# Python Machine Learning - Predicting the Loan Approval Status

This dataset is about past loans. The Loan_train.csv data set includes details of 346 customers whose loan are already paid off or defaulted. It includes following fields:

```
Loan_status - Whether a loan is paid off on in collection
Principal - Basic principal loan amount at the
Terms - Origination terms which can be weekly (7 days), biweekly, and monthly payoff s
chedule
Effective_date - When the loan got originated and took effects
Due_date - Since it's one-time payoff schedule, each loan has one single due date
Age - Age of applicant
Education - Education of applicant
Gender - The gender of applicant
```

In [589]:

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

## Load Data From CSV File

In [590]:

```
df = pd.read_csv('U:/Documents/Gunn Notes/Data Analyst Training/Machine_Learning_w
ith_Python/Loan_train.csv')
df.head()
```

Out[590]:

| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | PAIDOFF | 1000 | 30 | 9/8/2016 | 10/7/2016 | 45 | High School or Below | male |
| 1 | 2 | 2 | PAIDOFF | 1000 | 30 | 9/8/2016 | 10/7/2016 | 33 | Bechalor | female |
| 2 | 3 | 3 | PAIDOFF | 1000 | 15 | 9/8/2016 | 9/22/2016 | 27 | college | male |
| 3 | 4 | 4 | PAIDOFF | 1000 | 30 | 9/9/2016 | 10/8/2016 | 28 | college | female |
| 4 | 6 | 6 | PAIDOFF | 1000 | 30 | 9/9/2016 | 10/8/2016 | 29 | college | male |

In [591]:

```
df.shape
```

Out[591]: (346, 10)

## Convert to date time object

In [592]:
```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

Out[592]:

| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | male |
| 1 | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | female |
| 2 | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | male |
| 3 | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | female |
| 4 | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | male |

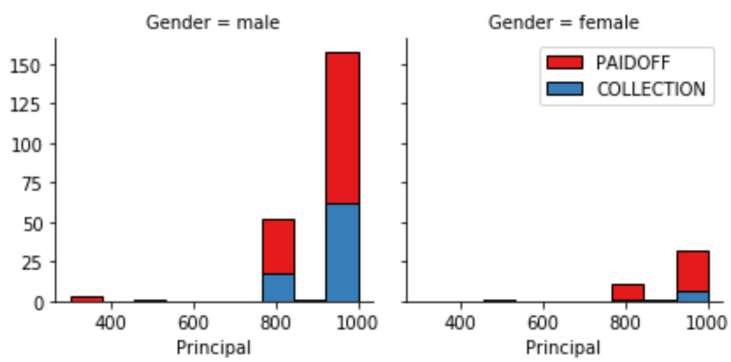
# Data visualization and pre-processing

In [593]:
```
# see how many of each class is in our data set
df['loan_status'].value_counts()
```

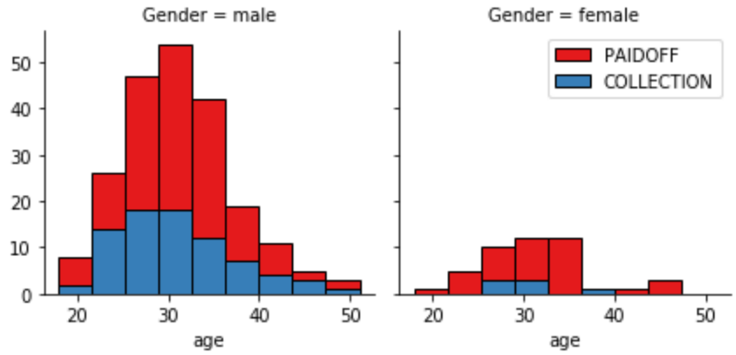
Out[593]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

260 people have paid off the loan on time while 86 have gone into collection

In [594]:
```
import seaborn as sns

bins = np.linspace(df.Principal.min(), df.Principal.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'Principal', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```

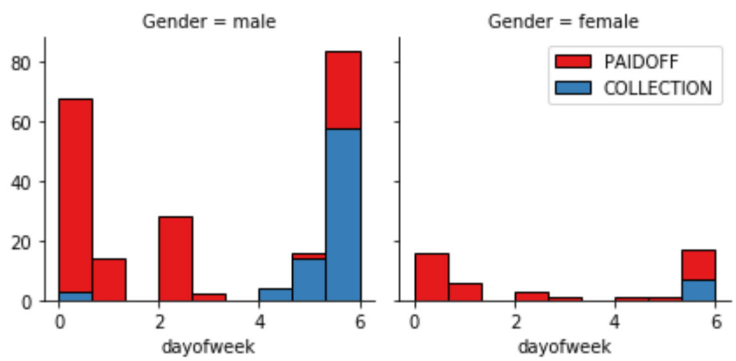


In [595]:
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```




## Pre-processing: Feature selection/extraction

In [596]:
```
# Lets look at the day of the week people get the loan
df['dayofweek'] = df['effective_date'].dt.dayofweek
bins = np.linspace(df.dayofweek.min(), df.dayofweek.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'dayofweek', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```




We see that people who get the loan at the end of the week dont pay it off, so lets use Feature binarization to set a threshold values less then day 4

In [597]:
```
# use Feature binarization to set a threshold values less then day 4
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

Out[597]:

| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender | da |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | male | |
| 1 | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | female | |
| 2 | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | male | |
| 3 | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | female | |
| 4 | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | male | |

## Convert Categorical features to numerical values

In [598]:
```
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[598]:
```
Gender  loan_status
female  PAIDOFF        0.865385
        COLLECTION     0.134615
male    PAIDOFF        0.731293
        COLLECTION     0.268707
Name: loan_status, dtype: float64
```

86 % of female pay there loans while only 73 % of males pay there loan

Lets convert male to 0 and female to 1:

In [599]:
```
df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
df.head()
```

Out[599]:

| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender | da |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | 0 | |
| 1 | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | 1 | |
| 2 | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | 0 | |
| 3 | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | 1 | |
| 4 | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | 0 | |

# One Hot Encoding

check education

In [600]:
```
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[600]:
```
education             loan_status
Bechalor              PAIDOFF        0.750000
                      COLLECTION     0.250000
High School or Below  PAIDOFF        0.741722
                      COLLECTION     0.258278
Master or Above       COLLECTION     0.500000
                      PAIDOFF        0.500000
college               PAIDOFF        0.765101
                      COLLECTION     0.234899
Name: loan_status, dtype: float64
```

## Feature before One Hot Encoding

In [601]:
```
df[['Principal','terms','age','Gender','weekend','education']].head()
```

Out[601]:

| | Principal | terms | age | Gender | weekend | education |
|---|---|---|---|---|---|---|
| 0 | 1000 | 30 | 45 | 0 | 0 | High School or Below |
| 1 | 1000 | 30 | 33 | 1 | 0 | Bechalor |
| 2 | 1000 | 15 | 27 | 0 | 0 | college |
| 3 | 1000 | 30 | 28 | 1 | 1 | college |
| 4 | 1000 | 30 | 29 | 0 | 1 | college |

## Use one hot encoding technique to conver categorical variables to binary variables and append them to the feature Data Frame

In [602]:
```
Feature = df[['Principal','terms','age','Gender','weekend']]
Feature = pd.concat([Feature,pd.get_dummies(df['education'])], axis=1)
Feature.drop(['Master or Above'], axis = 1,inplace=True)
Feature.head()
```

Out[602]:

| | Principal | terms | age | Gender | weekend | Bechalor | High School or Below | college |
|---|---|---|---|---|---|---|---|---|
| 0 | 1000 | 30 | 45 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1000 | 30 | 33 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1000 | 15 | 27 | 0 | 0 | 0 | 0 | 1 |
| 3 | 1000 | 30 | 28 | 1 | 1 | 0 | 0 | 1 |
| 4 | 1000 | 30 | 29 | 0 | 1 | 0 | 0 | 1 |

## Feature selection

defind feature sets, X

In [603]:
```
X = Feature
X[0:5]
```

Out[603]:

| | Principal | terms | age | Gender | weekend | Bechalor | High School or Below | college |
|---|---|---|---|---|---|---|---|---|
| 0 | 1000 | 30 | 45 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1000 | 30 | 33 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1000 | 15 | 27 | 0 | 0 | 0 | 0 | 1 |
| 3 | 1000 | 30 | 28 | 1 | 1 | 0 | 0 | 1 |
| 4 | 1000 | 30 | 29 | 0 | 1 | 0 | 0 | 1 |

### What are our lables?

In [604]:
```
y = df['loan_status'].values
y[0:5]
```

Out[604]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

In [605]:
```
df.loan_status.unique()
```

Out[605]:
```
array(['PAIDOFF', 'COLLECTION'], dtype=object)
```

## Normalize Data

Data Standardization give data zero mean and unit variance
(technically should be done after train test split; the dataset has already been split)

In [606]:
```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[606]:
```
array([[ 0.52,  0.92,  2.33, -0.42, -1.21, -0.38,  1.14, -0.87],
       [ 0.52,  0.92,  0.34,  2.38, -1.21,  2.62, -0.88, -0.87],
       [ 0.52, -0.96, -0.65, -0.42, -1.21, -0.38, -0.88,  1.15],
       [ 0.52,  0.92, -0.49,  2.38,  0.83, -0.38, -0.88,  1.15],
       [ 0.52,  0.92, -0.32, -0.42,  0.83, -0.38, -0.88,  1.15]])
```

# Classification

Use the training set to build an accurate model. Then use the test set to report the accuracy of the model.
Use the following algorithm:

```
K Nearest Neighbor(KNN)
Decision Tree
Support Vector Machine
Logistic Regression
```

**Notice:**

```
  Use either scikit-learn, Scipy or Numpy libraries for developing the classification
algorithms.
```

# K Nearest Neighbor(KNN)

Find the best k to build the model with the best accuracy.

### Splitting data into train and test

In [607]:
```
from sklearn.model_selection import train_test_split
# seed=50; if putting seed than update random_state=seed
# random number generator will decide the splitting of data into train and test in
dices
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
ate=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (276, 8) (276,)
Test set: (70, 8) (70,)
```

### Calculate euclidean distance - NEED HELP

In [608]:
```
# (276+70)*8=2768
import math
math.sqrt(2768)
```

Out[608]:
```
52.61178575186362
```

In [609]:
```
from sklearn.metrics.pairwise import pairwise_distances
dist = pairwise_distances(X_train, X_test, n_jobs=-1)
print ('Euclidean distance: ',dist)
```

```
Euclidean distance:  [[4.06 2.86 2.93 ... 3.5  3.6  3.42]
 [0.66 3.93 2.89 ... 4.53 2.99 2.52]
 [3.51 3.32 3.42 ... 3.94 4.25 2.93]
 ...
 [4.01 2.78 2.93 ... 3.56 4.11 3.67]
 [3.57 3.39 3.52 ... 4.06 4.61 3.22]
 [0.   3.88 2.78 ... 4.43 2.51 2.2 ]]
```

### KNN - K Nearest Neighbor algorithm - feature similarity - can be used when data is labeled, noise free and small dataset

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
*** Classifying cases based on their similarity to other cases
```

In [610]:
```
# Choosing the right value of k is a process called parameter tuning, and is impor
tant for better accuracy
from sklearn.neighbors import KNeighborsClassifier

score=[]
for k in range(1,53):
#k = 8
#Train Model and Predict
    knn = KNeighborsClassifier(n_neighbors=k,weights='uniform')
    knn.fit(X_train,y_train)
    # modelknn = knn.fit(preprocessing.scale(X_train),xtest) if data is not prepro
cessed
    #given a trained model, predict the label of a new set of data
    predKNN = knn.predict(X_test)
    # predknn = modelknn.predict(preprocessing.scale(y_train))

    accuracy=metrics.accuracy_score(y_test, predKNN)
    # appends a passed obj into the existing list
    score.append(accuracy*100)
    print (k,': ',round(accuracy,4)*100,'%')
```

```
1 :  67.14 %
2 :  65.71000000000001 %
3 :  71.43 %
4 :  68.57 %
5 :  75.71 %
6 :  71.43 %
7 :  78.57 %
8 :  75.71 %
9 :  75.71 %
10 :  67.14 %
11 :  70.0 %
12 :  72.86 %
13 :  70.0 %
14 :  70.0 %
15 :  68.57 %
16 :  72.86 %
17 :  72.86 %
18 :  72.86 %
19 :  70.0 %
20 :  68.57 %
21 :  71.43 %
22 :  68.57 %
23 :  70.0 %
24 :  70.0 %
25 :  72.86 %
26 :  71.43 %
27 :  77.14 %
28 :  68.57 %
29 :  78.57 %
30 :  75.71 %
31 :  78.57 %
32 :  74.29 %
33 :  78.57 %
34 :  78.57 %
35 :  78.57 %
36 :  75.71 %
37 :  78.57 %
38 :  78.57 %
39 :  78.57 %
40 :  78.57 %
41 :  77.14 %
42 :  77.14 %
43 :  77.14 %
44 :  78.57 %
45 :  77.14 %
46 :  78.57 %
47 :  78.57 %
48 :  78.57 %
49 :  78.57 %
50 :  78.57 %
51 :  78.57 %
52 :  78.57 %
```

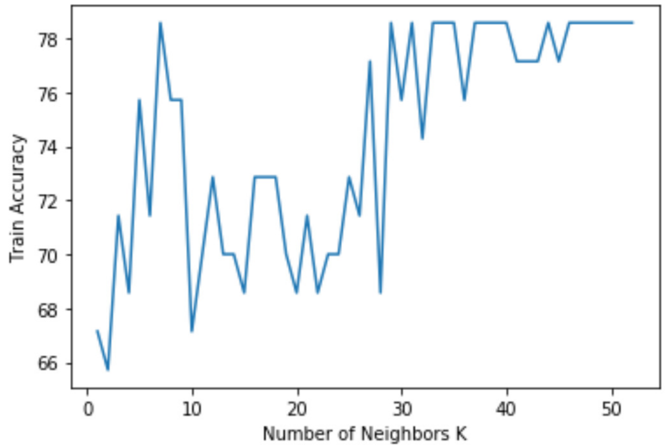
In [611]:
```
print(score.index(max(score))+1,' : ',round(max(score),2),'%')
```

```
7  :  78.57 %
```

The highest accuracy is achieved by K=7 with accuracy score= 78.57%

In [612]:
```
plt.plot(range(1,53),score)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Train Accuracy')
```

Out[612]:
```
Text(0, 0.5, 'Train Accuracy')
```




## Evaluation Metrics

In [613]:
```
# Accuracy classification score is a function that computes subset accuracy. This
function is equal to the jaccard_similarity_score function.
# It calculates how closely the actual labels and predicted labels are matched in
the test set.
from sklearn.metrics import classification_report,jaccard_similarity_score,log_los
s,f1_score, confusion_matrix
print(classification_report(y_test,predKNN))
print('\n')
print('Jaccard Similarity Score: ',round(jaccard_similarity_score(y_test,predKNN)*
100,2),'%')
print('F1 Score: ',f1_score(y_test,predKNN,average=None))
print('Train Accuracy: ',round(metrics.accuracy_score(y_train, knn.predict(X_trai
n))*100,2),'%')
print("Test Accuracy: ", round(metrics.accuracy_score(y_test, predKNN)*100,2),'%')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.00      0.00      0.00        15
     PAIDOFF       0.79      1.00      0.88        55

    accuracy                           0.79        70
   macro avg       0.39      0.50      0.44        70
weighted avg       0.62      0.79      0.69        70



Jaccard Similarity Score:  78.57 %
F1 Score:  [0.   0.88]
Train Accuracy:  74.64 %
Test Accuracy:  78.57 %

P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1437: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labe
ls with no predicted samples.
  'precision', 'predicted', average, warn_for)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1437: UndefinedM
etricWarning: F-score is ill-defined and being set to 0.0 in labels with no pred
icted samples.
  'precision', 'predicted', average, warn_for)
```

In [614]:
```
# creating an output for my predicted data
for i in range(len(predKNN)):
    print("X_test=%s, Predicted=%s" % (X_test[i], predKNN[i]))
```

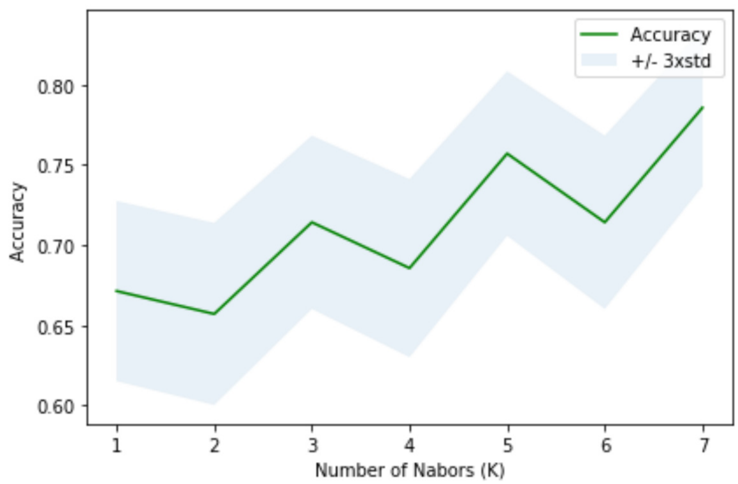
```
X_test=[ 0.52  0.92 -0.16 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[-1.31 -0.96 -0.16 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  0.01 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -1.15 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.51 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96 -0.65 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.84 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92  1.5  -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -1.15 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96 -1.81 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.82 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.16 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[-4.06 -0.96 -1.32  2.38 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.65 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  2.33 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -1.96 -0.82  2.38 -1.21  2.62 -0.88 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -1.48 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -0.96 -0.16 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.49 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92  1.   -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.34  2.38  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.01 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.82 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -1.15  2.38  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[-0.4  -1.96 -0.82  2.38  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.49 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.65 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[-1.31 -0.96  2.5   2.38  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.65 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  1.    2.38 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96 -0.16 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.82  2.38  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[-1.31 -0.96 -0.82 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.67 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[-1.31 -0.96  0.67  2.38  0.83  2.62 -0.88 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  0.51 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.49 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -1.96  0.01 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.16  2.38  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92  2.17 -0.42  0.83  2.62 -0.88 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.82 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.84 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  1.34 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.49 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -0.96 -0.32 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.49 -0.42  0.83  2.62 -0.88 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -0.96 -0.32 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.16 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  0.67 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -1.96 -0.32 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.34  2.38 -1.21  2.62 -0.88 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -1.96 -0.65 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[-1.31 -0.96 -0.82 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92  0.34 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[-4.06 -1.96 -1.48 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[-1.31 -0.96  3.33 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92  1.34 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92  1.17  2.38  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96 -0.16 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.16 -0.42 -1.21 -0.38  1.14 -0.87], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  3.16 -0.42  0.83 -0.38 -0.88 -0.87], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -0.32 -0.42 -1.21 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52 -0.96  0.51 -0.42  0.83 -0.38 -0.88  1.15], Predicted=PAIDOFF
X_test=[ 0.52  0.92 -1.65 -0.42  0.83 -0.38  1.14 -0.87], Predicted=PAIDOFF
```

### Other method - using Kolmogorov-Smirnov Statistics

In [615]:
```
# calculate the accuracy of KNN for different Ks
# Kolmogorov-Smirnov (KS) Statistics used for validating predictive models
# Compares the two cumulative distributions and returns the maximum difference bet
ween them
Ks = 8
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];

for n in range(1,Ks):
#Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)

    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)
    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

print ('Mean Accuracy: ',mean_acc*100,'%')
print ('Standard Accuracy: ',std_acc*100,'%')
```

```
Mean Accuracy:  [67.14 65.71 71.43 68.57 75.71 71.43 78.57] %
Standard Accuracy:  [5.61 5.67 5.4  5.55 5.13 5.4  4.9 ] %
```

In [616]:
```
# Plot model accuracy for Different number of Neighbors
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=
0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
# tight_layout()- provides routines to adjust subplot params so that subplots are
nicely fit in the figure
plt.show()
```




In [617]:
```
# argmax()+1 - Returns the indices of the maximum values along an axis
print( "The best accuracy was with", round(mean_acc.max()*100,2), "with k=", mean_
acc.argmax()+1)
```

```
The best accuracy was with 78.57 with k= 7
```

### Evaluation Metrics

In [618]:
```
# Accuracy classification score is a function that computes subset accuracy. This
function is equal to the jaccard_similarity_score function.
# It calculates how closely the actual labels and predicted labels are matched in
the test set.
from sklearn.metrics import classification_report, jaccard_similarity_score
print (classification_report(y_test, yhat))
print('\n')
#print ('Confusion Matrix: ',ConfustionMx(X_train, yhat))
print('Jaccard Similarity Score: ',round(jaccard_similarity_score(y_test, yhat)*10
0,2),'%')
print('Train set Accuracy: ',round(metrics.accuracy_score(y_train, neigh.predict(X
_train))*100,2),'%')
print('Test set Accuracy: ',round(metrics.accuracy_score(y_test, yhat)*100,2),'%')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.50      0.40      0.44        15
     PAIDOFF       0.84      0.89      0.87        55

    accuracy                           0.79        70
   macro avg       0.67      0.65      0.66        70
weighted avg       0.77      0.79      0.78        70



Jaccard Similarity Score:  78.57 %
Train set Accuracy:  80.8 %
Test set Accuracy:  78.57 %

P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

## Decision Tree

In [619]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree=DecisionTreeClassifier()
```

In [620]:
```
parameter_grid = {'max_depth': [1, 2, 3, 4, 5,6,5,9,15,20],
                   'max_features': [1, 2, 3, 4,5,6,7,8],
                  'random_state':[0,15,20,35,50,80,100,150,180,200],
                  'criterion':['gini','entropy'],
                  }

grid_search = GridSearchCV(dtree, param_grid = parameter_grid,
                           cv =10)

grid_search.fit(X_train, y_train)

print ("Best Score: {}".format(grid_search.best_score_))
print ("Best Parameters: {}".format(grid_search.best_params_))
```

```
Best Score: 0.7644927536231884
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 1, 'rando
m_state': 180}

P:\Anaconda\lib\site-packages\sklearn\model_selection\_search.py:814: Deprecatio
nWarning: The default of the `iid` parameter will change from True to False in v
ersion 0.22 and will be removed in 0.24. This will change numeric results when t
est-set sizes are unequal.
  DeprecationWarning)
```

In [621]:
```
dtree = DecisionTreeClassifier(criterion="entropy", max_depth = 5, max_features=2,
random_state=0)
dtree
```

Out[621]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
                       max_features=2, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [622]:
```
dtree.fit(X_train,y_train)
pred_Dtree=dtree.predict(X_test)
```

In [623]:
```
from sklearn import metrics
print(classification_report(y_test,pred_Dtree))
print('\n')
print('Jaccard Similarity Score: ',round(jaccard_similarity_score(y_test,pred_Dtre
e)*100,2),'%')
print('F1 Score: ',f1_score(y_test,pred_Dtree,average=None))
print('Train Accuracy: ',metrics.accuracy_score(y_train, dtree.predict(X_train))*1
00,'%')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.31      0.27      0.29        15
     PAIDOFF       0.81      0.84      0.82        55

    accuracy                           0.71        70
   macro avg       0.56      0.55      0.55        70
weighted avg       0.70      0.71      0.71        70



Jaccard Similarity Score:  71.43 %
F1 Score:  [0.29 0.82]
Train Accuracy:  77.89855072463769 %
```

```
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

# Support Vector Machine

In [624]:
```
df.loan_status.unique()
```

Out[624]:
```
array(['PAIDOFF', 'COLLECTION'], dtype=object)
```

#### With kernel = rbf - Radial Basis Function

In [625]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn import svm
```

In [626]:
```
svm = svm.SVC(kernel='rbf')
svm.fit(X_train, y_train)
pred_svm = svm.predict(X_test)
```

```
P:\Anaconda\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: The defaul
t value of gamma will change from 'auto' to 'scale' in version 0.22 to account b
etter for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid
this warning.
  "avoid this warning.", FutureWarning)
```

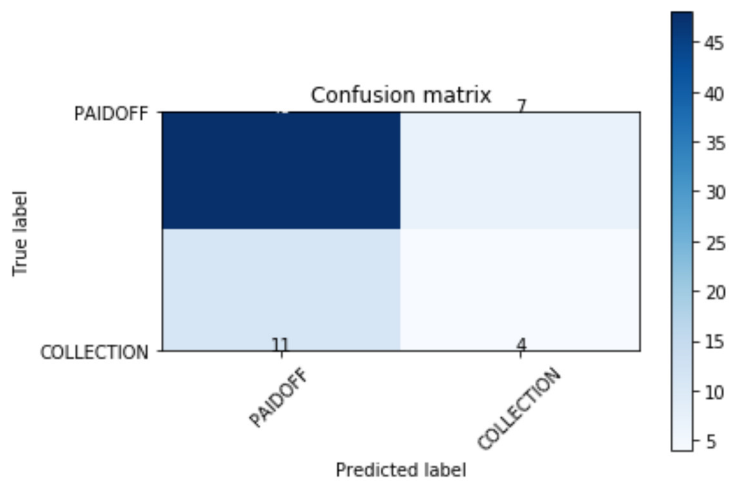
# Evaluation Metrics

In [627]:
```
from sklearn.metrics import classification_report, confusion_matrix, f1_score, jac
card_similarity_score, accuracy_score
import itertools
```

In [628]:
```
def plot_confusion_matrix(cm, classes,normalize=False,title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [629]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_svm, labels=['PAIDOFF','COLLECTION'])
np.set_printoptions(precision=2)

print (classification_report(y_test, pred_svm))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['PAIDOFF','COLLECTION'],normalize= Fals
e,title='Confusion matrix')
print ('\n')
print ('Jaccard Similarity Score:',round(jaccard_similarity_score(y_test, pred_sv
m)*100,2),'%')
# average='weighted'
print ('F1 Score: ',f1_score(y_test, pred_svm, average=None))
print('Train Accuracy : ',metrics.accuracy_score(y_train, svm.predict(X_train))*10
0,'%')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.36      0.27      0.31        15
     PAIDOFF       0.81      0.87      0.84        55

    accuracy                           0.74        70
   macro avg       0.59      0.57      0.57        70
weighted avg       0.72      0.74      0.73        70

Confusion matrix, without normalization
[[48  7]
 [11  4]]


Jaccard Similarity Score: 74.29 %
F1 Score:  [0.31 0.84]
Train Accuracy :  78.26086956521739 %

P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```




**With kernel = linear**

In [630]:
```
from sklearn.svm import SVC
from sklearn import svm
svm2 = svm.SVC(kernel='linear')
svm2.fit(X_train, y_train)
```

Out[630]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [631]:
```
pred_svm2 = svm2.predict(X_test)
```

In [632]:
```
# calculate f1 score and jaccard score
print('Jaccard Similarity Score: ', round(jaccard_similarity_score(y_test, pred_sv
m2)*100,2),'%')
# average='weighted'
print('F1 Score: ', f1_score(y_test, pred_svm2, average=None))
```

```
Jaccard Similarity Score:  78.57 %
F1 Score:  [0.   0.88]

P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1437: UndefinedM
etricWarning: F-score is ill-defined and being set to 0.0 in labels with no pred
icted samples.
  'precision', 'predicted', average, warn_for)
```

# Logistic Regression

In [633]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
# other solvers: 'sag', 'saga', 'lbfgs', 'newton-cg'
```

Out[633]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [634]:
```
pred_LG = LR.predict(X_test)
# predict_proba to calculate log loss
pred_LG_prob = LR.predict_proba(X_test)
```

## Evaluation Metrics

In [635]:
```
from sklearn.metrics import jaccard_similarity_score, classification_report, confu
sion_matrix, log_loss
import itertools
print (classification_report(y_test, pred_LG))
print ('\n')
print ('Jaccard Similarity Score: ',round(jaccard_similarity_score(y_test, pred_L
G)*100,2),'%')
print('F1 Score: ',f1_score(y_test,pred_LG,average=None))
print('Train Accuracy: ',metrics.accuracy_score(y_train, LR.predict(X_train))*10
0,'%')
print ('Log Loss: ',log_loss(y_test, pred_LG_prob))
```

```
              precision    recall  f1-score   support

  COLLECTION       0.18      0.13      0.15        15
     PAIDOFF       0.78      0.84      0.81        55

    accuracy                           0.69        70
   macro avg       0.48      0.48      0.48        70
weighted avg       0.65      0.69      0.67        70



Jaccard Similarity Score:  68.57 %
F1 Score:  [0.15 0.81]
Train Accuracy:  75.72463768115942 %
Log Loss:  0.5772287609479654

P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

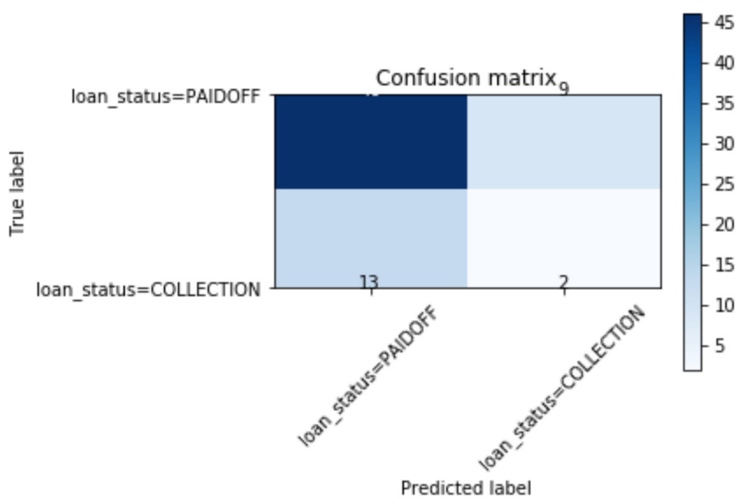
In [636]:

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [637]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, pred_LG, labels=['PAIDOFF','COLLECTION'])
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['loan_status=PAIDOFF','loan_status=COLL
ECTION'],normalize= False,
title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[46  9]
 [13  2]]
```



# Choosing the best classifier (algorithm) on the loan dataset.

## Report

Report the accuracy of the built model using different evaluation metrics:

| Algorithm | Jaccard | F1-score | LogLoss |
|---:|---:|---:|---:|
| KNN | ? | ? | NA |
| Decision Tree | ? | ? | NA |
| SVM | ? | ? | NA |
| LogisticRegression | ? | ? | ? |

In [685]:
```
Algoritm=['KNN','Decision Tree','SVM','Logistic Regression']

# Jaccard scores
j_knn=round(jaccard_similarity_score(y_test,predKNN)*100,2)
j_dtree=round(jaccard_similarity_score(y_test,pred_Dtree)*100,2)
j_svm=round(jaccard_similarity_score(y_test,pred_svm)*100,2)
j_lgm=round(jaccard_similarity_score(y_test,pred_LG)*100,2)
Jaccard=[j_knn,j_dtree,j_svm,j_lgm]

# F1 scores
f1_knn=f1_score(y_test,predKNN,average=None)
f1_dtree=f1_score(y_test,pred_Dtree,average=None)
f1_svm=f1_score(y_test,pred_svm,average=None)
f1_lgm=f1_score(y_test,pred_LG,average=None)
F1_score=[f1_knn,f1_dtree,f1_svm,f1_lgm]

# Log Loss scores
l_LG=log_loss(y_test, pred_LG_prob)
```

```
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1437: UndefinedM
etricWarning: F-score is ill-defined and being set to 0.0 in labels with no pred
icted samples.
  'precision', 'predicted', average, warn_for)
```

In [686]:
```
table = pd.DataFrame({
    "Algorithm": Algoritm,
    "Jaccard": Jaccard,
    "F1-Score": F1_score,
    "LogLoss":[np.NAN,np.NAN,np.NAN,l_LG]})
```

const ( ResetAll = "\033[0m"

```
Bold       = "\033[1m"
Dim        = "\033[2m"
Underlined = "\033[4m"
Blink      = "\033[5m"
Reverse    = "\033[7m"
Hidden     = "\033[8m"

ResetBold       = "\033[21m"
ResetDim        = "\033[22m"
ResetUnderlined = "\033[24m"
ResetBlink      = "\033[25m"
ResetReverse    = "\033[27m"
ResetHidden     = "\033[28m"

Default      = "\033[39m"
Black        = "\033[30m"
Red          = "\033[31m"
Green        = "\033[32m"
Yellow       = "\033[33m"
Blue         = "\033[34m"
Magenta      = "\033[35m"
Cyan         = "\033[36m"
LightGray    = "\033[37m"
DarkGray     = "\033[90m"
LightRed     = "\033[91m"
LightGreen   = "\033[92m"
LightYellow  = "\033[93m"
LightBlue    = "\033[94m"
LightMagenta = "\033[95m"
LightCyan    = "\033[96m"
White        = "\033[97m"

BackgroundDefault      = "\033[49m"
BackgroundBlack        = "\033[40m"
BackgroundRed          = "\033[41m"
BackgroundGreen        = "\033[42m"
BackgroundYellow       = "\033[43m"
BackgroundBlue         = "\033[44m"
BackgroundMagenta      = "\033[45m"
BackgroundCyan         = "\033[46m"
BackgroundLightGray    = "\033[47m"
BackgroundDarkGray     = "\033[100m"
BackgroundLightRed     = "\033[101m"
BackgroundLightGreen   = "\033[102m"
BackgroundLightYellow  = "\033[103m"
BackgroundLightBlue    = "\033[104m"
BackgroundLightMagenta = "\033[105m"
BackgroundLightCyan    = "\033[106m"
BackgroundWhite        = "\033[107m"
```

)

In [687]:
```
class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'
```

In [688]:
```
print (color.BOLD + color.BLUE + 'Loan_train data results' + color.END)
table
```

**Loan_train data results**

Out[688]:

| | Algorithm | Jaccard | F1-Score | LogLoss |
|---|---|---|---|---|
| 0 | KNN | 78.57 | [0.0, 0.88] | NaN |
| 1 | Decision Tree | 71.43 | [0.28571428571428575, 0.8214285714285714] | NaN |
| 2 | SVM | 74.29 | [0.30769230769230765, 0.8421052631578948] | NaN |
| 3 | Logistic Regression | 68.57 | [0.15384615384615383, 0.8070175438596492] | 0.577229 |

**After using KNN, Decision Tree, SVM and Logistic Regression models, we see that the KNN model results in the highest Jaccard Similarity Score and F1 Score with k=7 folds.**

# Model Evaluation using Test set

# Using Loan_train.csv for train model, and Loan_test.csv as new data for testing model

In [642]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

### Load Test set for evaluation

In [643]:
```
test_df = pd.read_csv('U:/Documents/Gunn Notes/Data Analyst Training/Machine_Learning_with_Python/Loan_test.csv')
test_df.head()
```

Out[643]:

| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | PAIDOFF | 1000 | 30 | 9/8/2016 | 10/7/2016 | 50 | Bechalor | female |
| 1 | 5 | 5 | PAIDOFF | 300 | 7 | 9/9/2016 | 9/15/2016 | 35 | Master or Above | male |
| 2 | 21 | 21 | PAIDOFF | 1000 | 30 | 9/10/2016 | 10/9/2016 | 43 | High School or Below | female |
| 3 | 24 | 24 | PAIDOFF | 1000 | 30 | 9/10/2016 | 10/9/2016 | 26 | college | male |
| 4 | 35 | 35 | PAIDOFF | 800 | 15 | 9/11/2016 | 9/25/2016 | 29 | Bechalor | male |

## Convert to date time object

In [644]:
```
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'])
test_df.head()
```

Out[644]:

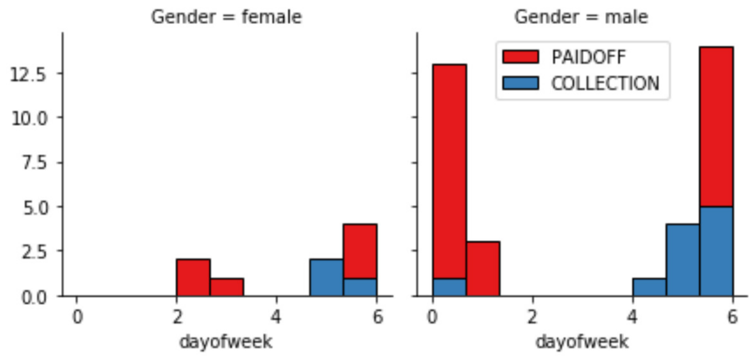
| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 50 | Bechalor | female |
| 1 | 5 | 5 | PAIDOFF | 300 | 7 | 2016-09-09 | 2016-09-15 | 35 | Master or Above | male |
| 2 | 21 | 21 | PAIDOFF | 1000 | 30 | 2016-09-10 | 2016-10-09 | 43 | High School or Below | female |
| 3 | 24 | 24 | PAIDOFF | 1000 | 30 | 2016-09-10 | 2016-10-09 | 26 | college | male |
| 4 | 35 | 35 | PAIDOFF | 800 | 15 | 2016-09-11 | 2016-09-25 | 29 | Bechalor | male |

In [645]:
```
# Lets look at the day of the week people get the loan
test_df['dayofweek'] = test_df['effective_date'].dt.dayofweek
test_df.head()
```

Out[645]:

| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender | da |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 50 | Bechalor | female | |
| 1 | 5 | 5 | PAIDOFF | 300 | 7 | 2016-09-09 | 2016-09-15 | 35 | Master or Above | male | |
| 2 | 21 | 21 | PAIDOFF | 1000 | 30 | 2016-09-10 | 2016-10-09 | 43 | High School or Below | female | |
| 3 | 24 | 24 | PAIDOFF | 1000 | 30 | 2016-09-10 | 2016-10-09 | 26 | college | male | |
| 4 | 35 | 35 | PAIDOFF | 800 | 15 | 2016-09-11 | 2016-09-25 | 29 | Bechalor | male | |

In [646]:
```
# Lets look at the day of the week people get the loan
bins = np.linspace(test_df.dayofweek.min(), test_df.dayofweek.max(), 10)
g = sns.FacetGrid(test_df, col="Gender", hue="loan_status", palette="Set1", col_wr
ap=2)
g.map(plt.hist, 'dayofweek', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```

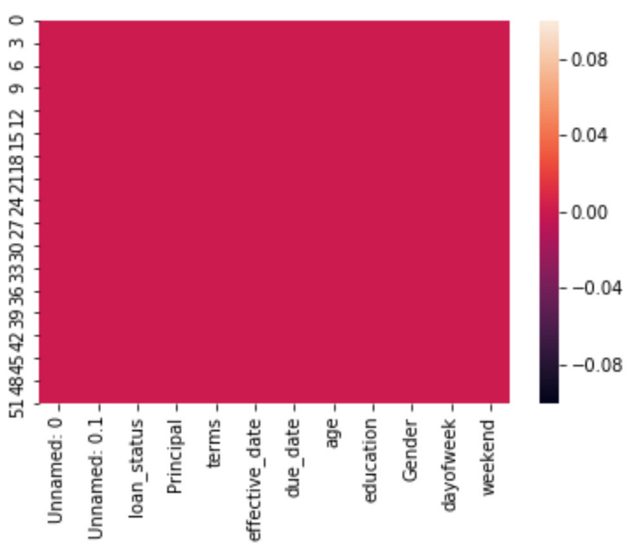



In [647]:
```
# use Feature binarization to set a threshold values less then day 4
test_df['weekend']=test_df['dayofweek'].apply(lambda x: 1 if (x>3) else 0)
test_df.head()
```

Out[647]:

| | Unnamed: 0 | Unnamed: 0.1 | loan_status | Principal | terms | effective_date | due_date | age | education | Gender | da |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 50 | Bechalor | female | |
| 1 | 5 | 5 | PAIDOFF | 300 | 7 | 2016-09-09 | 2016-09-15 | 35 | Master or Above | male | |
| 2 | 21 | 21 | PAIDOFF | 1000 | 30 | 2016-09-10 | 2016-10-09 | 43 | High School or Below | female | |
| 3 | 24 | 24 | PAIDOFF | 1000 | 30 | 2016-09-10 | 2016-10-09 | 26 | college | male | |
| 4 | 35 | 35 | PAIDOFF | 800 | 15 | 2016-09-11 | 2016-09-25 | 29 | Bechalor | male | |

In [648]:
```
sns.heatmap(test_df.isnull());
```



In [649]:
```
test_df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[649]:
```
Gender  loan_status
female  PAIDOFF        0.727273
        COLLECTION     0.272727
male    PAIDOFF        0.744186
        COLLECTION     0.255814
Name: loan_status, dtype: float64
```

In [650]:
```
test_df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
```

In [651]:
```
test_df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[651]:
```
education             loan_status
Bechalor              PAIDOFF        1.000000
High School or Below  PAIDOFF        0.523810
                      COLLECTION     0.476190
Master or Above       PAIDOFF        1.000000
college               PAIDOFF        0.826087
                      COLLECTION     0.173913
Name: loan_status, dtype: float64
```

In [652]:
```
test_df[['Principal','terms','age','Gender','weekend', 'education']].head()
```

Out[652]:

| | Principal | terms | age | Gender | weekend | education |
|---|---|---|---|---|---|---|
| 0 | 1000 | 30 | 50 | 1 | 0 | Bechalor |
| 1 | 300 | 7 | 35 | 0 | 1 | Master or Above |
| 2 | 1000 | 30 | 43 | 1 | 1 | High School or Below |
| 3 | 1000 | 30 | 26 | 0 | 1 | college |
| 4 | 800 | 15 | 29 | 0 | 1 | Bechalor |

Move education from rows to columns

In [653]:
```
test_Feature = test_df[['Principal','terms','age','Gender','weekend']]
test_Feature = pd.concat([test_Feature,pd.get_dummies(test_df['education'])], axi
s=1)
test_Feature.drop(['Master or Above'], axis = 1,inplace=True)
test_Feature.head()
```

Out[653]:

| | Principal | terms | age | Gender | weekend | Bechalor | High School or Below | college |
|---|---|---|---|---|---|---|---|---|
| 0 | 1000 | 30 | 50 | 1 | 0 | 1 | 0 | 0 |
| 1 | 300 | 7 | 35 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1000 | 30 | 43 | 1 | 1 | 0 | 1 | 0 |
| 3 | 1000 | 30 | 26 | 0 | 1 | 0 | 0 | 1 |
| 4 | 800 | 15 | 29 | 0 | 1 | 1 | 0 | 0 |

The same can be broken into 2 step process

In [654]:
```
# test_dumm=pd.get_dummies(test_df['education'])
# test_dumm=test_dumm.drop('Master or Above',axis=1,inplace=True)
# test_dumm=test_dumm[['Bechalor','High School or Below','college']]
```

In [655]:
```
# test_Feature = test_df[['Principal','terms','age','Gender','weekend']]
# test_Feature = pd.concat([test_feature,test_dumm], axis=1)
# test_Feature.drop(['Master or Above'], axis = 1,inplace=True)
# test_Feature.head()
```

In [656]:
```
test_data= test_Feature
test_data= preprocessing.StandardScaler().fit(test_data).transform(test_data)
```

In [657]:
```
target=test_df['loan_status']
```

## KNN

In [658]:
```
knn=KNeighborsClassifier()
knn.fit(X,y)
# test_data is my test file without target
predknn_test=knn.predict(test_data)
accuracy=metrics.accuracy_score(predknn_test,target)
print("accuracy : ",round(accuracy,3)*100,'%')
```

```
accuracy :  74.1 %
```

#### BEST K FOR TEST DATA

In [659]:
```
score=[]
for k in range(1,100):
    knn=KNeighborsClassifier(n_neighbors=k,weights='uniform')
    knn.fit(X,y)
    predknn=knn.predict(test_data)
    accuracy=metrics.accuracy_score(predknn,target)
    score.append(accuracy*100)
    print (k,': ',accuracy)
```

```
1 :  0.7037037037037037
2 :  0.5740740740740741
3 :  0.6481481481481481
4 :  0.6296296296296297
5 :  0.7407407407407407
6 :  0.6851851851851852
7 :  0.7222222222222222
8 :  0.7037037037037037
9 :  0.7037037037037037
10 :  0.6851851851851852
11 :  0.6851851851851852
12 :  0.6666666666666666
13 :  0.7037037037037037
14 :  0.7037037037037037
15 :  0.7222222222222222
16 :  0.7037037037037037
17 :  0.7222222222222222
18 :  0.7037037037037037
19 :  0.7222222222222222
20 :  0.7407407407407407
21 :  0.7592592592592593
22 :  0.7592592592592593
23 :  0.7592592592592593
24 :  0.7222222222222222
25 :  0.7407407407407407
26 :  0.7777777777777778
27 :  0.7592592592592593
28 :  0.7777777777777778
29 :  0.7592592592592593
30 :  0.7777777777777778
31 :  0.7407407407407407
32 :  0.7962962962962963
33 :  0.7777777777777778
34 :  0.7962962962962963
35 :  0.7962962962962963
36 :  0.7777777777777778
37 :  0.7962962962962963
38 :  0.7962962962962963
39 :  0.7962962962962963
40 :  0.7962962962962963
41 :  0.7962962962962963
42 :  0.7962962962962963
43 :  0.7777777777777778
44 :  0.7962962962962963
45 :  0.7962962962962963
46 :  0.7962962962962963
47 :  0.7777777777777778
48 :  0.7777777777777778
49 :  0.7592592592592593
50 :  0.7777777777777778
51 :  0.7777777777777778
52 :  0.7777777777777778
53 :  0.7407407407407407
54 :  0.7407407407407407
55 :  0.7407407407407407
56 :  0.7407407407407407
57 :  0.7407407407407407
58 :  0.7407407407407407
59 :  0.7407407407407407
60 :  0.7407407407407407
61 :  0.7407407407407407
62 :  0.7407407407407407
63 :  0.7407407407407407
64 :  0.7407407407407407
```

In [660]:
```
print(score.index(max(score))+1,' : ',round(max(score),2),'%')
```

```
32  :  79.63 %
```

In [661]:
```
knn=KNeighborsClassifier(n_neighbors=32)
knn.fit(X,y)
predknn_test=knn.predict(test_data)
accuracy=metrics.accuracy_score(predknn_test,target)
print("accuracy : ",round(accuracy,4)*100,'%')
```

```
accuracy :  79.63 %
```

In [689]:
```
print(classification_report(target,predknn_test))
print('\n')
print('Jaccard Similarity Score: ',round(jaccard_similarity_score(target,predknn_t
est)*100,2),'%')
print('F1 Score: ',f1_score(target,predknn_test,average=None))
print('Train Accuracy: ',metrics.accuracy_score(y,knn.predict(X))*100,'%')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.71      0.36      0.48        14
     PAIDOFF       0.81      0.95      0.87        40

    accuracy                           0.80        54
   macro avg       0.76      0.65      0.67        54
weighted avg       0.78      0.80      0.77        54



Jaccard Similarity Score:  79.63 %
F1 Score:  [0.48 0.87]
Train Accuracy:  74.85549132947978 %

P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

In [663]:
```
# creating an output for my predicted data
for i in range(len(predknn_test)):
    print("test_data=%s, Predicted=%s" % (test_data[i], predknn_test[i]))
```

```
test_data=[ 0.49  0.93  3.06  1.98 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.88  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.98 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.48 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-1.24 -0.79  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -1.32 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.81  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.87 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.32 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.14 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.88 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -1.7   0.03  1.98  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49 -1.7  -0.48 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -1.7  -0.81 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.87 -0.51 -1.3  -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67  0.93 -0.65 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  2.39 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.65 -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.49 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04  1.98 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31  1.98 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49 -0.79 -0.14  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.54 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.98  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -1.99 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49  0.93 -1.32  1.98  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[-0.67 -0.79 -0.81 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[-0.67 -0.79 -0.48 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[-0.67 -0.79  0.7  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
```

## Decision tree

In [664]:
```
parameter_grid = {'max_depth': [1, 2, 3, 4, 5,6,5,9,15,20],
                  'max_features': [1,2,3,4,5,6,7,8],
                 'random_state':[0,15,20,35,50,80,100,150,180,200],
                 'criterion':['gini','entropy'],
                 }

grid_search = GridSearchCV(dtree, param_grid = parameter_grid,
                          cv =10)

grid_search.fit(X, y)

print ("Best Score: {}".format(grid_search.best_score_))
print ("Best params: {}".format(grid_search.best_params_))
```

```
Best Score: 0.7687861271676301
Best params: {'criterion': 'entropy', 'max_depth': 6, 'max_features': 4, 'random
_state': 20}

P:\Anaconda\lib\site-packages\sklearn\model_selection\_search.py:814: Deprecatio
nWarning: The default of the `iid` parameter will change from True to False in v
ersion 0.22 and will be removed in 0.24. This will change numeric results when t
est-set sizes are unequal.
  DeprecationWarning)
```

In [665]:
```
dtree=DecisionTreeClassifier(max_depth=6,criterion='entropy',max_features=4,random
_state=20).fit(X,y)
pred_dtree_test=dtree.predict(test_data)
```

In [666]:
```
print(classification_report(target,pred_dtree_test))
print('\n')
print('Jaccard Similarity Score: ',round(jaccard_similarity_score(target,pred_dtre
e_test)*100,2),'%')
print('F1 Score: ',f1_score(target,pred_dtree_test,average=None))
print('Train Accuracy: ',metrics.accuracy_score(y, dtree.predict(X))*100,'%')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.44      0.29      0.35        14
     PAIDOFF       0.78      0.88      0.82        40

    accuracy                           0.72        54
   macro avg       0.61      0.58      0.59        54
weighted avg       0.69      0.72      0.70        54



Jaccard Similarity Score:  72.22 %
F1 Score:  [0.35 0.82]
Train Accuracy:  79.47976878612717 %

P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

In [667]:

```
# creating an output for my predicted data
for i in range(len(pred_dtree_test)):
    print("test_data=%s, Predicted=%s" % (test_data[i], pred_dtree_test[i]))
```

```
test_data=[ 0.49  0.93  3.06  1.98 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.88  1.98  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49  0.93 -0.98 -0.51  0.77 -0.42 -0.8   1.16], Predicted=COLLECTIO
N
test_data=[-0.67 -0.79 -0.48 -0.51  0.77  2.4  -0.8  -0.86], Predicted=COLLECTIO
N
test_data=[-1.24 -0.79  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -1.32 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.81  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.87 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.32 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.14 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.88 -0.51  0.77  2.4  -0.8  -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49 -1.7   0.03  1.98  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -1.7  -0.48 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -1.7  -0.81 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.87 -0.51 -1.3  -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67  0.93 -0.65 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  2.39 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.65 -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.49 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04  1.98 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31  1.98 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49 -0.79 -0.14  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.54 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.98  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -1.99 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -1.32  1.98  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[-0.67 -0.79 -0.81 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[-0.67 -0.79 -0.48 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[-0.67 -0.79  0.7  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
```

## Support Vector Machine (SVM)

In [668]:
```
svm=SVC().fit(X,y)
pred_svm_test=svm.predict(test_data)
```

In [669]:
```
print(classification_report(target,pred_svm_test))
print('\n')
print('Jaccard Similarity Score: ',round(jaccard_similarity_score(target,pred_svm_
test)*100,2),'%')
print('F1 Score: ',f1_score(target,pred_svm_test,average=None))
print('Train Accuracy: ',metrics.accuracy_score(y, svm.predict(X))*100,'%')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.00      0.00      0.00        14
     PAIDOFF       0.74      0.97      0.84        40

    accuracy                           0.72        54
   macro avg       0.37      0.49      0.42        54
weighted avg       0.55      0.72      0.62        54



Jaccard Similarity Score:  72.22 %
F1 Score:  [0.   0.84]
Train Accuracy:  76.01156069364163 %
```

```
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

In [670]:
```
# creating an output for my predicted data
for i in range(len(pred_svm_test)):
    print("test_data=%s, Predicted=%s" % (test_data[i], pred_svm_test[i]))
```

```
test_data=[ 0.49  0.93  3.06  1.98 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.88  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.98 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.48 -0.51  0.77  2.4  -0.8  -0.86], Predicted=COLLECTIO
N
test_data=[-1.24 -0.79  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -1.32 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.81  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.87 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.32 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.14 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.88 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -1.7   0.03  1.98  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -1.7  -0.48 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -1.7  -0.81 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.87 -0.51 -1.3  -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67  0.93 -0.65 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  2.39 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.65 -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.49 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04  1.98 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31  1.98 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.14  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.54 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.98  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -1.99 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -1.32  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.81 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.48 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79  0.7  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
```

# Logistic Regression

In [671]:
```
lgm=LogisticRegression().fit(X,y)
```

```
P:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarnin
g: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silenc
e this warning.
  FutureWarning)
```

In [672]:
```
pred_lgm_test=lgm.predict(test_data)
```

In [673]:
```
pred_LG_test = LR.predict(test_data)
# predict_proba to calculate log loss
pred_LG_prob_test = LR.predict_proba(test_data)
```

In [674]:
```
print(classification_report(target,pred_lgm_test))
print('\n')
print('Jaccard Similarity Score: ',round(jaccard_similarity_score(target,pred_lgm_
test)*100,2),'%')
print('F1 Score: ',f1_score(target,pred_lgm_test,average=None))
print('Train Accuracy: ',metrics.accuracy_score(y, lgm.predict(X))*100,'%')
print ('Log Loss: ',log_loss(target, pred_LG_prob_test))
```

```
              precision    recall  f1-score   support

  COLLECTION       1.00      0.07      0.13        14
     PAIDOFF       0.75      1.00      0.86        40

    accuracy                           0.76        54
   macro avg       0.88      0.54      0.50        54
weighted avg       0.82      0.76      0.67        54



Jaccard Similarity Score:  75.93 %
F1 Score:  [0.13 0.86]
Train Accuracy:  75.43352601156069 %
Log Loss:  0.5672153379912981
```

```
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

In [675]:
```
# creating an output for my predicted data
for i in range(len(pred_lgm_test)):
    print("test_data=%s, Predicted=%s" % (test_data[i], pred_lgm_test[i]))
```

```
test_data=[ 0.49  0.93  3.06  1.98 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.88  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.98 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.48 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-1.24 -0.79  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -1.32 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.81  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.87 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.32 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.53 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.14 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.88 -0.51  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -1.7   0.03  1.98  0.77  2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -1.7  -0.48 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -1.7  -0.81 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-3.56 -1.7   0.87 -0.51 -1.3  -0.42 -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.98 -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[-0.67  0.93 -0.65 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  2.39 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3   2.4  -0.8  -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49 -0.79  0.7  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.48 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.65 -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -1.49 -0.51 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  1.04  1.98 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31  1.98 -1.3  -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49 -0.79 -0.14  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79  1.54 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -0.31 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.98  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -1.99 -0.51  0.77 -0.42  1.25 -0.86], Predicted=COLLECTIO
N
test_data=[ 0.49 -0.79 -0.98 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93 -1.32  1.98  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.81 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.03 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79 -0.48 -0.51  0.77 -0.42 -0.8   1.16], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.87 -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[-0.67 -0.79  0.7  -0.51  0.77 -0.42  1.25 -0.86], Predicted=PAIDOFF
test_data=[ 0.49  0.93  0.2  -0.51 -1.3  -0.42  1.25 -0.86], Predicted=PAIDOFF
```

# Choosing the best classifier (algorithm) on the loan test dataset.

## Report

Report the accuracy of the built model using different evaluation metrics:

| Algorithm | Jaccard | F1-score | LogLoss |
|---|---|---|---|
| KNN | ? | ? | NA |
| Decision Tree | ? | ? | NA |
| SVM | ? | ? | NA |
| LogisticRegression | ? | ? | ? |

In [681]:

```
Algoritm=['KNN','Decision Tree','SVM','Logistic Regression']

# Jaccard scores
j_knn_test=round(jaccard_similarity_score(target,predknn_test)*100,2)
j_dtree_test=round(jaccard_similarity_score(target,pred_dtree_test)*100,2)
j_svm_test=round(jaccard_similarity_score(target,pred_svm_test)*100,2)
j_lgm_test=round(jaccard_similarity_score(target,pred_lgm_test)*100,2)
Jaccard=[j_knn_test,j_dtree_test,j_svm_test,j_lgm_test]

# F1 scores
f1_knn_test=f1_score(target,predknn_test,average=None)
f1_dtree_test=f1_score(target,pred_dtree_test,average=None)
f1_svm_test=f1_score(target,pred_svm_test,average=None)
f1_lgm_test=f1_score(target,pred_lgm_test,average=None)
F1_score=[f1_knn_test,f1_dtree_test,f1_svm_test,f1_lgm_test]

# Log Loss scores
l_LG=log_loss(target, pred_LG_prob_test)
```

```
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
P:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:635: Deprecation
Warning: jaccard_similarity_score has been deprecated and replaced with jaccard_
score. It will be removed in version 0.23. This implementation has surprising be
havior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

In [682]:
```
table = pd.DataFrame({
    "Algorithm": Algoritm,
    "Jaccard": Jaccard,
    "F1-Score": F1_score,
    "LogLoss":[np.NAN,np.NAN,np.NAN,l_LG]})
```

In [683]:
```
class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'
```

In [684]:
```
print (color.BOLD + color.BLUE + 'Loan_test data results' + color.END)
table
```

**Loan_test data results**

Out[684]:

| | Algorithm | Jaccard | F1-Score | LogLoss |
|---|---|---|---|---|
| 0 | KNN | 79.63 | [0.4761904761904762, 0.8735632183908046] | NaN |
| 1 | Decision Tree | 72.22 | [0.34782608695652173, 0.823529411764706] | NaN |
| 2 | SVM | 72.22 | [0.0, 0.8387096774193549] | NaN |
| 3 | Logistic Regression | 75.93 | [0.13333333333333333, 0.8602150537634409] | 0.567215 |

**After using KNN, Decision Tree, SVM and Logistic Regression models, we see that the KNN model results in the highest Jaccard Similarity Score and F1 Score with k=32 folds.**

In [ ]: