Problem Statement

Based on the given loan data can we understand the major factors or characteristics of a borrower which makes them to get into delinquent stage.

• Delinquency is a major metric in assessing risk, as more and more customers getting delinquent means the risk of customers that will default will also increase.

• The main objective is to minimize the risk for which you need to build a decision tree model using CART technique that will identify various risk and non-risk attributes of borrower's to get into delinquent stage

width = 400, height = 400

Importing libraries and Loading data

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

ld_df = pd.read_csv("Loan Delinquent Dataset.csv")
```

Checking the data

```
ld_df.head()

   ID delinquent  Sdelinquent       term  gender purpose 
home_ownership  \
0   1        Yes            1  36 months  Female   House 
Mortgage   
1   2         No            0  36 months  Female   House 
Rent   
2   3        Yes            1  36 months  Female   House 
Rent   
3   4        Yes            1  36 months  Female     Car 
Mortgage   
4   5        Yes            1  36 months  Female   House 
Rent   

     age     FICO  
0    >25  300-500  
1  20-25     >500  
2    >25  300-500  
3    >25  300-500  
4    >25  300-500  
```

Dropping unwanted variables

Sdelinquent can also be dropped instead of delinquent.

```
ld_df=ld_df.drop(["ID","delinquent"],axis=1)
```

```
ld_df.head()
```

```
   Sdelinquent       term  gender purpose home_ownership    age
FICO
0            1  36 months  Female   House       Mortgage    >25  300-
500
1            0  36 months  Female   House           Rent  20-25
>500
2            1  36 months  Female   House           Rent    >25  300-
500
3            1  36 months  Female     Car       Mortgage    >25  300-
500
4            1  36 months  Female   House           Rent    >25  300-
500
```

```
ld_df.shape
```

```
(11548, 7)
```

```
ld_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11548 entries, 0 to 11547
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sdelinquent     11548 non-null  int64
 1   term            11548 non-null  object
 2   gender          11548 non-null  object
 3   purpose         11548 non-null  object
 4   home_ownership  11548 non-null  object
 5   age             11548 non-null  object
 6   FICO            11548 non-null  object
dtypes: int64(1), object(6)
memory usage: 631.7+ KB
```

many columns are of type object i.e. strings. These need to be converted to ordinal type

Geting unique counts of all Objects

```
print('term \n',ld_df.term.value_counts())
print('\n')
print('gender \n',ld_df.gender.value_counts())
print('\n')
print('purpose \n',ld_df.purpose.value_counts())
print('\n')
print('home_ownership \n',ld_df.home_ownership.value_counts())
```

```
print('\n')
print('age \n',ld_df.age.value_counts())
print('\n')
print('FICO \n',ld_df.FICO.value_counts())
```

```
term
 36 months    10589
60 months      959
Name: term, dtype: int64


gender
 Male      6555
Female    4993
Name: gender, dtype: int64


purpose
 House       6892
Car         2080
Other        928
Personal     892
Wedding      408
Medical      266
other         82
Name: purpose, dtype: int64


home_ownership
 Mortgage    5461
Rent        5216
Own          871
Name: home_ownership, dtype: int64


age
 20-25    5888
>25      5660
Name: age, dtype: int64


FICO
 300-500    6370
>500       5178
Name: FICO, dtype: int64
```

Note:

Decision tree in Python can take only numerical / categorical colums. It cannot take string / object types.

The following code loops through each column and checks if the column type is object then converts those columns into categorical with each distinct value becoming a category.

```
for feature in ld_df.columns:
    if ld_df[feature].dtype == 'object':
        print('\n')
        print('feature:',feature)
        print(pd.Categorical(ld_df[feature].unique()))
        print(pd.Categorical(ld_df[feature].unique()).codes)
        ld_df[feature] = pd.Categorical(ld_df[feature]).codes
```

```


feature: term
[36 months, 60 months]
Categories (2, object): [36 months, 60 months]
[0 1]


feature: gender
[Female, Male]
Categories (2, object): [Female, Male]
[0 1]


feature: purpose
[House, Car, Other, Personal, Wedding, Medical, other]
Categories (7, object): [Car, House, Medical, Other, Personal,
Wedding, other]
[1 0 3 4 5 2 6]


feature: home_ownership
[Mortgage, Rent, Own]
Categories (3, object): [Mortgage, Own, Rent]
[0 2 1]


feature: age
[>25, 20-25]
Categories (2, object): [20-25, >25]
[1 0]


feature: FICO
[300-500, >500]
Categories (2, object): [300-500, >500]
[0 1]
```

For each feature, look at the 2nd and 4th row to get the encoding mappings. Do not look at the line starting with 'Categories'

Comparing the unique counts from above

```
print('term \n',ld_df.term.value_counts())
print('\n')
print('gender \n',ld_df.gender.value_counts())
print('\n')
print('purpose \n',ld_df.purpose.value_counts())
print('\n')
print('home_ownership \n',ld_df.home_ownership.value_counts())
print('\n')
print('age \n',ld_df.age.value_counts())
print('\n')
print('FICO \n',ld_df.FICO.value_counts())
```

```
term
 0    10589
1      959
Name: term, dtype: int64


gender
 1    6555
0    4993
Name: gender, dtype: int64


purpose
 1    6892
0    2080
3     928
4     892
5     408
2     266
6      82
Name: purpose, dtype: int64


home_ownership
 0    5461
2    5216
1     871
Name: home_ownership, dtype: int64


age
 0    5888
1    5660
```

```
Name: age, dtype: int64
```

```
FICO
 0    6370
1    5178
Name: FICO, dtype: int64
```

```
ld_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11548 entries, 0 to 11547
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sdelinquent     11548 non-null  int64
 1   term            11548 non-null  int8
 2   gender          11548 non-null  int8
 3   purpose         11548 non-null  int8
 4   home_ownership  11548 non-null  int8
 5   age             11548 non-null  int8
 6   FICO            11548 non-null  int8
dtypes: int64(1), int8(6)
memory usage: 158.0 KB
```

```
ld_df.head()
```

```
   Sdelinquent  term  gender  purpose  home_ownership  age  FICO
0            1     0       0        1               0    1     0
1            0     0       0        1               2    0     1
2            1     0       0        1               2    1     0
3            1     0       0        0               0    1     0
4            1     0       0        1               2    1     0
```

Label Encoding has been done and all columns are converted to number

Proportion of 1s and 0s

```
ld_df.Sdelinquent.value_counts(normalize=True)
```

```
1    0.668601
0    0.331399
Name: Sdelinquent, dtype: float64
```

```
print(ld_df.Sdelinquent.value_counts())
print('%1s = ',7721/(7721+3827)*100)
print('%0s = ',3827/(7721+3827)*100)
```

```
1    7721
0    3827
Name: Sdelinquent, dtype: int64
%1s =  66.8600623484586
%0s =  33.13993765154139
```

Extracting the target column into separate vectors for training set and test set

```
X = ld_df.drop("Sdelinquent", axis=1)

y = ld_df.pop("Sdelinquent")

X.head()
```

```
   term  gender  purpose  home_ownership  age  FICO
0     0       0        1               0    1     0
1     0       0        1               2    0     1
2     0       0        1               2    1     0
3     0       0        0               0    1     0
4     0       0        1               2    1     0
```

Splitting data into training and test set

```
from sklearn.model_selection import train_test_split

X_train, X_test, train_labels, test_labels = train_test_split(X, y,
test_size=.30, random_state=1)
```

Checking the dimensions of the training and test data

```
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('train_labels',train_labels.shape)
print('test_labels',test_labels.shape)
print('Total Obs',8083+3465)
```

```
X_train (8083, 6)
X_test (3465, 6)
train_labels (8083,)
test_labels (3465,)
Total Obs 11548
```

Building a Decision Tree Classifier

```
# Initialise a Decision Tree Classifier
dt_model = DecisionTreeClassifier(criterion = 'gini', random_state=1)

# Fit the model
dt_model.fit(X_train, train_labels)
```

```
DecisionTreeClassifier(random_state=1)
```

```
from sklearn import tree

train_char_label = ['No', 'Yes']
ld_Tree_File = open('ld_Tree_File.dot','w')
dot_data = tree.export_graphviz(dt_model,
                                out_file=ld_Tree_File,
                                feature_names = list(X_train),
                                class_names = list(train_char_label))
```

```
ld_Tree_File.close()
```

The above code will save a .dot file in your working directory.
WebGraphviz is Graphviz in the Browser.
Copy paste the contents of the file into the link below to get the visualization
http://webgraphviz.com/

Variable Importance

```
print (pd.DataFrame(dt_model.feature_importances_, columns = ["Imp"],
index = X_train.columns).sort_values('Imp',ascending=False))
```

```
                     Imp
FICO            0.393915
term            0.370052
gender          0.158664
age             0.055813
purpose         0.010924
home_ownership  0.010633
```

Predicting Test Data

```
y_predict = dt_model.predict(X_test)
```

```
y_predict.shape
```

```
(3465,)
```

Regularising the Decision Tree

Adding Tuning Parameters

```
reg_dt_model = DecisionTreeClassifier(criterion = 'gini', max_depth =
30,min_samples_leaf=100,min_samples_split=1000,random_state=1)
reg_dt_model.fit(X_train, train_labels)
```

```
DecisionTreeClassifier(max_depth=30, min_samples_leaf=100,
                       min_samples_split=1000, random_state=1)
```

Generating New Tree

```
ld_tree_regularized = open('ld_tree_regularized.dot','w')
dot_data = tree.export_graphviz(reg_dt_model, out_file=
ld_tree_regularized , feature_names = list(X_train), class_names =
list(train_char_label))

ld_tree_regularized.close()
dot_data
```

Variable Importance

```
print (pd.DataFrame(reg_dt_model.feature_importances_, columns =
["Imp"], index = X_train.columns).sort_values('Imp',ascending=False))
```

```
                     Imp
FICO            0.424274
term            0.339141
gender          0.169328
age             0.060114
home_ownership  0.003659
purpose         0.003483
```

Predicting on Training and Test dataset

```
ytrain_predict = reg_dt_model.predict(X_train)
ytest_predict = reg_dt_model.predict(X_test)
```

```
print('ytrain_predict',ytrain_predict.shape)
print('ytest_predict',ytest_predict.shape)
```

```
ytrain_predict (8083,)
ytest_predict (3465,)
```

Getting the Predicted Classes

```
ytest_predict
```

```
array([1, 1, 0, ..., 1, 1, 1], dtype=int64)
```

Getting the Predicted Probabilities

```
ytest_predict_prob=reg_dt_model.predict_proba(X_test)
ytest_predict_prob
```

```
array([[0.12938331, 0.87061669],
       [0.        , 1.        ],
       [0.75366876, 0.24633124],
       ...,
       [0.        , 1.        ],
       [0.16071429, 0.83928571],
       [0.22881356, 0.77118644]])
```

```
 pd.DataFrame(ytest_predict_prob).head()
```

```
          0         1
0  0.129383  0.870617
1  0.000000  1.000000
2  0.753669  0.246331
3  0.753669  0.246331
4  0.136986  0.863014
```

Model Evaluation

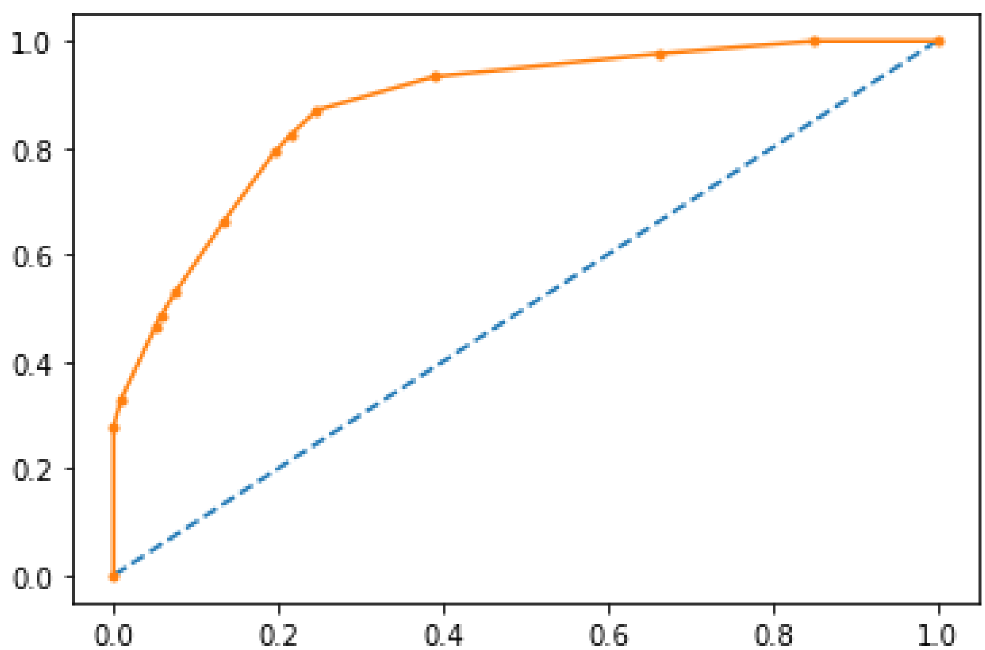
Measuring AUC-ROC Curve

```
import matplotlib.pyplot as plt
```

AUC and ROC for the training data

```
# predict probabilities
probs = reg_dt_model.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(train_labels, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(train_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

```
AUC: 0.879
```

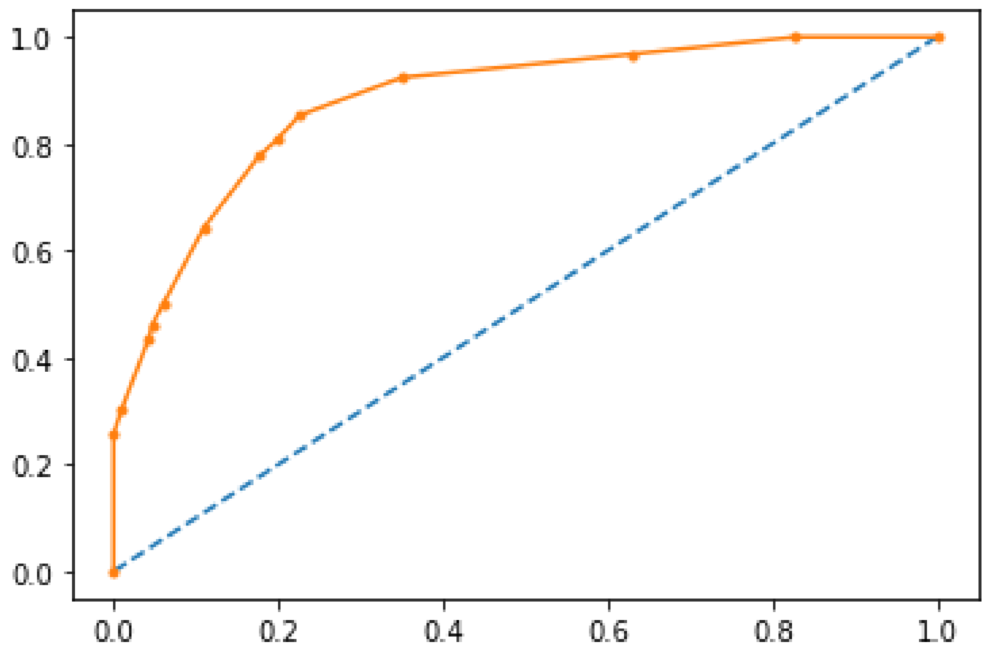


AUC and ROC for the test data

```
# predict probabilities
probs = reg_dt_model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(test_labels, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
```

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(test_labels, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

```
AUC: 0.881
```



Confusion Matrix for the training data

```
from sklearn.metrics import classification_report,confusion_matrix

confusion_matrix(train_labels, ytrain_predict)
```

```
array([[1985,  650],
       [ 706, 4742]], dtype=int64)
```

```
#Train Data Accuracy
reg_dt_model.score(X_train,train_labels)
```

```
0.832240504763083
```

```
print((1985+4742)/(1985+650+706+4742))
```

```
0.832240504763083
```

```
print(classification_report(train_labels, ytrain_predict))
```

```
              precision    recall  f1-score   support
```

```
           0       0.74      0.75      0.75      2635
           1       0.88      0.87      0.87      5448

    accuracy                           0.83      8083
   macro avg       0.81      0.81      0.81      8083
weighted avg       0.83      0.83      0.83      8083
```

Confusion Matrix for test data

```
confusion_matrix(test_labels, ytest_predict)
```

```
array([[ 922,  270],
       [ 332, 1941]], dtype=int64)
```

```
#Test Data Accuracy
reg_dt_model.score(X_test,test_labels)
```

```
0.8262626262626263
```

```
print((922+1941)/(922+270+332+1941))
```

```
0.8262626262626263
```

```
print(classification_report(test_labels, ytest_predict))
```

```
              precision    recall  f1-score   support

           0       0.74      0.77      0.75      1192
           1       0.88      0.85      0.87      2273

    accuracy                           0.83      3465
   macro avg       0.81      0.81      0.81      3465
weighted avg       0.83      0.83      0.83      3465
```

Conclusion

Accuracy on the Training Data: 83%
Accuracy on the Test Data: 82%

AUC on the Training Data: 87.9%
AUC on the Test: 88.1%

Accuracy, AUC, Precision and Recall for test data is almost inline with training data. This proves no overfitting or underfitting has happened, and overall the model is a good model for classification

FICO, term and gender (in same order of preference) are the most important variables in determining if a borrower will get into a delinquent stage

End