Decision Tree Case Study

September 2, 2023

1 Problem statement:

PeerLoanKart is an NBFC (non-banking financial company) that facilitates peer-to-peer loans. It connects people who need money (borrowers) with people who have money (investors). As an investor, you would want to invest in people who showed a profile of having a high probability of paying you back. Create a model that will help predict whether a borrower will repay the loan. # Analysis to be done:

Increase profits by up to 20% as NPAs will be reduced due to loan disbursal to creditworthy borrowers only

[1]:
```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

[3]:
```
loan = pd.read_csv('loan_borowwer_data.csv')
loan.head()
```

[3]:
```
   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
```

[4]:
```
# lets first do the understanding of data
loan.columns
```

[4]:
```
Index(['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.paid'],
      dtype='object')
```

[6]:
```
loan.shape
```

[6]:
```
(9578, 14)
```

[8]:
```
loan.duplicated().value_counts()
```

[8]:
```
False    9578
dtype: int64
```

[10]:
```
loan.describe()
```

[10]:
```
       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
```

[11]:
```
loan.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
```

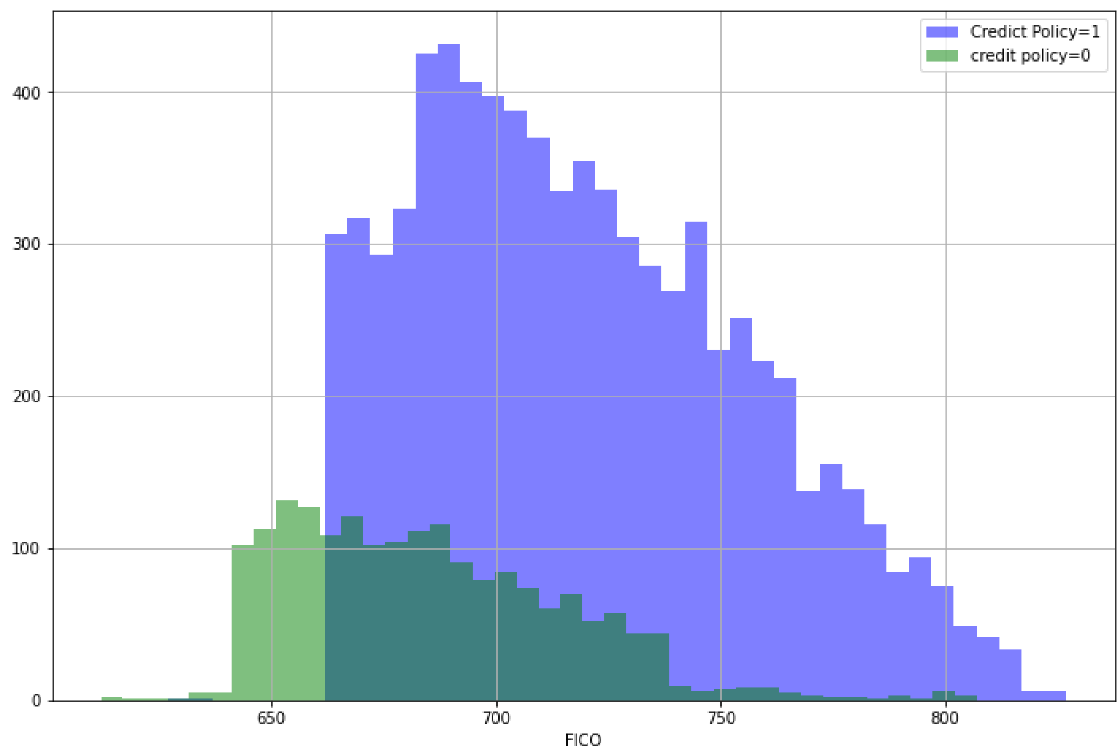
[12]:
```
loan.dtypes
```

[12]:
```
credit.policy          int64
purpose               object
int.rate             float64
installment          float64
log.annual.inc       float64
dti                  float64
fico                   int64
days.with.cr.line    float64
revol.bal              int64
revol.util           float64
inq.last.6mths         int64
delinq.2yrs            int64
pub.rec                int64
not.fully.paid         int64
dtype: object
```

Interpretation: we can see that purpose column is categorical which needs to convertin numerical

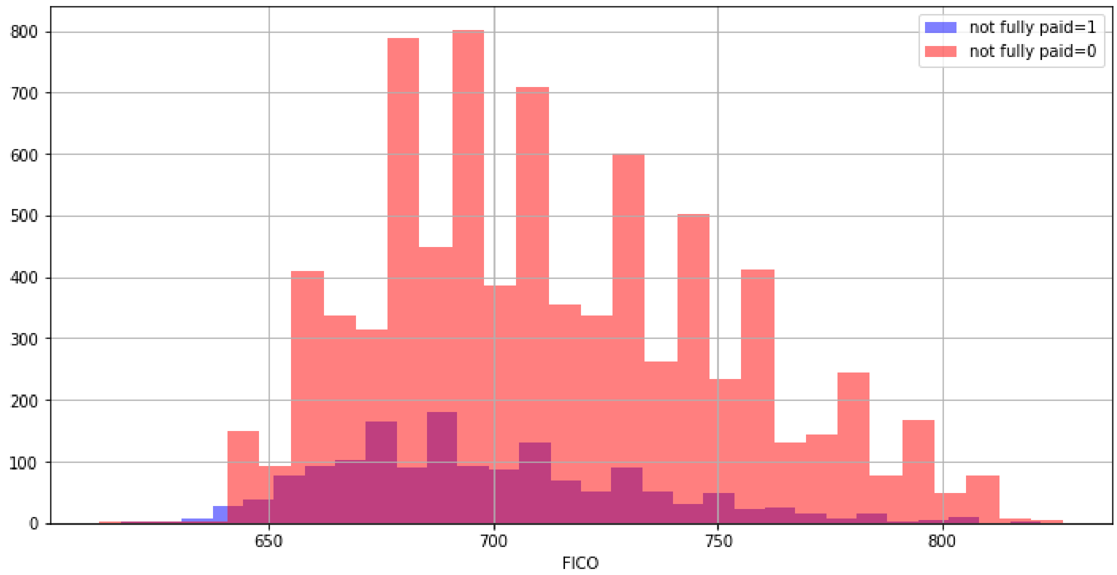
[20]:
```
# lets apply EDA(Exploratory Data Analysis )
plt.figure(figsize=(12,8))
loan[loan['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue', bins =40,␣
 ↪label="Credict Policy=1")
loan[loan['credit.policy']==0]['fico'].hist(alpha=0.5,bins=40,␣
 ↪color='green',label="credit policy=0")
plt.legend()
plt.xlabel('FICO')
```

[20]: Text(0.5, 0, 'FICO')



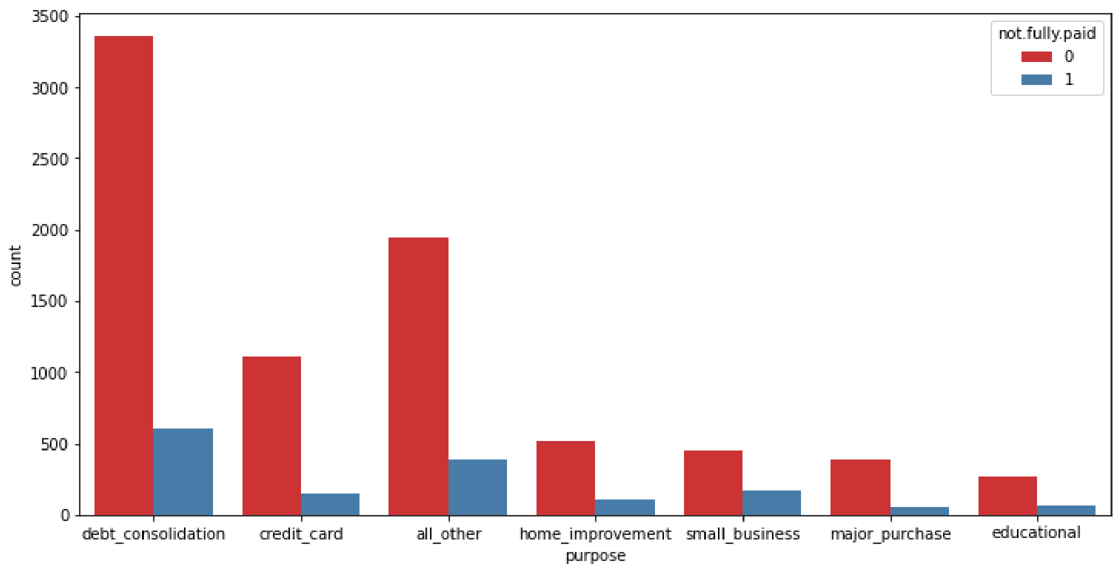
[26]:
```
plt.figure(figsize=(12,6))
loan[loan['not.fully.paid']==1]['fico'].hist(alpha=0.
 ↪5,color='blue',bins=30,label="not fully paid=1")

loan[loan['not.fully.paid']==0]['fico'].hist(alpha=0.
 ↪5,color='red',bins=30,label="not fully paid=0")
plt.legend()
plt.xlabel("FICO")
```

[26]: Text(0.5, 0, 'FICO')



[27]:
```
#Create the coutplot
plt.figure(figsize=(12,6))
sns.countplot(x='purpose', hue='not.fully.paid',data=loan,palette='Set1')
```

[27]: `<AxesSubplot: xlabel='purpose', ylabel='count'>`



lets perform the encoding to column purpose for categorical to numerical data

[29]:
```
loan['purpose'].value_counts()
```

[29]:
```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

[30]:
```
# lets use pd.get_dummies for all above categories
col=['purpose']
final_data =pd.get_dummies(loan,columns=col,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
```

Interpretation: here we use final data as the new data name, therefore it wont affect the original data loan, we will fed the dummies of purpose column to the new data.

2 Train Test the data Set (Evaluation of Model)

[31]:
```
from sklearn.model_selection import train_test_split
```

[32]:
```
X= final_data.drop('not.fully.paid',axis=1)
y =final_data['not.fully.paid']
```

[33]:
```
print('X shape:',X.shape)
print('y shape:',y.shape)
```

```
X shape: (9578, 18)
y shape: (9578,)
```

[34]:
```
x_train,x_test,y_train,y_test =train_test_split(X,y,test_size=0.
↪3,random_state=23)
```

[35]:
```
print('x_train shape:',x_train.shape)
print('x_test shape:',x_test.shape)
print('y_train shape:',y_train.shape)
print('y_test shape:',y_test.shape)
```

```
x_train shape: (6704, 18)
x_test shape: (2874, 18)
y_train shape: (6704,)
y_test shape: (2874,)
```

3 Training the Model Using Decision Tree with different function of Dicision Tress.

[39]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, f1_score, confusion_matrix,␣
↪precision_score,accuracy_score
from sklearn.model_selection import cross_val_score, KFold
```

4 Lets try the criterion{"gini"} to understand any accuracy change

[72]:
```
DT_gini =DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=7,␣
↪random_state=50, max_leaf_nodes=100, min_impurity_decrease=0.0)
DT_gini
```

[72]:
```
DecisionTreeClassifier(max_depth=7, max_leaf_nodes=100, random_state=50)
```

[73]:
```
# lets fit the data
DT_gini.fit(x_train,y_train)
```

[73]: DecisionTreeClassifier(max_depth=7, max_leaf_nodes=100, random_state=50)

[74]:
```
predict_gini =DT_gini.predict(x_test)
predict_gini
```

[74]: array([0, 0, 0, …, 0, 0, 0])

[75]:
```
print(classification_report(y_test,predict_gini))
print(accuracy_score(y_test,predict_gini))
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      2420
           1       0.32      0.10      0.15       454

    accuracy                           0.82      2874
   macro avg       0.59      0.53      0.53      2874
weighted avg       0.77      0.82      0.78      2874

0.824634655532359
```

[76]:
```
Acc_gini= accuracy_score(y_test,predict_gini)
Acc_gini
```

[76]: 0.824634655532359

[81]:
```
print(confusion_matrix(y_test,predict_gini))
```

```
[[2325   95]
 [ 409   45]]
```

[86]:
```
# calculate the cross validation:
kfold_gini =KFold(n_splits =5,random_state=40,shuffle= True)
```

[90]:
```
cross_gini= cross_val_score(DT_gini,X,y,cv=kfold_gini,scoring='accuracy')
print(cross_gini.mean()*100)
```

```
82.67920548139345
```

Lets try the criterion{" "entropy"} to understand any accuracy changecriterion

[43]:
```
DT_entropy =DecisionTreeClassifier(criterion='entropy', splitter='best',␣
↪max_depth=7, random_state=50, max_leaf_nodes=100, min_impurity_decrease=0.0)
DT_entropy
```

[43]: DecisionTreeClassifier(criterion='entropy', max_depth=7, max_leaf_nodes=100,
random_state=50)

[46]:
```
# lets fit the data
DT_entropy.fit(x_train,y_train)
```

[46]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=7, max_leaf_nodes=100,
                       random_state=50)
```

[49]:
```
predict_entropy =DT_entropy.predict(x_test)
predict_entropy
```

[49]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[58]:
```
print(classification_report(y_test,predict_entropy))
```

```
              precision    recall  f1-score   support

           0       0.84      0.98      0.91      2420
           1       0.23      0.03      0.05       454

    accuracy                           0.83      2874
   macro avg       0.54      0.51      0.48      2874
weighted avg       0.75      0.83      0.77      2874
```

[62]:
```
print(accuracy_score(y_test,predict_entropy))
```

```
0.8312456506610996
```

[67]:
```
Acc_entropy= accuracy_score(y_test,predict_entropy)
Acc_entropy
```

[67]:
```
0.8312456506610996
```

[80]:
```
print(confusion_matrix(y_test,predict_entropy))
```

```
[[2376   44]
 [ 441   13]]
```

[92]:
```
# calculate the cross validation:
kfold_entropy =KFold(n_splits =5,random_state=40,shuffle= True)
```

[93]:
```
cross_entropy=␣
 ↪cross_val_score(DT_entropy,X,y,cv=kfold_entropy,scoring='accuracy')
print(cross_entropy.mean()*100)
```

```
83.34734569953723
```

[ ]:
```

```

Lets try the criterion{“ “log_loss”} to understand any accuracy changecriterion

[44]:
```
DT_log_loss =DecisionTreeClassifier(criterion='log_loss', splitter='best',␣
 ↪max_depth=7, random_state=50, max_leaf_nodes=100, min_impurity_decrease=0.0)
DT_log_loss
```

[44]:
```
DecisionTreeClassifier(criterion='log_loss', max_depth=7, max_leaf_nodes=100,
                       random_state=50)
```

[47]:
```
# lets fit the data
DT_log_loss.fit(x_train,y_train)
```

[47]:
```
DecisionTreeClassifier(criterion='log_loss', max_depth=7, max_leaf_nodes=100,
                       random_state=50)
```

[57]:
```
predict_log_loss =DT_log_loss.predict(x_test)
predict_log_loss
```

[57]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[63]:
```
print(classification_report(y_test,predict_log_loss))
```

```
              precision    recall  f1-score   support

           0       0.84      0.98      0.91      2420
           1       0.23      0.03      0.05       454

    accuracy                           0.83      2874
   macro avg       0.54      0.51      0.48      2874
weighted avg       0.75      0.83      0.77      2874
```

[64]:
```
print(accuracy_score(y_test,predict_log_loss))
```

```
0.8312456506610996
```

[68]:
```
Acc_log_loss= accuracy_score(y_test,predict_log_loss)
Acc_log_loss
```

[68]:
```
0.8312456506610996
```

[79]:
```
print(confusion_matrix(y_test,predict_log_loss))
```

```
[[2376   44]
 [ 441   13]]
```

[95]:
```
# calculate the cross validation:
kfold_log_loss =KFold(n_splits =5,random_state=40,shuffle= True)
```

[96]:
```
cross_logloss␣
↪=cross_val_score(DT_log_loss,X,y,cv=kfold_log_loss,scoring='accuracy')
print(cross_logloss.mean()*100)
```

```
83.34734569953723
```

[65]:
```
#Make a tabulate formate
from tabulate import tabulate
```

[71]:
```
prediction_table = pd.DataFrame(columns=["Actual vlue","Gini Accuracy Value",␣
↪"Entropy Accuracy Value","Log Loss Accuracy Vaue"])
prediction_table["Actual Value"] = y_test

prediction_table["Gini Accuracy Value"] = Acc_gini
prediction_table["Entropy Accuracy Value"] = Acc_entropy
prediction_table["Log Loss Accuracy Value"] = Acc_log_loss

print(tabulate(prediction_table.head(10), headers = 'keys', tablefmt = 'psql',␣
↪numalign="left"))
```

```
+------+---------------+----------------------+--------------------------+-----
---------------------+----------------+--------------------------+
|      | Actual vlue   | Gini Accuracy Value  | Entropy Accuracy Value   | Log
Loss Accuracy Vaue   | Actual Value   | Log Loss Accuracy Value  |
|------+---------------+----------------------+--------------------------+-----
---------------------+----------------+--------------------------|
| 3409 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 8289 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 9293 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 3839 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 9050 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 540  | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 8610 | nan           | 0.738692             | 0.831246                 | nan
| 1              | 0.831246                 |
| 3410 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 9437 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
| 3150 | nan           | 0.738692             | 0.831246                 | nan
| 0              | 0.831246                 |
+------+---------------+----------------------+--------------------------+-----
```

```
---------------------+----------------+---------------------------+
```

5 Conclusion:

Decision Tree Accuracy Using GINI= 82% Decision Tree Accuracy Using ENTROPY= 83% Decision Tree Accuracy Using LOG_LOSS=83% cross_gini:82% cross_entropy:83% cross_logloss:83% Hence the model is working perfectly well. it shows that there is no cause of overfitting in the dataset and entropy and logloss creteria hs given a slight more model accuracy as compared to gini creteria. PeerLoan Kart shows 83% chances to repay the loan.

[ ]:

```

```