Decision Trees and Random Forest Project

Problem Statement- For this project 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 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, 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.

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).

Method: Decision Trees and Random Forest

Source of Data: Udemy/Tool: Python

Imports multiple libraries as below screen:

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

Dataset: head of dataset is as below and Target Colum**: not.fully.paid**

Get the Data

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

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

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

```
loans.info()
```

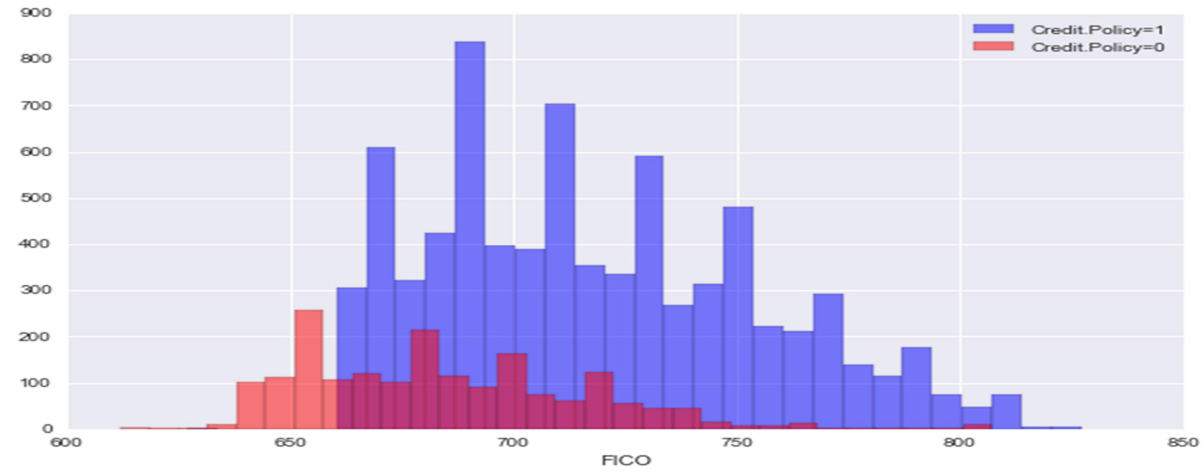
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
credit.policy        9578 non-null int64
purpose              9578 non-null object
int.rate             9578 non-null float64
installment          9578 non-null float64
log.annual.inc       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
```

Exploratory Data Analysis: we will check data behavior via random plot analysis.

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

```
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')
```

```
<matplotlib.text.Text at 0x119963f28>
```

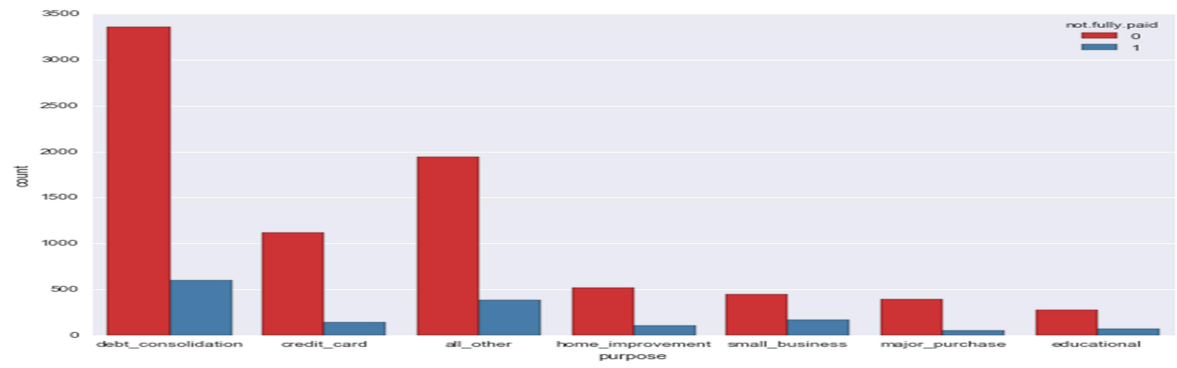


Observation: as we c an see blue line for customers who all was eligible for loan and red that' have not good score or these were not right customer to get loan. However they got the amount.

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

```
plt.figure(figsize=(11,7))
sns.countplot(x='purpose',hue='not.fully.paid',data=loans,palette='Set1')
```

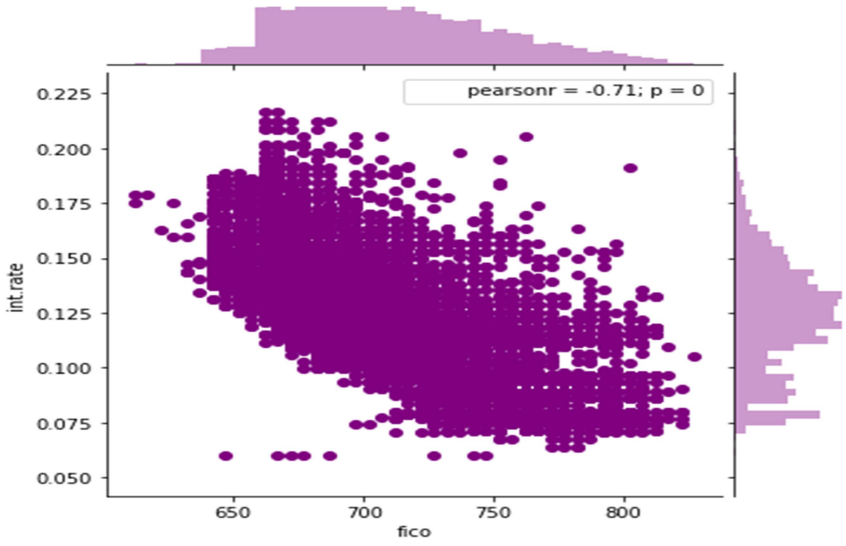
```
<matplotlib.axes._subplots.AxesSubplot at 0x119996828>
```



Observation: this gives clear picture of taking loan for purpose with not paid status, we can see highest due for debt consolidation followed by all other expense and credit card.

- **Let's see the trend between FICO score and interest rate. Recreate the following jointplot.**

sns.jointplot(x='fico',y='int.rate',data=loan,color='purple',space=0.1)

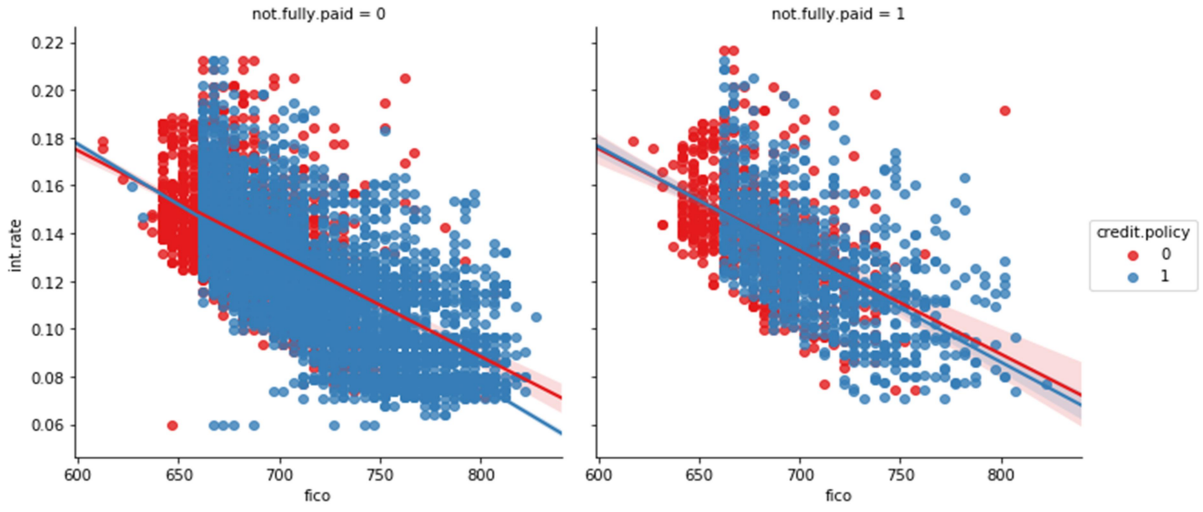


Observation: as it is very common to get higher int rate while people have less credit score which also shows by trends.

- **Create the following lmplots to see if the trend differed between not.fully.paid and credit.policy.**

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

```
<seaborn.axisgrid.FacetGrid at 0x2b7c60bbe48>
```



Setting up the Data

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

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.

- cat_feats = ['purpose']
- final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True)

```
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 float64
purpose_debt_consolidation    9578 non-null float64
purpose_educational           9578 non-null float64
purpose_home_improvement      9578 non-null float64
purpose_major_purchase        9578 non-null float64
purpose_small_business        9578 non-null float64
dtypes: float64(12), int64(7)
memory usage: 1.4 MB
```

Observation: we can see new data call final_data have transform value for ‘purpose’ now and have separate columns for all categories inside purpose.

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

- from sklearn.model_selection import train_test_split
- 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=0.30, random_state=101)

Training a Decision Tree Model

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

Import DecisionTreeClassifier

- **from sklearn.tree import DecisionTreeClassifier**

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

- dtree = DecisionTreeClassifier()
- dtree.fit(X_train,y_train)

Predictions and Evaluation of Decision Tree

- **predictions = dtree.predict(X_test)**
- **from sklearn.metrics import classification_report,confusion_matrix**
- **print(classification_report(y_test,predictions))**

	precision	recall	f1-score	support
0	0.85	0.82	0.84	2431
1	0.19	0.23	0.20	443
avg / total	0.75	0.73	0.74	2874

- **print(confusion_matrix(y_test,predictions))**

```
[[1995  436]
 [ 343  100]]
```

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.

- from sklearn.ensemble import RandomForestClassifier
- rfc = RandomForestClassifier(n_estimators=600)
- rfc.fit(X_train,y_train)

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.

- predictions = rfc.predict(X_test)

Now create a classification report from the results.

- from sklearn.metrics import classification_report,confusion_matrix
- print(classification_report(y_test,predictions))

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2431
           1       0.57      0.03      0.05       443

avg / total       0.81      0.85      0.78      2874
```

Show the Confusion Matrix for the predictions.

```
print(confusion_matrix(y_test,predictions))
```

```
[[2422    9]
 [ 431   12]]
```

Final Observation: Model has perform good as can see 81% precision, 85% recall, 78% F1-score, but comparing with both above classification report,

there is huge gap between “recall value”. So it depends what metric you are trying to optimize model.