PeerLoanKart is an NBFC (Non-banking Financial Company) that facilitates peer- to-peer loan. It connects people who need money (borrowers) with people who have money (investors). As an investor, you would want to invest in people who have a high probability of paying you back. You as an ML expert create a model that will help predict whether a borrower will pay the loan or not.

Objective: Increase profits up to 20% as NPA will be reduced due to loan disbursal for only creditworthy borrowers.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
```

In [2]:
```
loans = pd.read_csv('loan_borowwer_data.csv')
```

In [3]:
```
loans.describe()
```

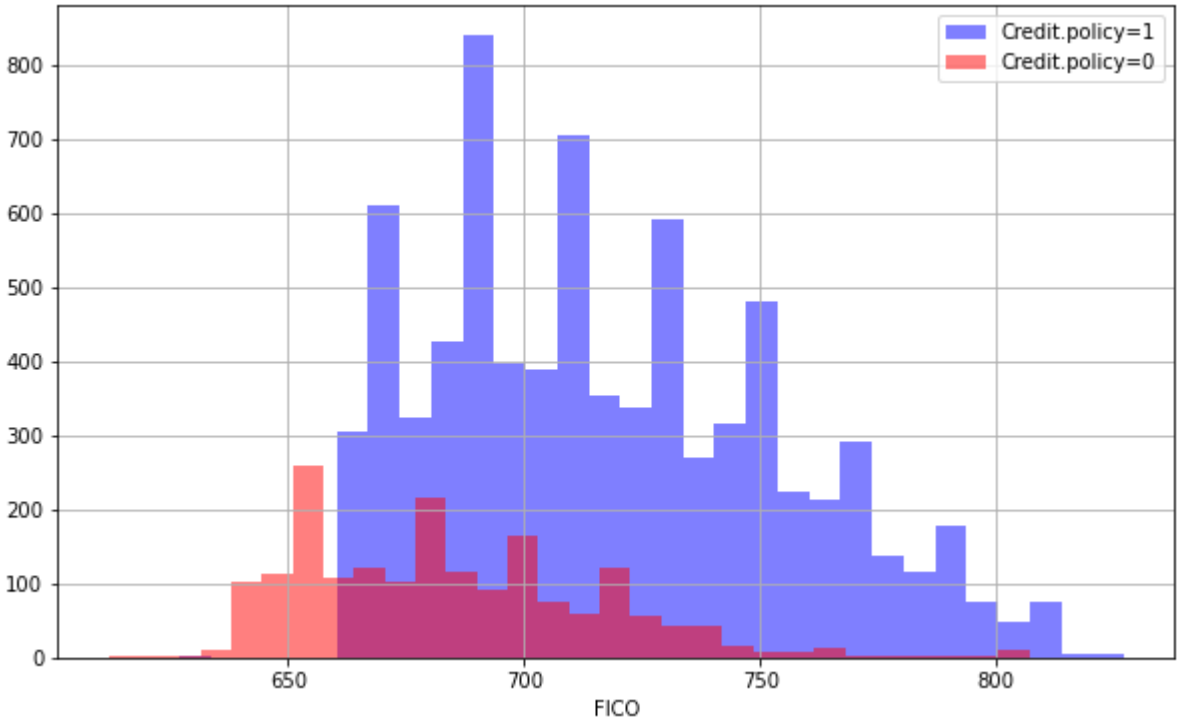
Out[3]:

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

In [7]:
```
#Create histogram of two FICO distribuitions 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')
```

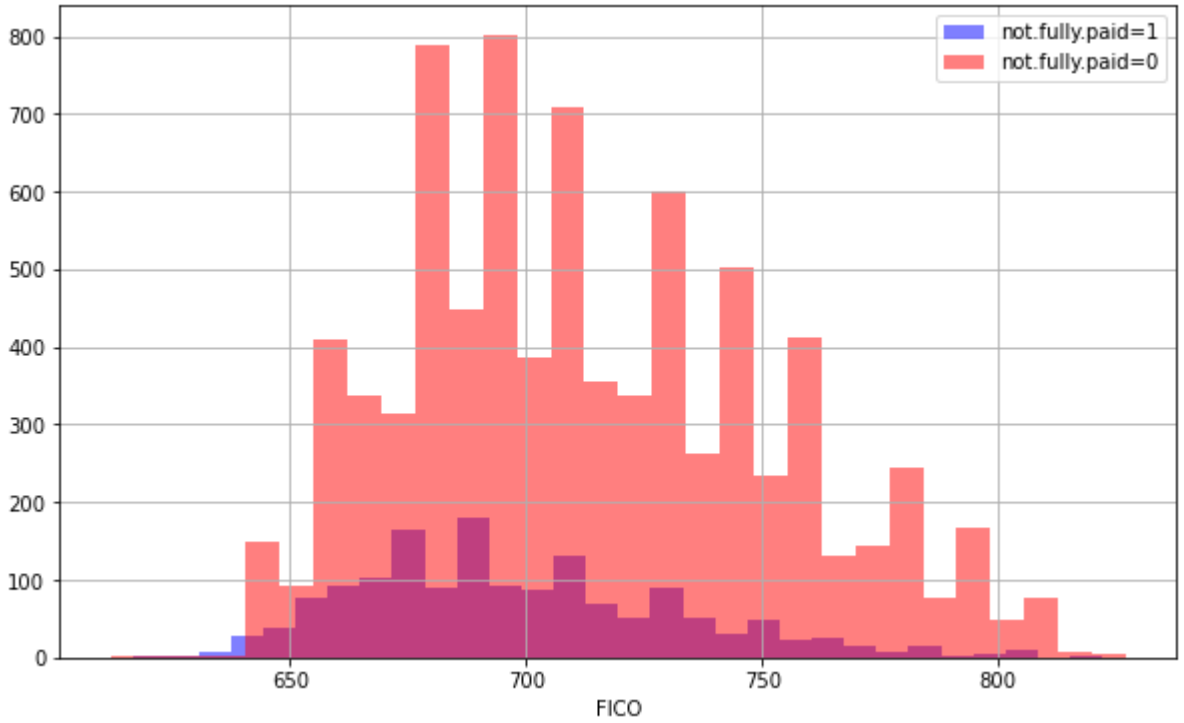
Out[7]:
```
Text(0.5, 0, 'FICO')
```



In [9]:
```
#Histogram for the not.fully.paid column

plt.figure(figsize=(10,6))
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='blue',bins=30,label='not.fully.paid=1')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='red',bins=30,label='not.fully.paid=0')
plt.legend()
plt.xlabel('FICO')
```

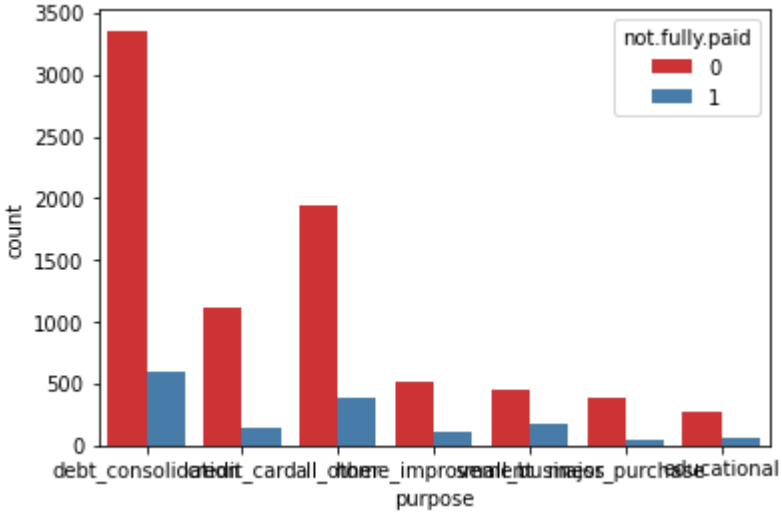
Out[9]:
```
Text(0.5, 0, 'FICO')
```



In [11]:
```
#create countplot in seaborn showing counts of loans by purpose, with hue defined by not.fully.paid

sns.countplot(x='purpose',hue='not.fully.paid',data=loans,palette='Set1')
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x28976ed0670>



In [12]:
```
#Create a list of elements, containing the string "purpose". Call this list cat_feats

cat_feats =['purpose']
```

In [13]:
```
#Now use pd.get_dummies(loans,columns=cat_feats,drop_first=True) to create a fixed larger data
#frame that has new feature columns with dummy variables. Set this data frame as final_data

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
```

In [19]:
```
#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)
```

In [20]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(x_train,y_train)
```

Out[20]:
```
DecisionTreeClassifier()
```

In [23]:
```
#Create predictions from the test set, and create a classification report and a confusion matrix

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

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

In [24]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[1992  439]
 [ 343  100]]
```

In [26]:
```
#Random Forest

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=600)
rfc.fit(x_train,y_train)
```

Out[26]:
```
RandomForestClassifier(n_estimators=600)
```

In [29]:
```
# Evaluating Random forest

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

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

In [30]:
```
# Printing confussion Matrix
print(confusion_matrix(y_test,predictions))
```

```
[[1992  439]
 [ 343  100]]
```

In [ ]:
```

```