Random Forest and Decision Trees- Loan data

March 31, 2021

Random Forest and Decision Trees on Loand Data

Sohini Mukherjee

31.03.2021

Importing Libraries

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Getting the Data

[3]:
```
loans = pd.read_csv('E:/2.PYTHON-ML-BOOTCAMP/resources/
↪15-Decision-Trees-and-Random-Forests/loan_data.csv')
```

[4]:
```
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
```

[5]:
```
loans.describe()
```

[5]:
```
       credit.policy     int.rate  installment  log.annual.inc          dti  \
count    9578.000000  9578.000000  9578.000000     9578.000000  9578.000000
mean        0.804970     0.122640   319.089413       10.932117    12.606679
std         0.396245     0.026847   207.071301        0.614813     6.883970
min         0.000000     0.060000    15.670000        7.547502     0.000000
25%         1.000000     0.103900   163.770000       10.558414     7.212500
50%         1.000000     0.122100   268.950000       10.928884    12.665000
75%         1.000000     0.140700   432.762500       11.291293    17.950000
max         1.000000     0.216400   940.140000       14.528354    29.960000

              fico  days.with.cr.line     revol.bal   revol.util  \
count  9578.000000        9578.000000  9.578000e+03  9578.000000
mean    710.846314        4560.767197  1.691396e+04    46.799236
std      37.970537        2496.930377  3.375619e+04    29.014417
min     612.000000         178.958333  0.000000e+00     0.000000
25%     682.000000        2820.000000  3.187000e+03    22.600000
50%     707.000000        4139.958333  8.596000e+03    46.300000
75%     737.000000        5730.000000  1.824950e+04    70.900000
max     827.000000       17639.958330  1.207359e+06   119.000000

       inq.last.6mths  delinq.2yrs      pub.rec  not.fully.paid
count     9578.000000  9578.000000  9578.000000     9578.000000
mean         1.577469     0.163708     0.062122        0.160054
std          2.200245     0.546215     0.262126        0.366676
min          0.000000     0.000000     0.000000        0.000000
25%          0.000000     0.000000     0.000000        0.000000
50%          1.000000     0.000000     0.000000        0.000000
75%          2.000000     0.000000     0.000000        0.000000
max         33.000000    13.000000     5.000000        1.000000
```

[6]:
```
loans.head()
```

[6]:
```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
```

```
3   8.10   712         2699.958333      33667         73.2                 1
4  14.97   667         4066.000000       4740         39.5                 0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

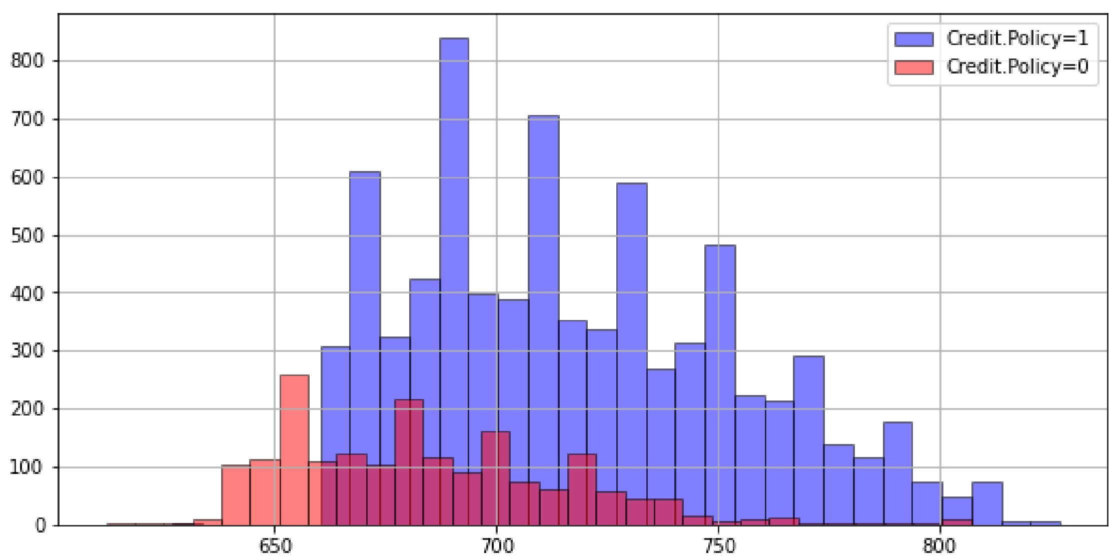
Exploratory Data Analysis

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

[7]:

```
plt.figure(figsize=(10,5))

loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5, bins=30, color='blue',␣
 ↪label='Credit.Policy=1', ec='black')
loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5, bins=30, color='red',␣
 ↪label='Credit.Policy=0', ec='black')
plt.legend()
plt.show()
```

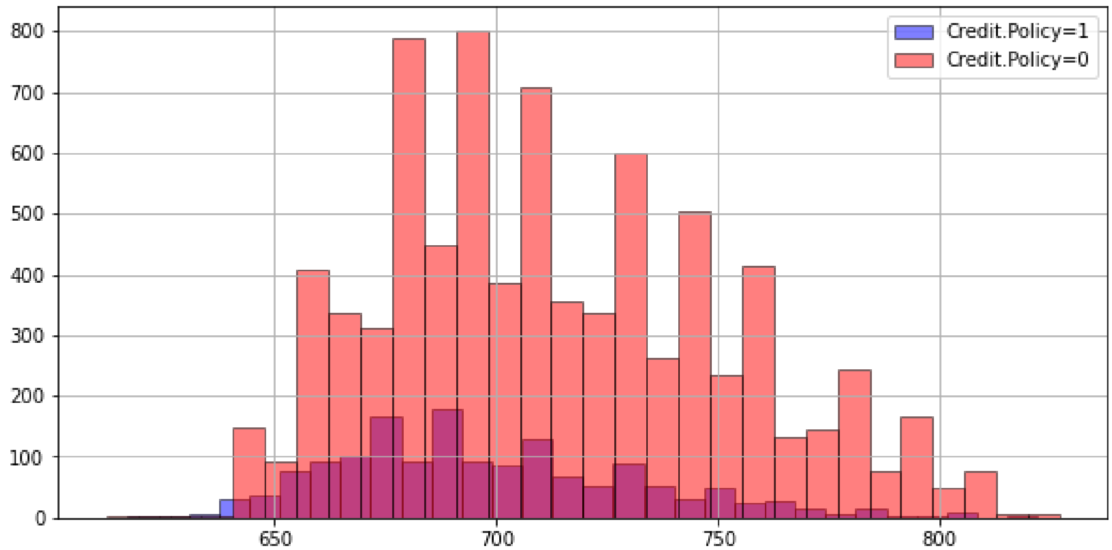


Creating a histogram of two FICO distributions on top of each other, selected by the not.fully.paid column.

[8]:

```
plt.figure(figsize=(10,5))
```

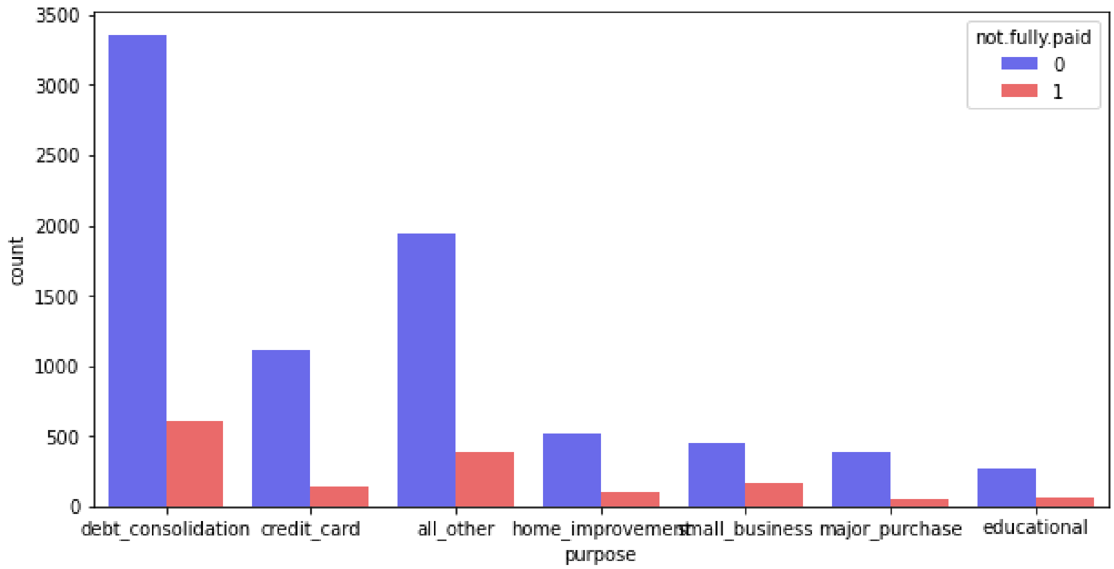
```
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5, bins=30,␣
↪color='blue', label='Credit.Policy=1', ec='black')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5, bins=30, color='red',␣
↪label='Credit.Policy=0', ec='black')
plt.legend()
plt.show()
```



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

[9]:
```
plt.figure(figsize=(10,5))
sns.countplot(x='purpose', data=loans, hue='not.fully.paid', palette='seismic')
plt.tight_layout
```

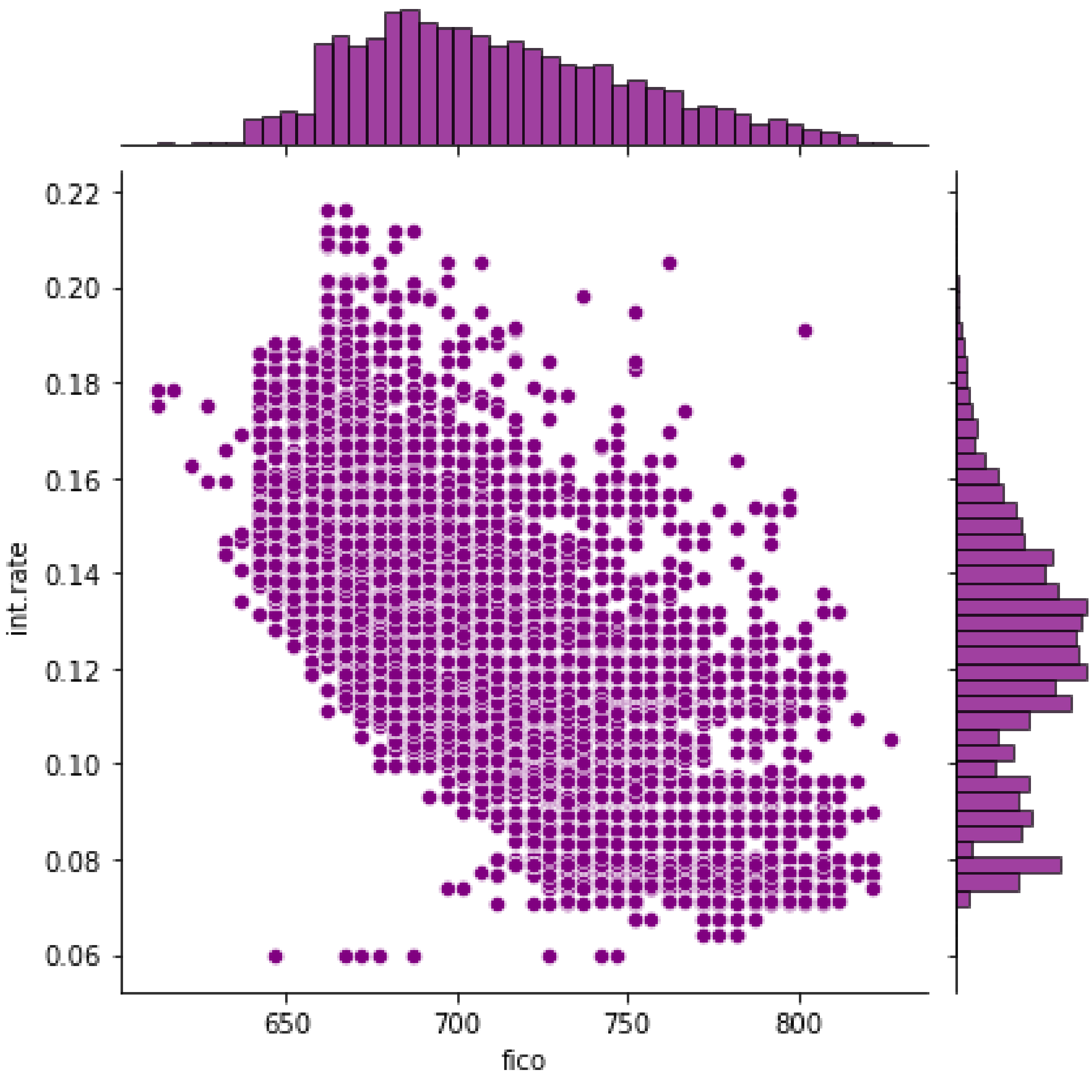
[9]:
```
<function matplotlib.pyplot.tight_layout(*, pad=1.08, h_pad=None, w_pad=None,
rect=None)>
```



Creating the trend between FICO score and interest rate.

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

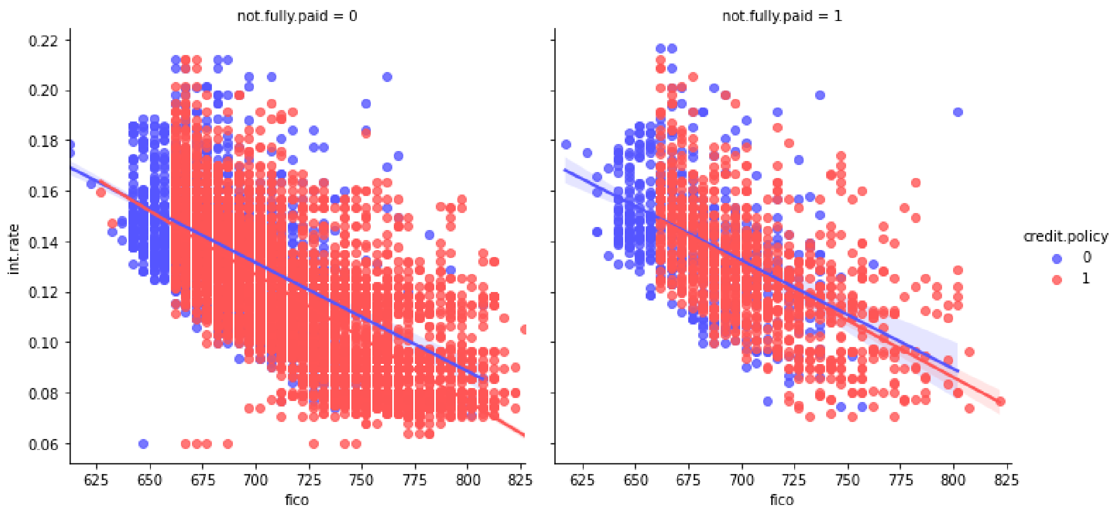
[10]: <seaborn.axisgrid.JointGrid at 0x1fab3b365b0>



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

[11]:
```
sns.lmplot(x='fico', y='int.rate', data=loans, col='not.fully.paid',␣
↪hue='credit.policy', palette='seismic')
```

[11]:
```
<seaborn.axisgrid.FacetGrid at 0x1fab3d53ee0>
```



Since we have 'purpose' as a categorical variable we need to make dummies for the column for ML to work

[12]:
```
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   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
```

Train Test Split

```
[13]: 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=0.3,␣
↪random_state=101)
```

Training a Decision Tree Model

```
[14]: from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()

dtree.fit(X_train, y_train)
```

```
[14]: DecisionTreeClassifier()
```

Predictions and Evaluation of Decision Tree

```
[15]: predictions = dtree.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

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

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.84      2431
           1       0.19      0.23      0.21       443

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

```
[15]: array([[1996,  435],
             [ 340,  103]], dtype=int64)
```

Training the Random Forest model

[17]:
```python
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=200)

rfc.fit(X_train, y_train)
```

[17]:
```
RandomForestClassifier(n_estimators=200)
```

Predictions and Evaluation

[18]:
```python
pred = rfc.predict(X_test)

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

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2431
           1       0.53      0.02      0.04       443

    accuracy                           0.85      2874
   macro avg       0.69      0.51      0.48      2874
weighted avg       0.80      0.85      0.78      2874
```

[18]:
```
array([[2422,    9],
       [ 433,   10]], dtype=int64)
```

Random Forest gives better accuracy than Decision Trees