In [1]:

```
#PROJECT INTRODUCTION and WHAT IT IS ABOUT

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

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

#I will use lending data from 2007-2010 and be trying to classify and predict whether o
r not the borrower paid back
#their loan in full.
```

In [2]:

```
import pandas as pd
import numpy as np
```

In [3]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:

```
loan=pd.read_csv('loan_data.csv')
```

In [7]:

```
#Let's start to know what's our data all about
loan.head()
```

Out[7]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.c
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

In [8]:

```
loan.info()
```

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

In [9]:

```
loan.describe()
```

Out[9]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days
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	1

In [10]:

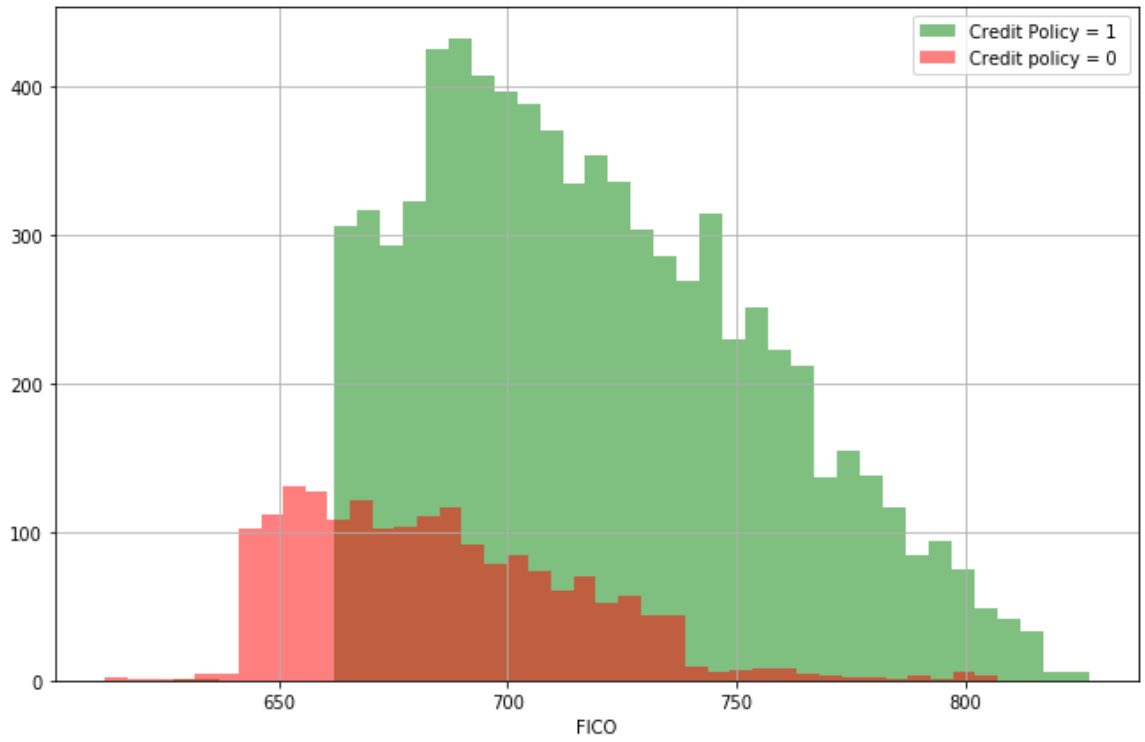
```
 #Here's moving on to some exploratory data analysis and visualisation to get some insi
ghts from our data
```

In [14]:

```
 #A histogram of two FICO distributions on top of each other, one for each credit.polic
y outcome
plt.figure(figsize=(11,7))
loan[loan['credit.policy']==1]['fico'].hist(bins=40,color='green',label='Credit Policy
 = 1 ',alpha=0.5)
loan[loan['credit.policy']==0]['fico'].hist(bins=40,color='red',label='Credit policy =
 0',alpha=0.5)
plt.legend()
plt.xlabel('FICO')
```

Out[14]:

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



In [15]:

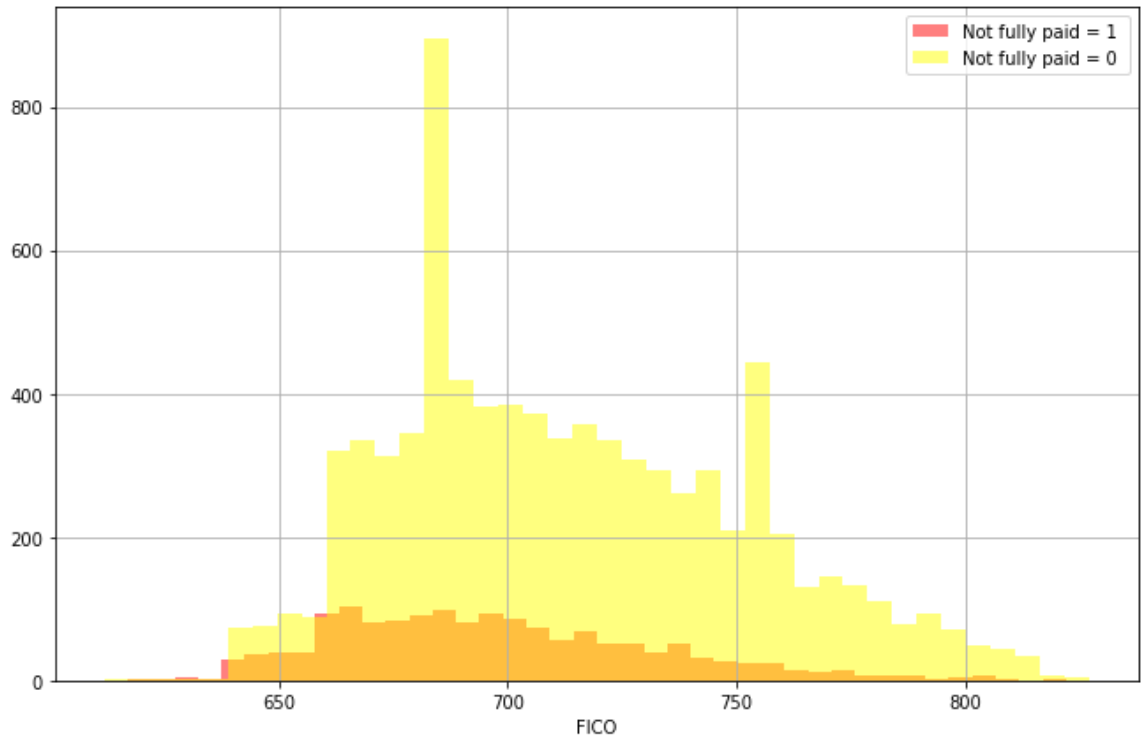
```
#We can see that the fico scores are directly proportional to the credit policy criteri
a,
#We can also see that there is some kind of a cut off point around 660 fico score, belo
w which credit policy is not matched
```

In [34]:

```
plt.figure(figsize=(11,7))
loan[loan['not.fully.paid']==1]['fico'].hist(bins=40,color='red',label='Not fully paid
 = 1 ',alpha=0.5)
loan[loan['not.fully.paid']==0]['fico'].hist(bins=40,color='yellow',label='Not fully pa
id = 0',alpha=0.5)
plt.legend()
plt.xlabel('FICO')
```

Out[34]:

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



In [19]:

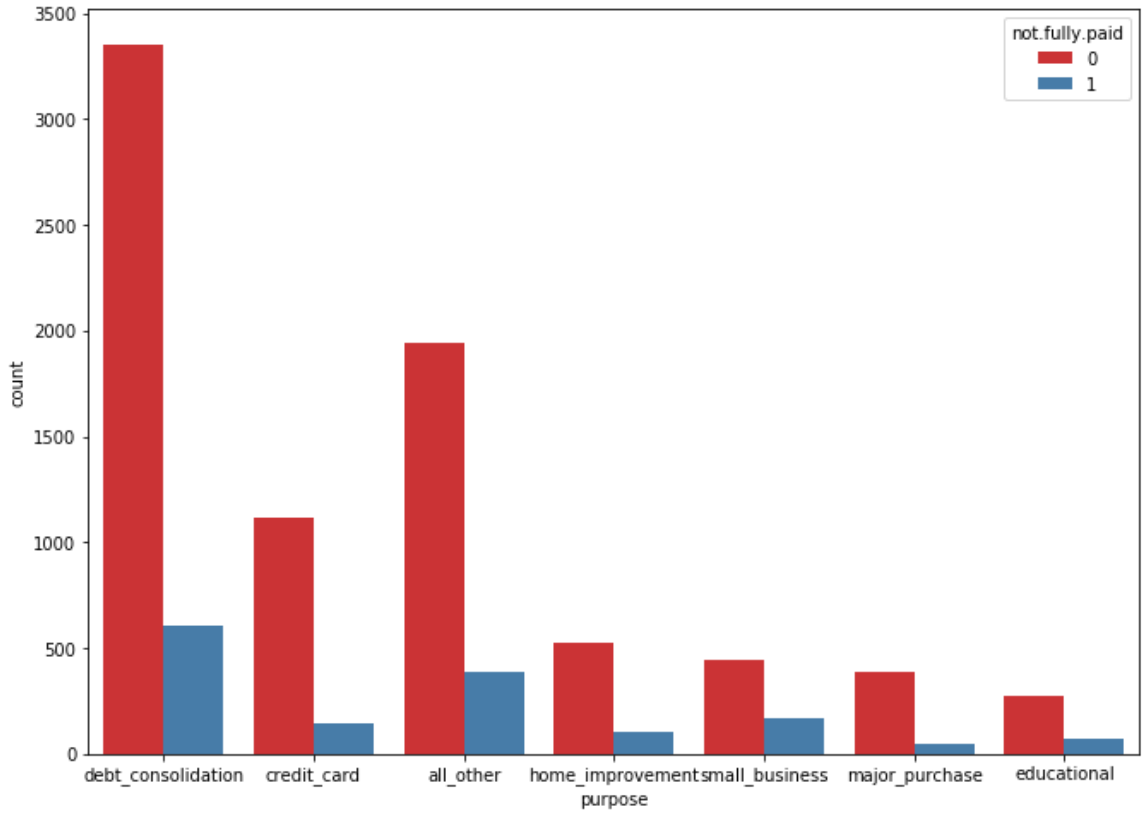
```
#Seems like most people have paid up, good work people, keep up the grind
```

In [21]:

```
#A countplot using seaborn showing the counts of loans by purpose, with the color hue d
efined by not.fully.paid
plt.figure(figsize=(11,8))
sns.countplot(x='purpose',data=loan,hue='not.fully.paid',palette='Set1')
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20f5ebf1708>
```



In [22]:

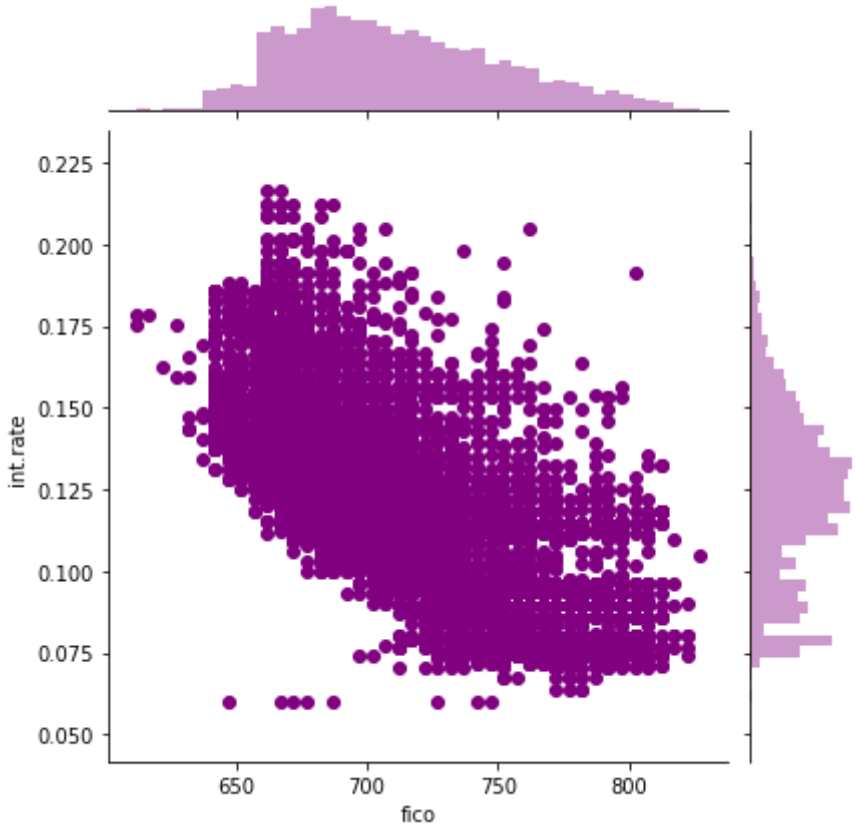
```
#Hmm, so we observe that debt consolidation is the most common reason of wanting a loan
```

In [25]:

```
#Let's see the trend between FICO score and interest rate with the following jointplot.
sns.jointplot(x='fico',y='int.rate',data=loan,color='purple')
```

Out[25]:

```
<seaborn.axisgrid.JointGrid at 0x20f5e917788>
```



In [26]:

```
#Now what we observe is as the fico score increases you have better credit,
#so the interest rate is probably going to go lower . Makes sense.
```

In [27]:

```
# The following lmplots are to see if the trend differed between not.fully.paid and cre
dit.policy.
```

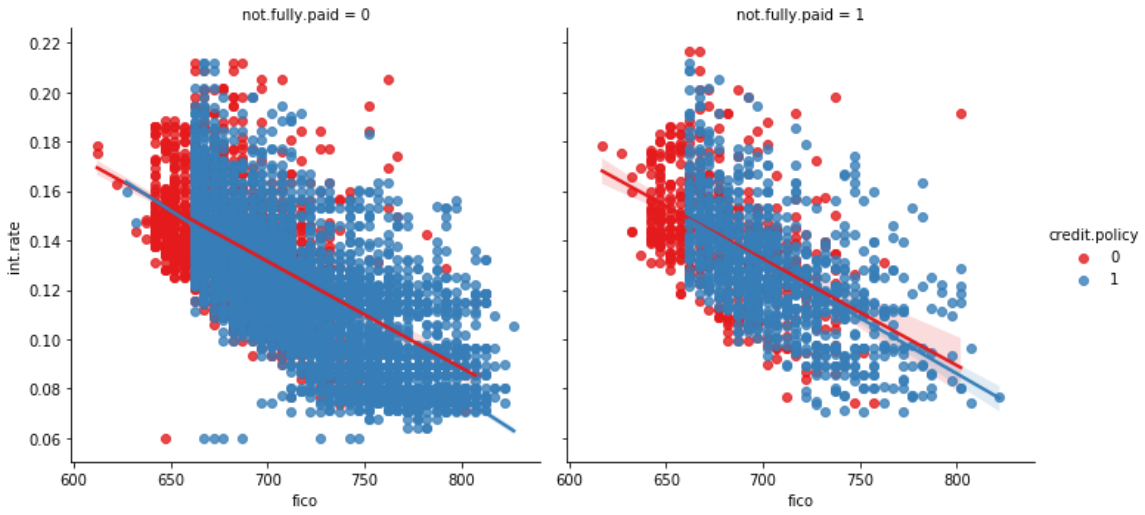
In [30]:

```
plt.figure(figsize=(12,8))
sns.lmplot(x='fico',y='int.rate',data=loan,hue='credit.policy',
           col='not.fully.paid',palette='Set1')
```

Out[30]:

```
<seaborn.axisgrid.FacetGrid at 0x20f5ed12f88>

<Figure size 864x576 with 0 Axes>
```



In [31]:

```
#Behavior seems pretty much the same, on both paid and not paid sides , based on the cr
edit policy
```

In [32]:

```
#We've done some good amount of exploratory data analysis
```

In [33]:

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

In [35]:

```
#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.
```

In [36]:

```
category_feats=['purpose']
```

In [37]:

```
set_data=pd.get_dummies(loan,columns=category_feats,drop_first=True)
```

In [38]:

```
set_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   credit.policy               9578 non-null   int64
 1   int.rate                    9578 non-null   float64
 2   installment                 9578 non-null   float64
 3   log.annual.inc              9578 non-null   float64
 4   dti                         9578 non-null   float64
 5   fico                        9578 non-null   int64
 6   days.with.cr.line           9578 non-null   float64
 7   revol.bal                   9578 non-null   int64
 8   revol.util                  9578 non-null   float64
 9   inq.last.6mths              9578 non-null   int64
 10  delinq.2yrs                 9578 non-null   int64
 11  pub.rec                     9578 non-null   int64
 12  not.fully.paid              9578 non-null   int64
 13  purpose_credit_card         9578 non-null   uint8
 14  purpose_debt_consolidation  9578 non-null   uint8
 15  purpose_educational         9578 non-null   uint8
 16  purpose_home_improvement    9578 non-null   uint8
 17  purpose_major_purchase      9578 non-null   uint8
 18  purpose_small_business      9578 non-null   uint8
dtypes: float64(6), int64(7), uint8(6)
memory usage: 1.0 MB
```

In [39]:

```
#We can see that the purpose column is disintegrated
```

In [40]:

```
set_data.head()
```

Out[40]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	r
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	

In [41]:

```
#So our data is ready , lets move forward
#Time to split our data into a training set and a testing set.
```

In [42]:

```
from sklearn.model_selection import train_test_split
```

In [45]:

```
#Splitting up our data
X=set_data.drop('not.fully.paid',axis=1)
y=set_data['not.fully.paid']
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=
101)
```

In [46]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [47]:

```
#Lets begin with a single tree, and then we'll move forward to random forrests
dtree=DecisionTreeClassifier()
```

In [48]:

```
dtree.fit(X_train,y_train)
```

Out[48]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=N
one,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [49]:

```
#Predictions and Evaluation of Decision Tree
```

In [50]:

```
dtree_pred=dtree.predict(X_test)
```

In [51]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [52]:

```
print(confusion_matrix(y_test,dtree_pred))
print('\n')
print(classification_report(y_test,dtree_pred))
```

```
[[1988  443]
 [ 336  107]]


              precision    recall  f1-score   support

           0       0.86      0.82      0.84      2431
           1       0.19      0.24      0.22       443

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

In [53]:

```
#Training the Random Forest model
from sklearn.ensemble import RandomForestClassifier
```

In [54]:

```
rfc=RandomForestClassifier(n_estimators=350)
```

In [55]:

```
rfc.fit(X_train,y_train)
```

Out[55]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=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=350,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [57]:

```
rfc_pred=rfc.predict(X_test)
```

In [58]:

```
print(confusion_matrix(y_test,rfc_pred))
print('\n')
print(classification_report(y_test,rfc_pred))
```

```
[[2423    8]
 [ 431   12]]


              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2431
           1       0.60      0.03      0.05       443

    accuracy                           0.85      2874
   macro avg       0.72      0.51      0.48      2874
weighted avg       0.81      0.85      0.78      2874
```

In [59]:

```
#What performed better the random forest or the decision tree??
```

In [61]:

```
#Well it really depends what metric we are trying to optimise for
#Look at recall,a single decision tree did well for class 1 than random forrest
#So it depends what the cost associated with each of the metrics
#overall average of random forrest was better, but not alike in some particuar metrics
#There comes the business domain experience
```

In [ ]:

```

```