Predicting Loan Repayment Using Machine Learning: Decision Tree vs. Random Forest

Exploring publicly available data from LendingClub.com. Lending data from 2007-2010 and 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).

DATA SETUP

- Importing numpy, pandas, visualisation libraries.

[1]:

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

- Reading csv file

[5]:

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

- Information of the file

[7]:

```
loans.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
```

[9]:

```
loans.describe()
```

[9]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	9578.000000	9578.000000	9578.000000	9578.00
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	46.799236	1.577469	0.163708	0.06
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	29.014417	2.200245	0.546215	0.26
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.000000	0.000000	0.000000	0.00
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	22.600000	0.000000	0.000000	0.00
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	46.300000	1.000000	0.000000	0.00
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	70.900000	2.000000	0.000000	0.00
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.000000	33.000000	13.000000	5.00

- Sample dataset

[11]:

```
loans.head()
```

[11]:

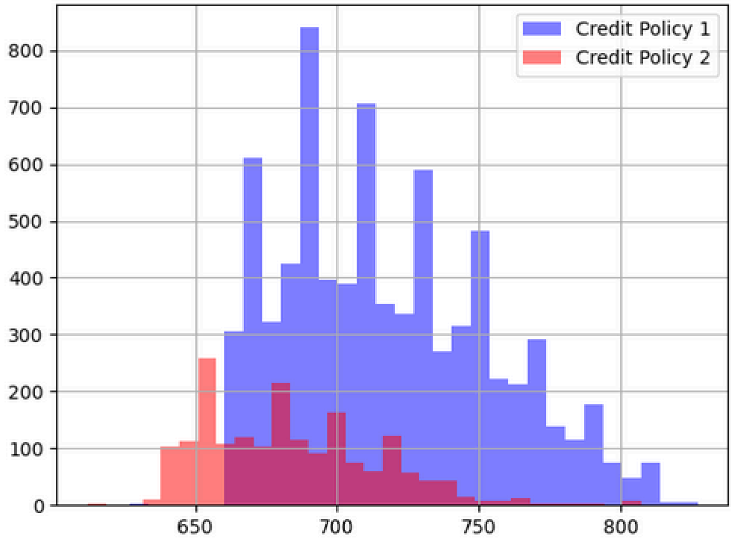
	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.p
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	

EXPLORATORY DATA ANALYSIS

- Creating a histogram of two FICO distributions on top of each other, one for each credit.policy outcome and not.fully.paid column.

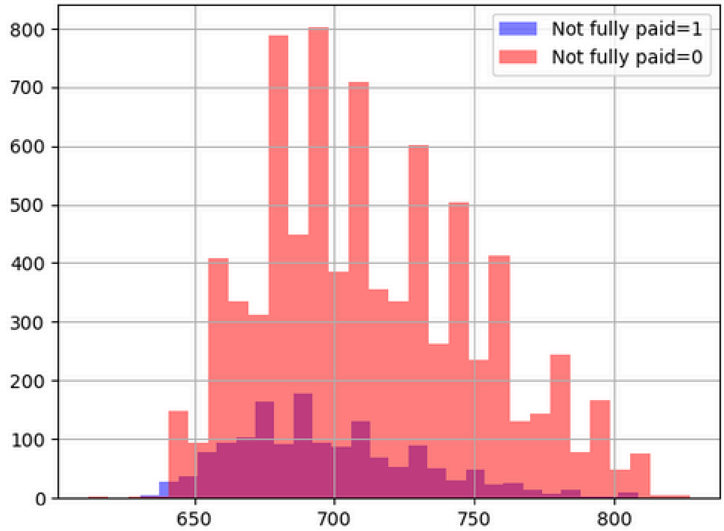
[41]:
```
loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',bins=30,label='Credit Policy 1')
loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5,color='red',bins=30,label='Credit Policy 2')
plt.legend()
```

[41]: `<matplotlib.legend.Legend at 0x1ef28522ed0>`



[47]:
```
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,bins=30,color='blue',label='Not fully paid=1')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,bins=30,color='red',label='Not fully paid=0')
plt.legend()
```

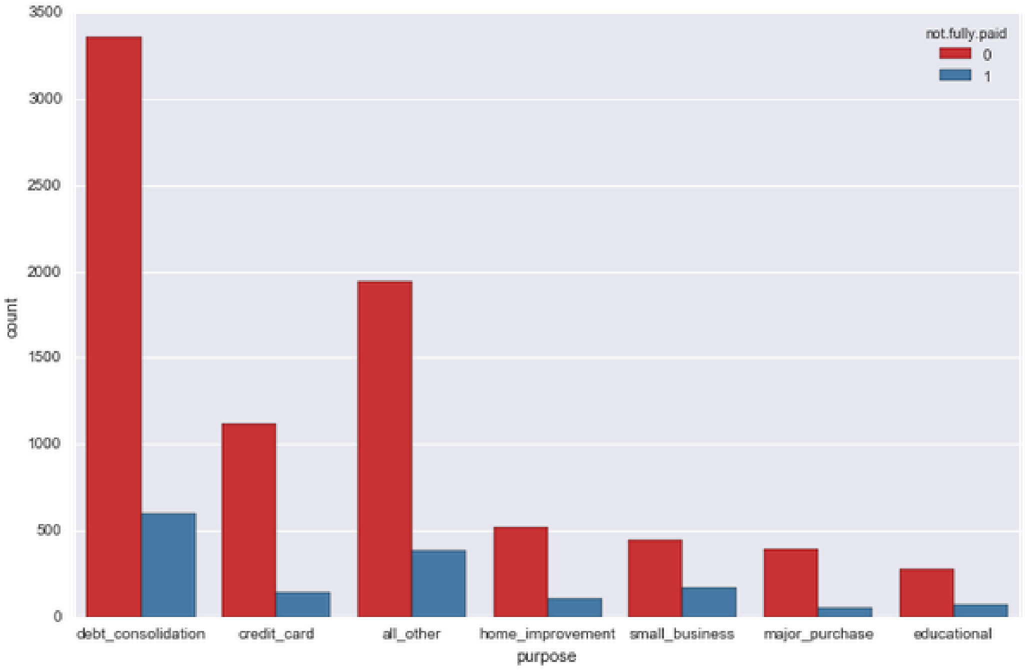
[47]: `<matplotlib.legend.Legend at 0x1ef2cde2ed0>`



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

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

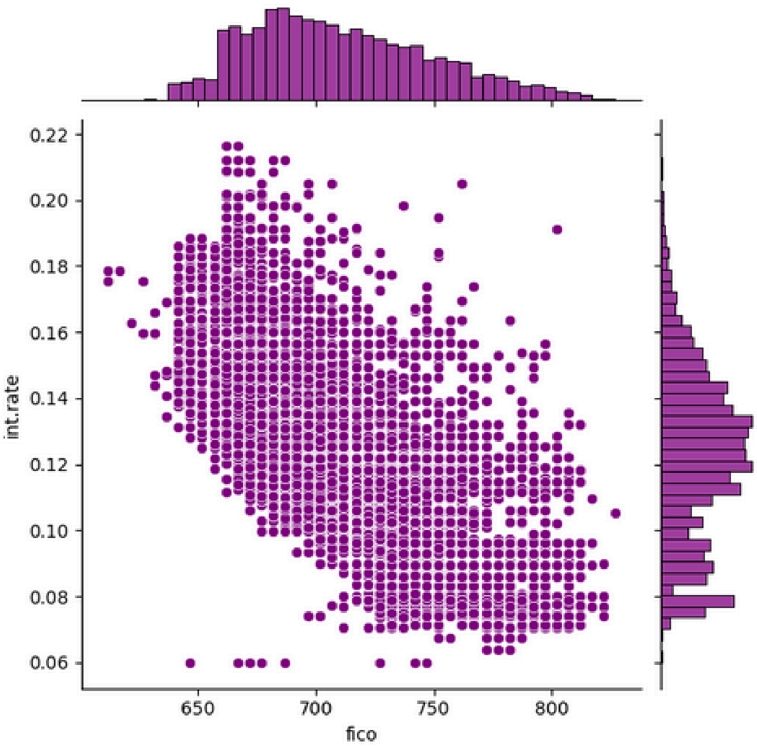
```
]: <matplotlib.axes._subplots.AxesSubplot at 0x119996828>
```



- The trend between FICO score and interest rate.

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

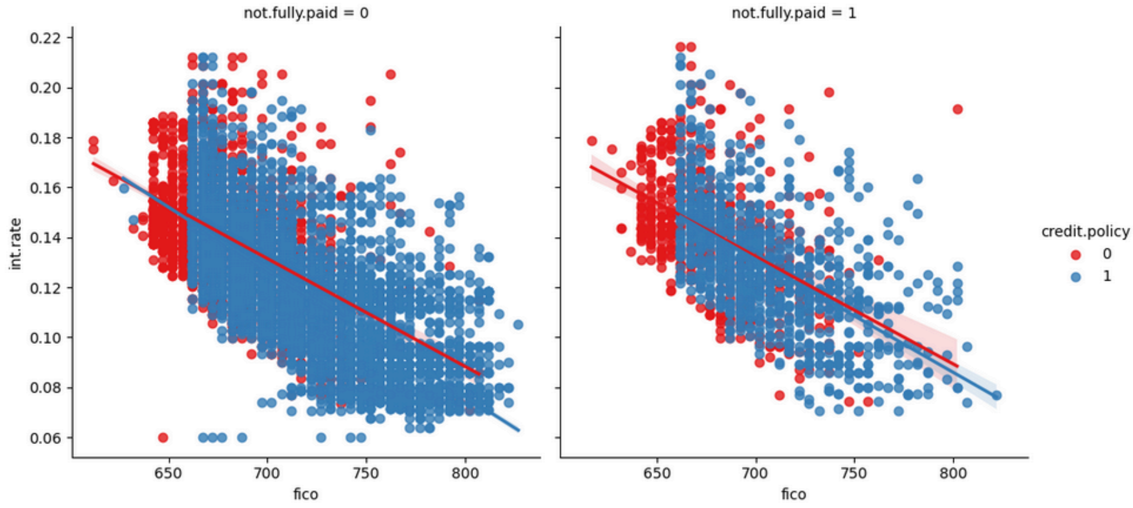
```
[55]: <seaborn.axisgrid.JointGrid at 0x1ef2d0b1750>
```



- Creating a lmplots to see if the trend differed between not.fully.paid and credit.policy.

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

```
[63]: <seaborn.axisgrid.FacetGrid at 0x1ef2c982110>
```



Categorical Features

- **Purpose** column is categorical, transforming them using dummy variables so sklearn will be able to understand them.

```
63]: 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):
 #   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   bool   
 14  purpose_debt_consolidation  9578 non-null   bool   
 15  purpose_educational         9578 non-null   bool   
 16  purpose_home_improvement    9578 non-null   bool   
 17  purpose_major_purchase      9578 non-null   bool   
 18  purpose_small_business      9578 non-null   bool   
dtypes: bool(6), float64(6), int64(7)
memory usage: 1.0 MB
```

In the info … purpose has categorical data now

Train Test Split

- Using sklearn to split the data into a training set and a testing set.

[107]:
```
from sklearn.model_selection import train_test_split
```

[119]:
```
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)
```

Training a Decision Tree Model

- Importing DecisionTreeClassifier, creating an instance and fit it to the training data.

[165]:
```
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

[165]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

Prediction and Evaluation of Decision Tree

- Creating predictions from the test set and creating a classification report and a confusion matrix.

[167]:
```
prediction=dtree.predict(X_test)
```

[169]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[171]:
```
print(classification_report(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85      2405
           1       0.24      0.24      0.24       469

    accuracy                           0.75      2874
   macro avg       0.54      0.54      0.54      2874
weighted avg       0.75      0.75      0.75      2874
```

[173]:
```
print(confusion_matrix(y_test,prediction))
```

```
[[2038  367]
 [ 356  113]]
```

Training a Random Forest Model

- Importing DecisionTreeClassifier, creating an instance and fit it to the training data.

[175]:
```
from sklearn.ensemble import RandomForestClassifier
```

[177]:
```
rfc=RandomForestClassifier(n_estimators=200)
```

[179]:
```
rfc.fit(X_train,y_train)
```

[179]:
```
RandomForestClassifier
RandomForestClassifier(n_estimators=200)
```

Prediction and Evaluation of Random Forest

- Creating predictions from the test set and creating a classification report and a confusion matrix.

[181]:
```
pred_rfc=rfc.predict(X_test)
print(classification_report(y_test,pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      2405
           1       0.53      0.02      0.04       469

    accuracy                           0.84      2874
   macro avg       0.68      0.51      0.48      2874
weighted avg       0.79      0.84      0.77      2874
```

[183]:
```
print(confusion_matrix(y_test,pred_rfc))
```

```
[[2396    9]
 [ 459   10]]
```

- **Preferred model:**
  The Random Forest model, since it demonstrated superior performance compared to the Decision Tree model, achieving higher accuracy and better handling of overfitting. This improvement is attributed to the ensemble nature of the Random Forest, which leverages multiple decision trees to enhance predictive performance and robustness.