Loan Worthiness of a Borrower - Data Modeling using Python

Project Goal:

To create a model that will help predict if the borrower has a high probability of paying their loan back in full.

Dataset:

In this project, we are using publicly available dataset from Lendingclub.com from 2007 - 2010 to predict if the boorower paid back their entire loan amount to the investor.

The data set contains the following features:

- 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).

Let us start with importing the required Libraries

In [2]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('whitegrid')
plt.rcParams["patch.force_edgecolor"] = True
```

Call the Dataset

In [3]:
```
loanworth = pd.read_csv('loan.csv')
```

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

In [4]:
```
loanworth.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 [5]:
```
loanworth.describe()
```

Out[5]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fi
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.0000
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.84631
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.00000
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.00000
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.00000
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.00000
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.00000

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

Out[6]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	day
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	563
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	276
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	471
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	269
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	406

Exploratory Data Analysis

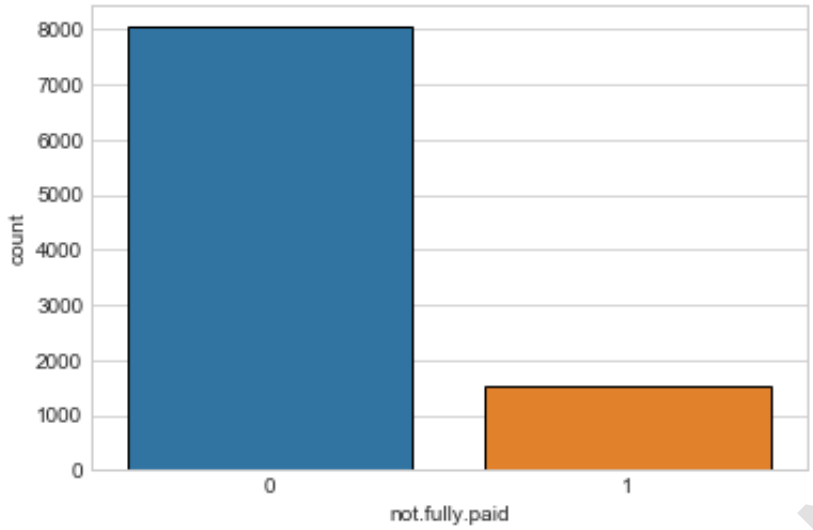
Finding out the count of the borrowers who have paid the loan to the full and those who haven't.

In [94]:
```
loanworth['not.fully.paid'].value_counts()
```

Out[94]:
```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

In [95]:
```
sns.countplot(x='not.fully.paid',data=loanworth)
```

Out[95]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1d774a58>
```

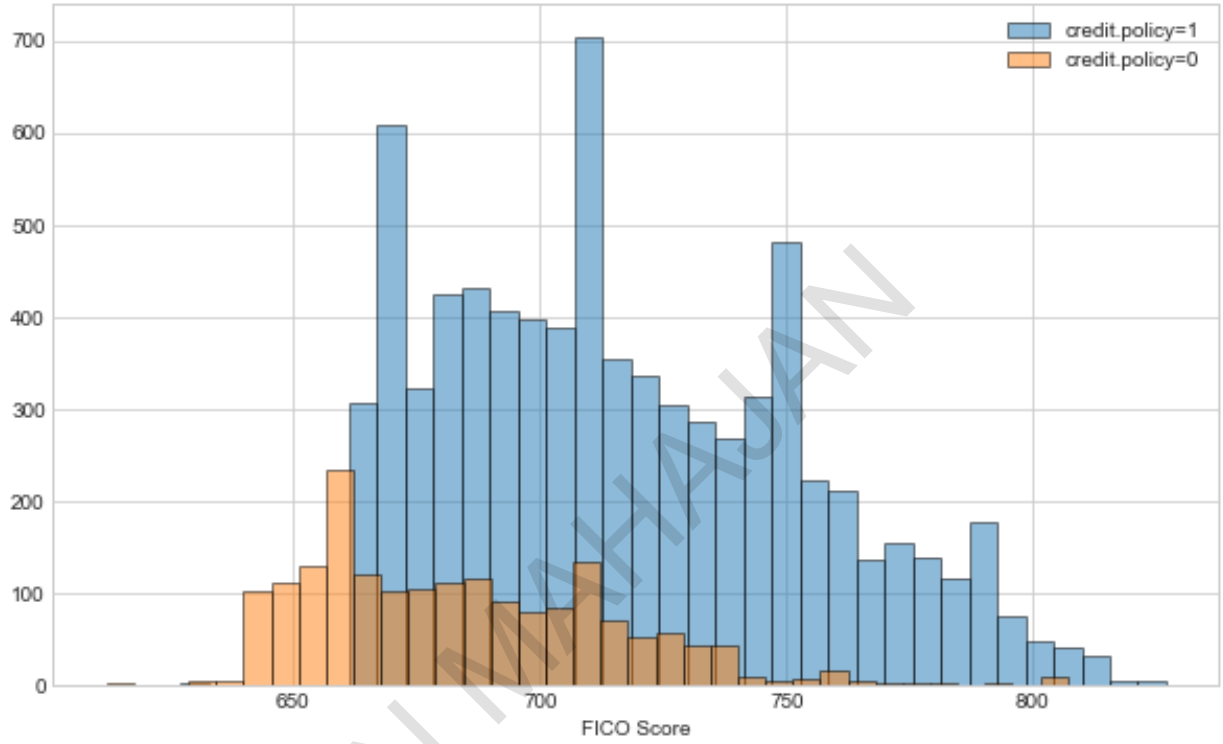


We see that the number of borrowers who have paid their loans to the full is higher.

Let us create the histogram for FICO score with respect to the Credit Policies

In [18]:
```
plt.figure(figsize=(10,6))
loanworth[loanworth['credit.policy']==1]['fico'].hist(bins=35,alpha=0.
5,label='credit.policy=1')
loanworth[loanworth['credit.policy']==0]['fico'].hist(bins=35,alpha=0.
5,label='credit.policy=0')
plt.xlabel('FICO Score')
plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x1a1be733c8>

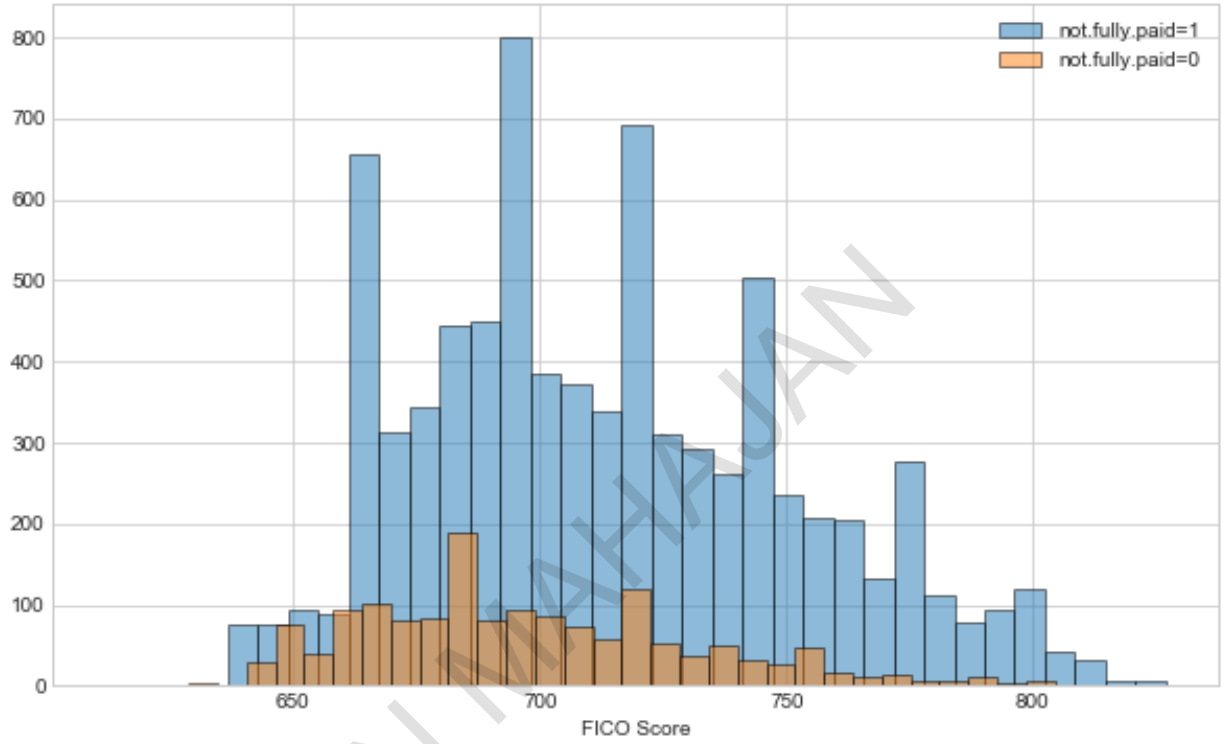


Borrowers having higher FICO scores meets the credit underwriting criteria of LendingClub.com

Creating the histogram of FICO score with respect to the not.fully.paid column.

In [17]:
```
plt.figure(figsize=(10,6))
loanworth[loanworth['not.fully.paid']==0]['fico'].hist(bins=35,alpha=0
.5,label='not.fully.paid=1')
loanworth[loanworth['not.fully.paid']==1]['fico'].hist(bins=35,alpha=0
.5,label='not.fully.paid=0')
plt.xlabel('FICO Score')
plt.legend()
```

Out[17]:
```
<matplotlib.legend.Legend at 0x1a1b7c1c50>
```

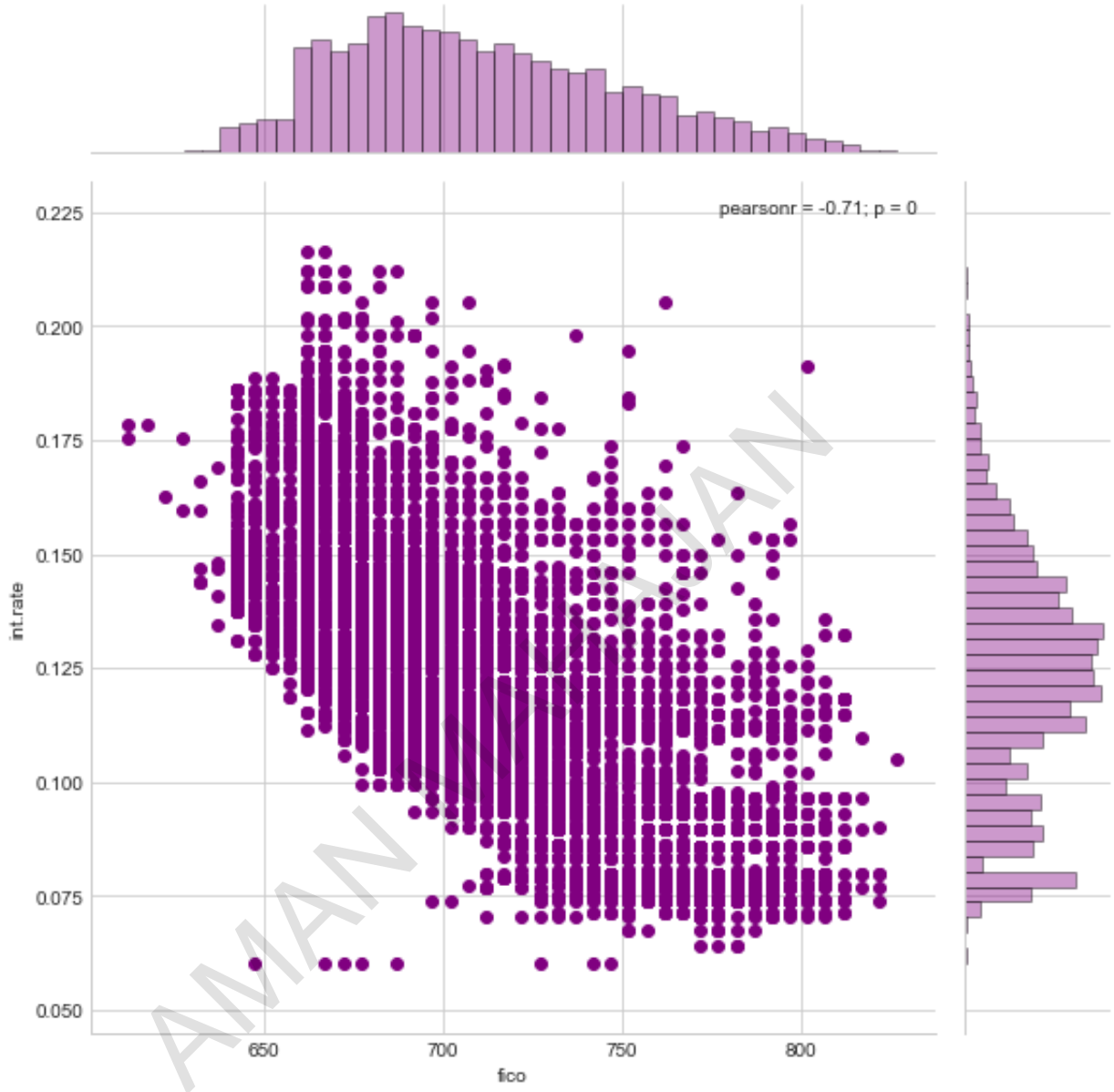


We see that the number of borrowers who have not paid the loan to the full is high for higher FICO scores.

Analyzing the trend between FICO score and interest rate now

In [14]:
```
sns.jointplot(x='fico',y='int.rate',data=loanworth,color='purple',size
=8)
```

Out[14]: <seaborn.axisgrid.JointGrid at 0x1a1aee6c18>



We see that lower FICO scores have higher interest rates.

Now analyzing the trend between not.fully.paid and credit.policy.

In [16]:
```
sns.lmplot(x='fico',y='int.rate',data=loanworth,col='not.fully.paid',hue='credit.policy',palette='Set1')
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x1a1b782e48>

The trend of FICO score and interest rate is similar as we expect for those that fully paid balances versus those that did not.

Setting up the Data

In [19]:
```
loanworth.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
```

Transforming Categorical Variables

Here, PURPOSE column is a categorical variable. Therefore we need to transform them using dummy variables so sklearn will be able to understand them. We will do it using pd.get_dummies.

In [20]:
```
cat_feats = ['purpose']
```

Converting categorical variable into dummy variable:

In [21]:
```
final_data = pd.get_dummies(loanworth,columns=cat_feats,drop_first=True)
```

In [22]:
```
final_data.head()
```

Out[22]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.l
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 of Dataset

Split our data into a training set and a test set! We will do a 70/30 split and use a random state for reproducibility.

In [23]:
```
from sklearn.model_selection import train_test_split
```

In [24]:
```
final_data.columns
```

Out[24]:
```
Index(['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', 'purpose_home_improvement',
       'purpose_major_purchase', 'purpose_small_business'],
      dtype='object')
```

In [76]:
```
X = final_data.drop('not.fully.paid',axis=1)
y = final_data['not.fully.paid']
```

In [77]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
3, random_state=101)
```

Training the data using Logistic Regression Model

In [78]:
```
from sklearn.linear_model import LogisticRegression
```

In [79]:
```
log_model = LogisticRegression()
```

In [80]:
```
log_model.fit(X_train,y_train)
```

Out[80]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jo
bs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0
.0001,
          verbose=0, warm_start=False)
```

In [81]:
```
log_predictions = log_model.predict(X_test)
```

Training the data using Decision Tree Model

In [82]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [83]:
```
dtree = DecisionTreeClassifier(random_state=101)
```

In [84]:
```
dtree.fit(X_train,y_train)
```

Out[84]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=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_stat
e=101,
            splitter='best')
```

In [85]:
```
dtree_predictions = dtree.predict(X_test)
```

Training the data using Random Forest Model

In [86]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [87]:
```
rfc = RandomForestClassifier(n_estimators=1000,random_state=101)
```

In [88]:
```
rfc.fit(X_train,y_train)
```

Out[88]:
```
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=1000, n_jobs=
1,
            oob_score=False, random_state=101, verbose=0, warm_start
=False)
```

In [89]:
```
rfc_predictions = rfc.predict(X_test)
```

Implying Confusion Matrix on all models to test the accuracy of models

In [90]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

Logistic Regression Model:

In [91]:
```
print('Confusion Matrix:')
print(confusion_matrix(y_test,log_predictions))
print('\n')
print('Classification Report:')
print(classification_report(y_test,log_predictions))
```

```
Confusion Matrix:
[[2425    6]
 [ 434    9]]


Classification Report:
             precision    recall  f1-score   support

          0       0.85      1.00      0.92      2431
          1       0.60      0.02      0.04       443

avg / total       0.81      0.85      0.78      2874
```

The logistic regression predicted very well for the fully paid class, but terribly on the not fully paid class.

Decision Tree Model:

In [92]:
```
print('Confusion Matrix:')
print(confusion_matrix(y_test,dtree_predictions))
print('\n')
print('Classification Report:')
print(classification_report(y_test,dtree_predictions))
```

```
Confusion Matrix:
[[1976  455]
 [ 336  107]]


Classification Report:
             precision    recall  f1-score   support

          0       0.85      0.81      0.83      2431
          1       0.19      0.24      0.21       443

avg / total       0.75      0.72      0.74      2874
```

The decision tree predicted decently on the fully paid class, but bad on the not fully paid class. However, it's an improvement from the logistic regression.

Random Forest Model:

In [93]:
```
print('Confusion Matrix:')
print(confusion_matrix(y_test,rfc_predictions))
print('\n')
print('Classification Report:')
print(classification_report(y_test,rfc_predictions))
```

```
Confusion Matrix:
[[2420   11]
 [ 431   12]]


Classification Report:
             precision    recall  f1-score   support

          0       0.85      1.00      0.92      2431
          1       0.52      0.03      0.05       443

avg / total       0.80      0.85      0.78      2874
```

The random forest model is very similar to the logistic regression. Better performance on the fully paid class, but terrible on the not fully paid class.

We now have the accuracy analyis for all the models on both the cases i.e. Paid Fully and Not Paid Fully.

Conclusion:

When we check the average/total of the classification report, we see that the Random Forest and Logistic Regression models perform better than Decision Tree model on all the criterias. But when analyzed indivisually, the performance of logistic regression and random forest models for the not fully paid class is bad in comparison to decision tree model. The decision tree is the better model for predicting accuracy in both classes. So, if the investors want to decide who they should lend loans to in future, they should consider both the classes simultaneously to make a fair decision and not loose out any potential customer neither give credit a bad borrower.