Loan Data Classification

Outline:

```
1. Data Visulization
2. Feature selection\extraction
3. Data Cleaning
4. Model Building
5. Model Evaluation
```

Project

We load a dataset using Pandas library, and apply the following algorithms, and find the best one for this specific dataset by accuracy evaluation methods.

In [1]:
```
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
```

About dataset

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:

Field	Description
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 schedule
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

Load data from CSV file

In [4]:
```
df = pd.read_csv('loan_train.csv')
df.head()
```

Out[4]:

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

In [5]:
```
df.shape
```

Out[5]: (346, 10)

Convert to date time object

In [6]:
```
df['due_date']=pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

Out[6]:

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

1. Data Visulaization

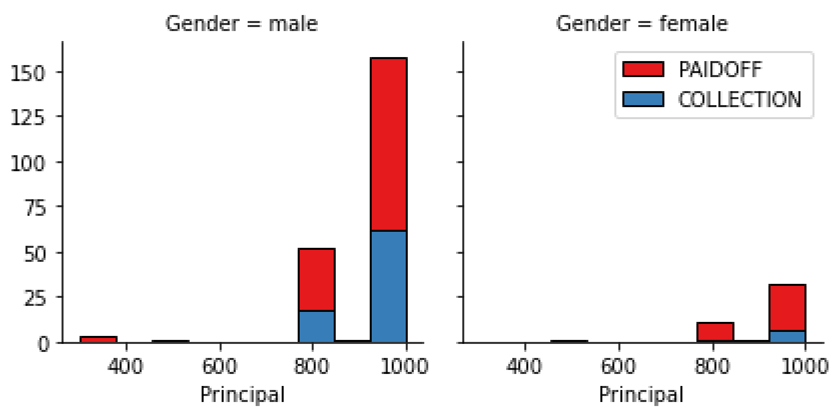
In [7]:
```
df['loan_status'].value_counts()
```

Out[7]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

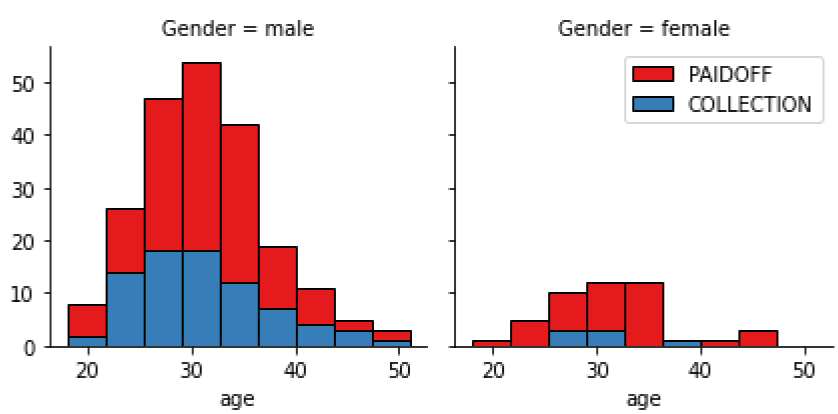
260 people have paid off the loan on time while 86 have gone into collection.

Lets plot some columns to understand data better.

In [8]:
```
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()
```



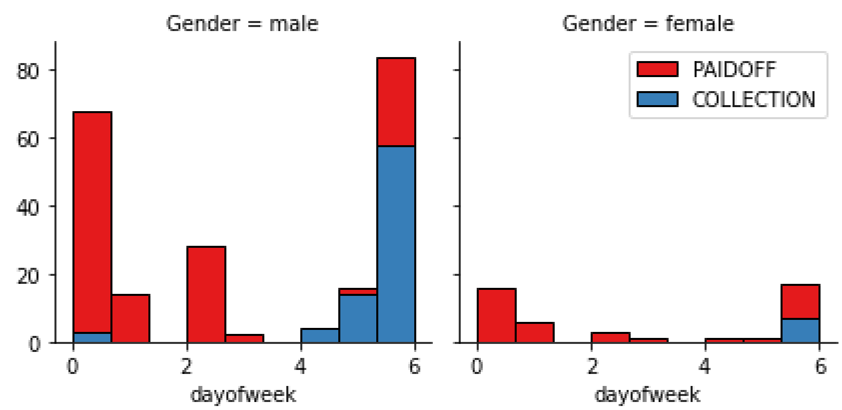
In [9]:
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2
g.map(plt.hist, 'age', bins=bins, ec="k")

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



2. Feature selection/extraction

Lets look at the day of the week people get the loan

In [10]:
```
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 [11]:
```
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

Out[11]:

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

3. Data Cleaning

Convert Categorical features to numerical values

Lets look at gender

In [12]:
```python
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[12]:
```
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 their loans and 73 % of males pay their loan.

Lets convert male to 0 and female to 1:

In [13]:
```python
df['Gender'].replace(to_replace=['male','female'],value=[0,1],inplace=True)
df.head()
```

Out[13]:

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

male = 0
female = 1

One Hot Encoding

How about education?

In [14]:
```
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[14]:
```
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
```

75 % of bachelor, 74 % of highschool or below, 50 % of master or above, and 76 % of college pay their loan.

Feature before One Hot Encoding

In [15]:
```
df[['Principal','terms','Gender','education']].head()
```

Out[15]:

	Principal	terms	Gender	education
0	1000	30	0	High School or Below
1	1000	30	1	Bechalor
2	1000	15	0	college
3	1000	30	1	college
4	1000	30	0	college

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

In [16]:
```
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[16]:

	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

Lets defined feature sets, X:

In [17]:
```
X = Feature
X[0:5]
```

Out[17]:

	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 labels?

In [18]:
```
y = df['loan_status'].values
y[0:5]
```

Out[18]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

Normalize Data

Data Standardization give data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases:

In [19]:
```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[19]:
```
array([[ 0.51578458,  0.92071769,  2.33152555, -0.42056004, -1.20577805,
        -0.38170062,  1.13639374, -0.86968108],
       [ 0.51578458,  0.92071769,  0.34170148,  2.37778177, -1.20577805,
         2.61985426, -0.87997669, -0.86968108],
       [ 0.51578458, -0.95911111, -0.65321055, -0.42056004, -1.20577805,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.48739188,  2.37778177,  0.82934003,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.3215732 , -0.42056004,  0.82934003,
        -0.38170062, -0.87997669,  1.14984679]])
```

4. Model Building: Classification Algorithms

Now, it is your turn, use the training set to build an accurate model. Then use the test set to report the accuracy of the model You should use the following algorithm:

- K Nearest Neighbor(KNN)
- Decision Tree

- Support Vector Machine
- Logistic Regression

__ Notice:__

- You can go above and change the pre-processing, feature selection, feature-extraction, and so on, to make a better model.
- You should use either scikit-learn, Scipy or Numpy libraries for developing the classification algorithms.
- You should include the code of the algorithm in the following cells.

K Nearest Neighbor(KNN)

Notice: You should find the best k to build the model with the best accuracy.
warning: You should not use the **loan_test.csv** for finding the best k, however, you can split your train_loan.csv into train and test to find the best **k**.

In [20]:
```
# We split the X into train (80%) test(20%) to find the best k
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=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,)
```

In [21]:
```
# Modeling
from sklearn.neighbors import KNeighborsClassifier
k = 3
#Train Model and Predict
kNN_model = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
kNN_model
```

Out[21]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [22]:
```
# just for sanity chaeck
yhat = kNN_model.predict(X_test)
yhat[0:5]
```

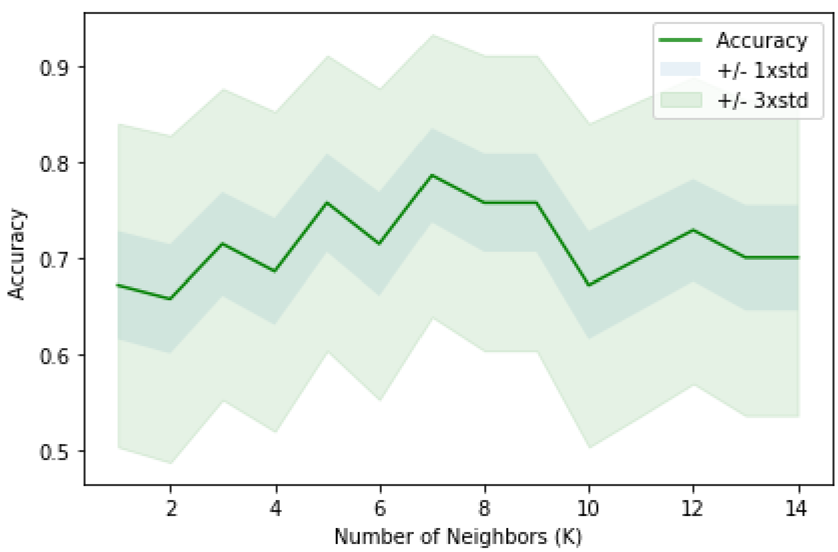
Out[22]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

Choose the best K

In [23]:
```
from sklearn.neighbors import KNeighborsClassifier
# Best k
Ks=15
mean_acc=np.zeros((Ks-1))
std_acc=np.zeros((Ks-1))
ConfustionMx=[];
for n in range(1,Ks):

    #Train Model and Predict
    kNN_model = KNeighborsClassifier(n_neighbors=n).fit(X_train,y_train)
    yhat = kNN_model.predict(X_test)


    mean_acc[n-1]=np.mean(yhat==y_test);

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

Out[23]:
```
array([0.67142857, 0.65714286, 0.71428571, 0.68571429, 0.75714286,
       0.71428571, 0.78571429, 0.75714286, 0.75714286, 0.67142857,
       0.7       , 0.72857143, 0.7       , 0.7       ])
```

In [24]:
```
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=
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [25]:
```
print( "The best accuracy was with", mean_acc.max(), "with k =", mean_acc.argmax(
```

```
The best accuracy was with 0.7857142857142857 with k = 7
```

In [26]:
```
# Building the model again, using k=7
from sklearn.neighbors import KNeighborsClassifier
k = 7
#Train Model and Predict
kNN_model = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
kNN_model
```

Out[26]:
```
KNeighborsClassifier(n_neighbors=7)
```

Decision Tree

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
DT_model = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
DT_model.fit(X_train,y_train)
DT_model
```

Out[27]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [28]:
```
yhat = DT_model.predict(X_test)
yhat
```

Out[28]:
```
array(['COLLECTION', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'COLLECTION', 'COLLECTION', 'PAIDOFF',
       'COLLECTION', 'COLLECTION', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'COLLECTION', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'COLLECTION', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF'], dtype=object)
```

Support Vector Machine

In [29]:
```
from sklearn import svm
SVM_model = svm.SVC()
SVM_model.fit(X_train, y_train)
```

Out[29]:
```
SVC()
```

In [30]:
```
yhat = SVM_model.predict(X_test)
yhat
```

Out[30]:
```
array(['COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'COLLECTION',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

Logistic Regression

In [31]:
```
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression(C=0.01).fit(X_train,y_train)
LR_model
```

Out[31]:
```
LogisticRegression(C=0.01)
```

In [32]:
```
yhat = LR_model.predict(X_test)
yhat
```

Out[32]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'], dtype=object)
```

5. Model Evaluation

In [33]:
```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

Load Test set for evaluation

In [34]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[34]:

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

In [35]:
```
## Preprocessing
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'])
test_df['dayofweek'] = test_df['effective_date'].dt.dayofweek
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
test_df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
test_Feature = test_df[['Principal','terms','age','Gender','weekend']]
test_Feature = pd.concat([test_Feature,pd.get_dummies(test_df['education'])], axi
test_Feature.drop(['Master or Above'], axis = 1,inplace=True)
test_X = preprocessing.StandardScaler().fit(test_Feature).transform(test_Feature)
test_X[0:5]
```

Out[35]:
```
array([[ 0.49362588,  0.92844966,  3.05981865,  1.97714211, -1.30384048,
         2.39791576, -0.79772404, -0.86135677],
       [-3.56269116, -1.70427745,  0.53336288, -0.50578054,  0.76696499,
        -0.41702883, -0.79772404, -0.86135677],
       [ 0.49362588,  0.92844966,  1.88080596,  1.97714211,  0.76696499,
        -0.41702883,  1.25356634, -0.86135677],
       [ 0.49362588,  0.92844966, -0.98251057, -0.50578054,  0.76696499,
        -0.41702883, -0.79772404,  1.16095912],
       [-0.66532184, -0.78854628, -0.47721942, -0.50578054,  0.76696499,
         2.39791576, -0.79772404, -0.86135677]])
```

In [36]:
```
test_y = test_df['loan_status'].values
test_y[0:5]
```

Out[36]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

In [37]:
```
knn_yhat = kNN_model.predict(test_X)
print("KNN Jaccard index: %.2f" % jaccard_score(test_y, knn_yhat,pos_label='PAIDC
print("KNN F1-score: %.2f" % f1_score(test_y, knn_yhat, average='weighted') )
```

```
KNN Jaccard index: 0.65
KNN F1-score: 0.63
```

In [38]:
```
DT_yhat = DT_model.predict(test_X)
print("DT Jaccard index: %.2f" % jaccard_score(test_y, DT_yhat,pos_label='PAIDOFF
print("DT F1-score: %.2f" % f1_score(test_y, DT_yhat, average='weighted') )
```

```
DT Jaccard index: 0.66
DT F1-score: 0.74
```

In [39]:
```
SVM_yhat = SVM_model.predict(test_X)
print("SVM Jaccard index: %.2f" % jaccard_score(test_y, SVM_yhat,pos_label='PAIDO
print("SVM F1-score: %.2f" % f1_score(test_y, SVM_yhat, average='weighted') )
```

```
SVM Jaccard index: 0.78
SVM F1-score: 0.76
```

In [40]:
```
LR_yhat = LR_model.predict(test_X)
LR_yhat_prob = LR_model.predict_proba(test_X)
print("LR Jaccard index: %.2f" % jaccard_score(test_y, LR_yhat,pos_label='PAIDOFF
print("LR F1-score: %.2f" % f1_score(test_y, LR_yhat, average='weighted') )
print("LR LogLoss: %.2f" % log_loss(test_y, LR_yhat_prob))
```

```
LR Jaccard index: 0.74
LR F1-score: 0.63
LR LogLoss: 0.52
```

Report

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.65	0.63	NA
Decision Tree	0.66	0.74	NA
SVM	0.78	0.76	NA
LogisticRegression	0.74	0.63	0.52

Date	Author
08-04-2021	**Ehsan Zia**