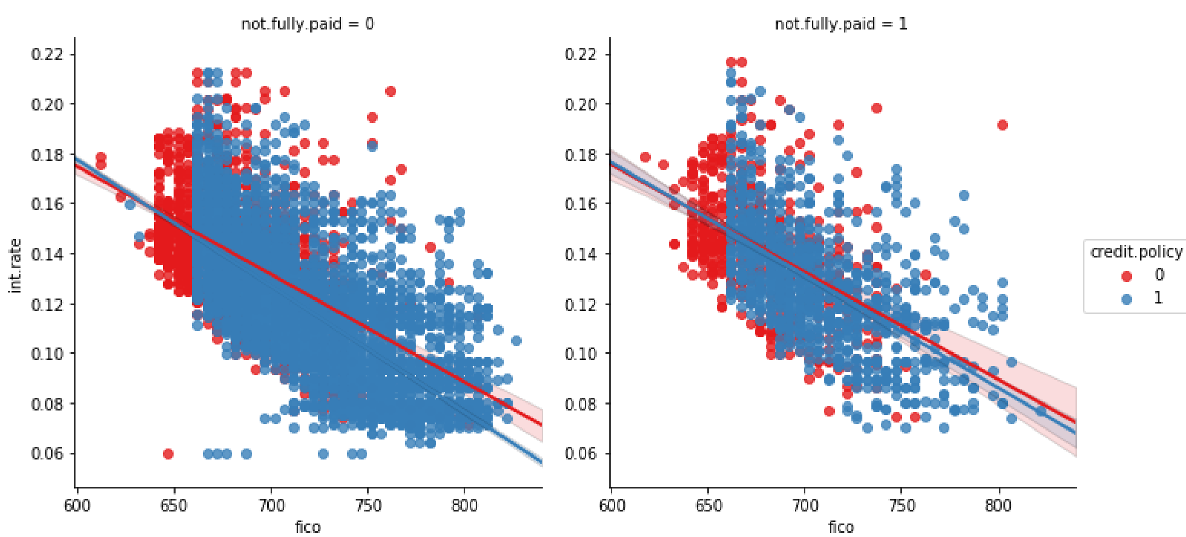
Out[10]: <seaborn.axisgrid.JointGrid at 0xc163f98>



In [13]:
```
plt.figure(figsize=(11,7))
sns.lmplot(x='fico', y='int.rate', data=df, hue='credit.policy',
           col='not.fully.paid', palette='Set1')
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0xcb2a9b0>

<matplotlib.figure.Figure at 0xc8f14e0>



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 [14]:
```
cat_feats = ['purpose']
```

In [15]:
```
final_data = pd.get_dummies(df, columns=cat_feats, drop_first=True)
```

In [16]:
```
final_data.head()
```

Out[16]:

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

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 [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
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 [19]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [20]:
```
dtree = DecisionTreeClassifier()
```

In [21]:
```
dtree.fit(X_train, y_train)
```

Out[21]:
```
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 [22]:
```
pred = dtree.predict(X_test)
```

In [23]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [24]:
```
print(classification_report(y_test, pred))
```

```
             precision    recall  f1-score   support

          0       0.85      0.81      0.83      2431
          1       0.19      0.23      0.21       443

avg / total       0.75      0.72      0.74      2874
```

In [25]:
```
print(confusion_matrix(y_test, pred))
```

```
[[1979  452]
 [ 339  104]]
```

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 [26]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [27]:
```
rfc = RandomForestClassifier(n_estimators=200)
```

In [28]:
```
rfc.fit(X_train, y_train)
```

Out[28]:
```
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=200, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

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 [29]:
```
pred = rfc.predict(X_test)
```

In [30]:
```
print(classification_report(y_test, pred))
```

```
             precision    recall  f1-score   support

          0       0.85      0.99      0.92      2431
          1       0.43      0.02      0.04       443

avg / total       0.78      0.84      0.78      2874
```

In [31]:
```
print(confusion_matrix(y_test, pred))
```

```
[[2418   13]
 [ 433   10]]
```

What performed better the random forest or the decision tree?

In [34]:
```
# Depends what metric you are trying to optimize for.
# Notice the recall for each class for the models.
# Neither did very well, more feature engineering is needed.
```