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
```

In [3]:

```
df = pd.read_csv('loan_train.csv')
df.head()
```

Out[3]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	educatio
0	0	0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45	Higl School o Belov
1	2	2	PAIDOFF	1000	30	9/8/2016	10/7/2016	33	Bechalo
2	3	3	PAIDOFF	1000	15	9/8/2016	9/22/2016	27	colleg
3	4	4	PAIDOFF	1000	30	9/9/2016	10/8/2016	28	colleg
4	6	6	PAIDOFF	1000	30	9/9/2016	10/8/2016	29	colleg

In [4]:

```
df.shape
```

Out[4]:

```
(346, 10)
```

Convert to date time object

In [5]:

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

Out[5]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	educatio
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School o Belov
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalo
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	colleg
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	colleg
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	colleg

In [8]:

```
df.dtypes
```

Out[8]:

```
Unnamed: 0                 int64
Unnamed: 0.1               int64
loan_status               object
Principal                  int64
terms                      int64
effective_date    datetime64[ns]
due_date          datetime64[ns]
age                        int64
education                 object
Gender                    object
dtype: object
```

Data visualization and pre-processing

Let’s see how many of each class is in our data set

In [9]:

```
df['loan_status'].value_counts()
```

Out[9]:

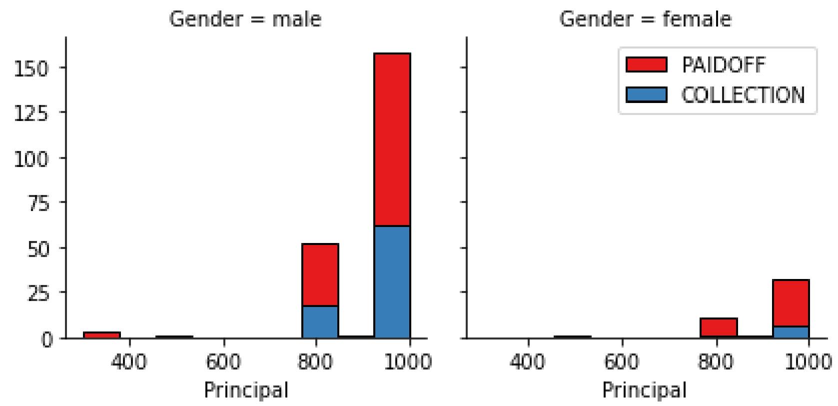
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

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

Lets plot some columns to underestand data better:

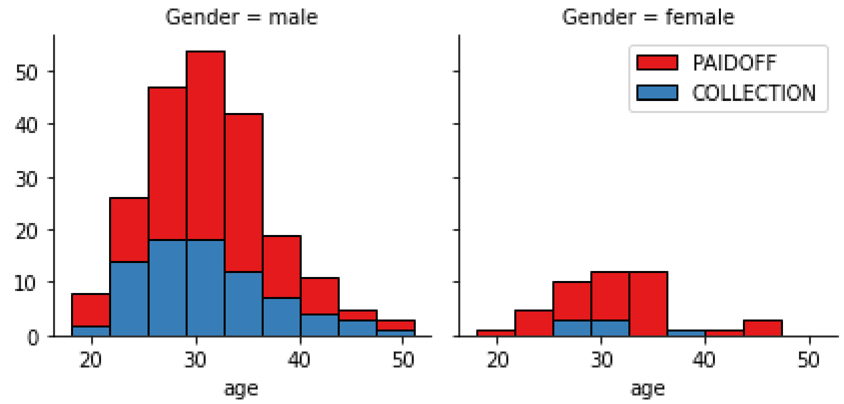
In [10]:

```
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 [11]:

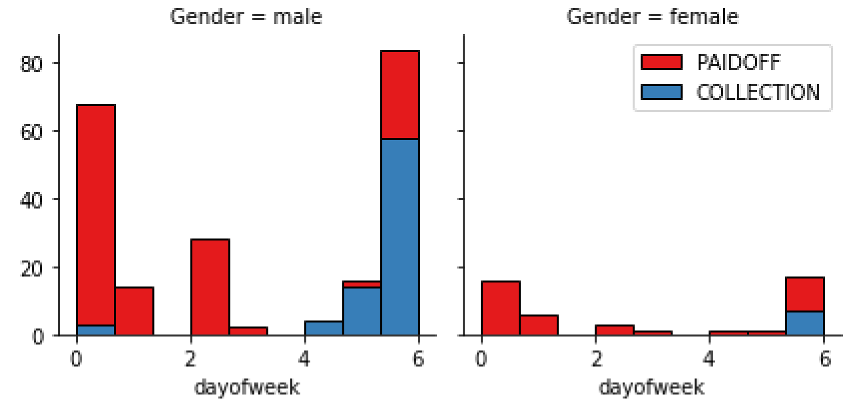
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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



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

In [12]:

```
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 [13]:

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

Out[13]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	educatioi
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	Higl School o Belov
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalo
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

Convert Categorical features to numerical values

In [14]:

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

Out[14]:

```
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

In [17]:

```
df['Gender'].replace({'Female':1,'Male':0},inplace=True)
df
```

Out[17]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	educ
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	Sch E
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bec
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	cc
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	cc
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	cc
...	...	...	...	...	...	...	...	...	
341	394	394	COLLECTION	800	15	2016-09-11	2016-09-25	32	Sch E
342	395	395	COLLECTION	1000	30	2016-09-11	2016-10-10	25	Sch E
343	397	397	COLLECTION	800	15	2016-09-12	2016-09-26	39	cc
344	398	398	COLLECTION	1000	30	2016-09-12	2016-11-10	28	cc
345	399	399	COLLECTION	1000	30	2016-09-12	2016-10-11	26	cc

346 rows × 12 columns

One Hot Encoding How about education?

In [18]:

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

Out[18]:

```
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
```

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

In [19]:

```
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[19]:

	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

In [20]:

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

Out[20]:

	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

In [21]:

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

Out[21]:

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

Normalize Data

In [22]:

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

Out[22]:

```
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]])
```

Classification

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.

Train Test split

In [23]:

```
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=
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,)
```

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 [24]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Checking for the best value of K

In [25]:

```
for k in range(1, 10):
    knn_model  = KNeighborsClassifier(n_neighbors = k).fit(x_train, y_train)
    knn_yhat = knn_model.predict(x_test)
    print("For K = {} accuracy = {}".format(k,accuracy_score(y_test,knn_yhat)))
```

```
For K = 1 accuracy = 0.6714285714285714
For K = 2 accuracy = 0.6571428571428571
For K = 3 accuracy = 0.7142857142857143
For K = 4 accuracy = 0.6857142857142857
For K = 5 accuracy = 0.7571428571428571
For K = 6 accuracy = 0.7142857142857143
For K = 7 accuracy = 0.7857142857142857
For K = 8 accuracy = 0.7571428571428571
For K = 9 accuracy = 0.7571428571428571
```

In [26]:

```
print("We can see that the KNN model is the best for K=7")
```

```
We can see that the KNN model is the best for K=7
```

Building the model with the best value of K = 7

In [27]:

```
best_knn_model = KNeighborsClassifier(n_neighbors = 7).fit(x_train, y_train)
best_knn_model
```

Out[27]:

```
KNeighborsClassifier(n_neighbors=7)
```

In [29]:

```
## Evaluation Metric

from sklearn.metrics import f1_score

print("Train set Accuracy (F1): ", f1_score(y_train, best_knn_model.predict(x_train), a
print("Test set Accuracy (F1): ", f1_score(y_test, best_knn_model.predict(x_test), aver
```

```
Train set Accuracy (F1):  0.8000194668761034
Test set Accuracy (F1):  0.7766540244416351
```

Decision Tree

In [30]:

```
# importing libraries
from sklearn.tree import DecisionTreeClassifier
```

In [31]:

```
for d in range(1,10):
    dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = d).fit(x_train, y_tr
    dt_yhat = dt.predict(x_test)
    print("For depth = {}  the accuracy score is {} ".format(d, accuracy_score(y_test,
```

```
For depth = 1  the accuracy score is 0.7857142857142857
For depth = 2  the accuracy score is 0.7857142857142857
For depth = 3  the accuracy score is 0.6142857142857143
For depth = 4  the accuracy score is 0.6142857142857143
For depth = 5  the accuracy score is 0.6428571428571429
For depth = 6  the accuracy score is 0.7714285714285715
For depth = 7  the accuracy score is 0.7571428571428571
For depth = 8  the accuracy score is 0.7571428571428571
For depth = 9  the accuracy score is 0.6571428571428571
```

In [32]:

```
print("The best value of depth is d = 2 ")
```

```
The best value of depth is d = 2
```

In [33]:

```
## Creating the best model for decision tree with best value of depth 2

best_dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 2).fit(x_tra
best_dt_model
```

Out[33]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

In [36]:

```

from sklearn.metrics import f1_score


print("Train set Accuracy (F1): ", f1_score(y_train, best_dt_model.predict(x_train), av
print("Test set Accuracy (F1): ", f1_score(y_test, best_dt_model.predict(x_test), avera
```

```
Train set Accuracy (F1):  0.6331163939859591
Test set Accuracy (F1):  0.6914285714285714
```

Support Vector Machine

In [37]:

```
#importing svm
from sklearn import svm
from sklearn.metrics import f1_score
```

In [38]:

```
for k in ('linear', 'poly', 'rbf','sigmoid'):
    svm_model = svm.SVC( kernel = k).fit(x_train,y_train)
    svm_yhat = svm_model.predict(x_test)
    print("For kernel: {}, the f1 score is: {}".format(k,f1_score(y_test,svm_yhat, aver
```

```
For kernel: linear, the f1 score is: 0.6914285714285714
For kernel: poly, the f1 score is: 0.7064793130366899
For kernel: rbf, the f1 score is: 0.7275882012724117
For kernel: sigmoid, the f1 score is: 0.6892857142857144
```

In [39]:

```
print("We can see the rbf has the best f1 score ")
```

```
We can see the rbf has the best f1 score
```

In [40]:

```
## building best SVM with kernel = rbf
best_svm = svm.SVC(kernel='rbf').fit(x_train,y_train)
best_svm
```

Out[40]:

```
SVC()
```

In [42]:

```
## Evaluation Metrics
# jaccard score and f1 score
from sklearn.metrics import f1_score


print("Train set Accuracy (F1): ", f1_score(y_train, best_svm.predict(x_train), average
print("Test set Accuracy (F1): ", f1_score(y_test, best_svm.predict(x_test), average='w
```

```
Train set Accuracy (F1):  0.7682165861513688
Test set Accuracy (F1):  0.7275882012724117
```

Logistic Regression

In [43]:

```
# importing libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
```

In [44]:

```
for k in ('lbfgs', 'saga', 'liblinear', 'newton-cg', 'sag'):
    lr_model = LogisticRegression(C = 0.01, solver = k).fit(x_train, y_train)
    lr_yhat = lr_model.predict(x_test)
    y_prob = lr_model.predict_proba(x_test)
    print('When Solver is {}, logloss is : {}'.format(k, log_loss(y_test, y_prob)))
```

```
When Solver is lbfgs, logloss is : 0.4920179847937498
When Solver is saga, logloss is : 0.49201758723624645
When Solver is liblinear, logloss is : 0.5772287609479654
When Solver is newton-cg, logloss is : 0.4920178014679269
When Solver is sag, logloss is : 0.492007407853154
```

In [45]:

```
print("We can see that the best solver is liblinear")
```

```
We can see that the best solver is liblinear
```

In [46]:

```
# Best logistic regression model with liblinear solver

best_lr_model = LogisticRegression(C = 0.01, solver = 'liblinear').fit(x_train, y_trair
best_lr_model
```

Out[46]:

```
LogisticRegression(C=0.01, solver='liblinear')
```

In [47]:

```
## Evaluation Metrics
# jaccard score and f1 score
from sklearn.metrics import f1_score

print("Train set Accuracy (F1): ", f1_score(y_train, best_lr_model.predict(x_train), av
print("Test set Accuracy (F1): ", f1_score(y_test, best_lr_model.predict(x_test), avera
```

```
Train set Accuracy (F1):  0.7341146337750953
Test set Accuracy (F1):  0.6670522459996144
```

Model Evaluation using Test set

In [50]:

```

from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

Load Test set for evaluation

In [52]:

```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[52]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	educatio
0	1	1	PAIDOFF	1000	30	9/8/2016	10/7/2016	50	Bechalo
1	5	5	PAIDOFF	300	7	9/9/2016	9/15/2016	35	Master o Abov
2	21	21	PAIDOFF	1000	30	9/10/2016	10/9/2016	43	Hig School o Belov
3	24	24	PAIDOFF	1000	30	9/10/2016	10/9/2016	26	colleg
4	35	35	PAIDOFF	800	15	9/11/2016	9/25/2016	29	Bechalo

In [63]:

```
# data processing
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)

Feature1 = test_df[['Principal','terms','age','Gender','weekend']]
Feature1 = pd.concat([Feature1,pd.get_dummies(test_df['education'])], axis=1)
Feature1.drop(['Master or Above'], axis = 1,inplace=True)


x_loan_test = Feature1
x_loan_test = preprocessing.StandardScaler().fit(x_loan_test).transform(x_loan_test)

y_loan_test = test_df['loan_status'].values
```

In [64]:

```
from sklearn.metrics import f1_score
```

In [69]:

```
# F1_score

# KNN
knn_yhat = best_knn_model.predict(x_loan_test)
f1 = round(f1_score(y_loan_test, knn_yhat, average = 'weighted'), 2)

# Decision Tree
dt_yhat = best_dt_model.predict(x_loan_test)
f2 = round(f1_score(y_loan_test, dt_yhat, average = 'weighted'), 2)

# Support Vector Machine
svm_yhat = best_svm.predict(x_loan_test)
f3 = round(f1_score(y_loan_test, svm_yhat, average = 'weighted'), 2)

# Logistic Regression
lr_yhat = best_lr_model.predict(x_loan_test)
f4 = round(f1_score(y_loan_test, lr_yhat, average = 'weighted'), 2)

f1_list = [f1, f2, f3, f4]
f1_list

```

Out[69]:

```
[0.63, 0.63, 0.76, 0.66]
```

In [70]:

```
# log loss

# Logistic Regression
lr_prob = best_lr_model.predict_proba(x_loan_test)
ll_list = ['NA','NA','NA', round(log_loss(y_loan_test, lr_prob), 2)]
ll_list
```

Out[70]:

```
['NA', 'NA', 'NA', 0.57]
```

In [71]:

```
columns = ['KNN', 'Decision Tree', 'SVM', 'Logistic Regression']
index = [ 'F1-score', 'Logloss']

accuracy_df = pd.DataFrame([ ll_list], index = index, columns = columns)
accuracy_df1 = accuracy_df.transpose()
accuracy_df1.columns.name = 'Algorithm'
accuracy_df1
```

Out[71]:

Algorithm	F1-score	Logloss
KNN	NA	NA
Decision Tree	NA	NA
SVM	NA	NA
Logistic Regression	0.57	0.57

Report

You should be able to report the accuracy of the built model using different evaluation metrics:

In [ ]:

```

```