The following notebook is an analysis and building of a model of dataset of approximately 300 student loans taken to pay for post secondary education.

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.

Lets first load required libraries:

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

Lets download the dataset

In [2]:

```
!wget -O loan_train.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses
```

```
--2019-06-07 18:07:40--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-course
s-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.sof
```

```
tlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorag
e.softlayer.net)|67.228.254.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23101 (23K) [text/csv]
Saving to: ‘loan_train.csv’

100%[=====================================>] 23,101      --.-K/s   in 0.002s

2019-06-07 18:07:40 (11.0 MB/s) - ‘loan_train.csv’ saved [23101/23101]
```

Load Data From CSV File

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	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 [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	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

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	G
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	

Data visualization and pre-processing

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

In [6]:

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

Out[6]:

```
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 [7]:

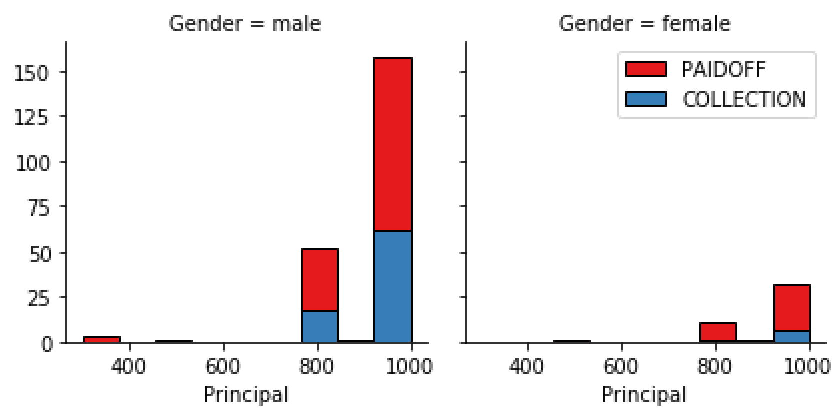
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
Solving environment: done

## Package Plan ##

  environment location: /opt/conda/envs/DSX-Python35

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2019.5.15  |                0         133 KB  anaconda
    seaborn-0.9.0              |           py35_0         378 KB  anaconda
    certifi-2018.8.24          |           py35_1         139 KB  anaconda
    openssl-1.0.2s             |       h7b6447c_0         3.1 MB  anaconda
    ------------------------------------------------------------
                                           Total:         3.8 MB

The following packages will be UPDATED:

    ca-certificates: 2019.1.23-0          --> 2019.5.15-0       anaconda
    certifi:         2018.8.24-py35_1     --> 2018.8.24-py35_1  anaconda
    openssl:         1.0.2s-h7b6447c_0    --> 1.0.2s-h7b6447c_0 anaconda
    seaborn:         0.8.0-py35h15a2772_0 --> 0.9.0-py35_0      anaconda
```

```
Downloading and Extracting Packages
ca-certificates-2019 | 133 KB    | ##################################### | 100%
seaborn-0.9.0        | 378 KB    | ##################################### | 100%
certifi-2018.8.24    | 139 KB    | ##################################### | 100%
openssl-1.0.2s       | 3.1 MB    | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

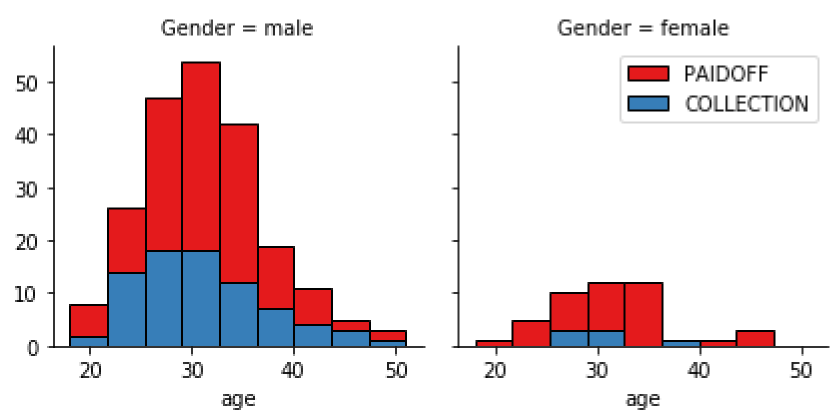
In [8]:

```python
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]:

```python
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()
```

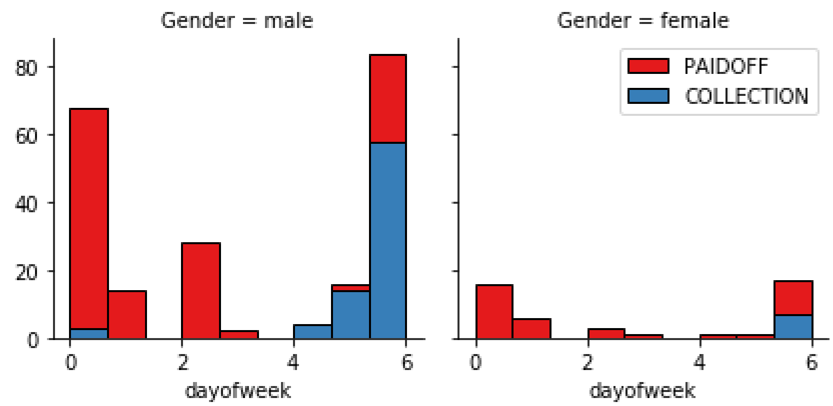


Pre-processing: Feature selection/extraction

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	G
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	

Convert Categorical features to numerical values

Lets look at gender:

In [12]:

```
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 there loans while only 73 % of males pay there loan

Lets convert male to 0 and female to 1:

In [13]:

```
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	G
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	

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
```

Feature befor One Hot Encoding

In [15]:

```
df[['Principal','terms','age','Gender','education']].head()
```

Out[15]:

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

Use one hot encoding technique to conver categorical varables to 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 defind 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 lables?

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 (technically should be done after train test split)

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]])
```

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.

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 [22]:

```
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_sta
print ('Training 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 [27]:

```
from sklearn.neighbors import KNeighborsClassifier
k = 3
kNN_model = KNeighborsClassifier(n_neighbors=k).fit(x_train,y_train)
kNN_model
yhat = kNN_model.predict(x_test)
yhat[0:5]
```

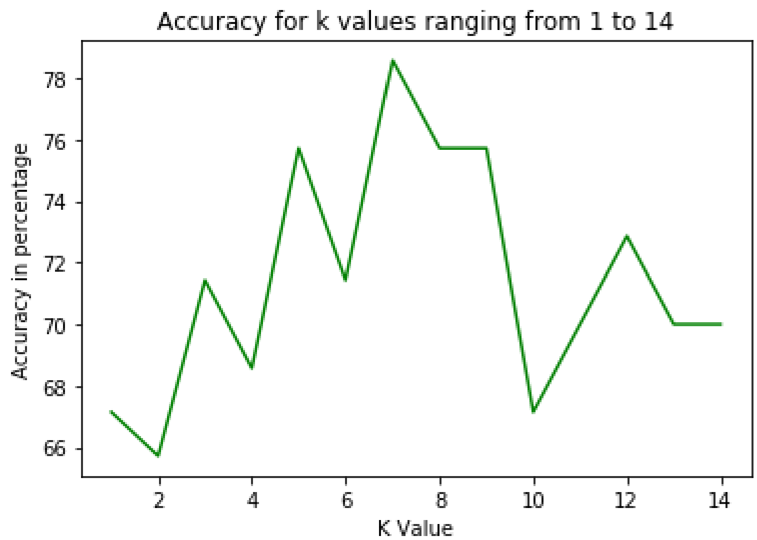
Out[27]:

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

In [146…

```
Ktestnum=15
mean_accuracy=np.zeros((Ktestnum-1))
std_accuracy=np.zeros((Ktestnum-1))
ConfustionMx=[];
for n in range(1,Ktestnum):

    #Train Model and Predict
    KNN = KNeighborsClassifier(n_neighbors=n).fit(x_train,y_train)
    yhat = KNN.predict(x_test)


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

    std_accuracy[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])
print(mean_accuracy)

KWithSeven = KNeighborsClassifier(n_neighbors=7).fit(x_train,y_train)
print(KWithSeven)
yhat=KWithSeven.predict(x_test)
print(yhat==y_test)
plt.plot(range(1,Ktestnum),mean_accuracy*100,'g')
plt.title('Accuracy for k values ranging from 1 to 14')
plt.ylabel('Accuracy in percentage')
plt.xlabel('K Value')
print('The best accuracy is ', np.mean(yhat==y_test), ' where k is equal to 7 as cle
```

```
[ 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       ]
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=7, p=2,
           weights='uniform')
[ True  True  True  True  True  True  True False  True  True False False
  True  True  True  True  True  True  True  True  True  True  True  True
  True False False  True  True  True  True  True  True  True  True  True
  True  True False False False  True False  True  True  True  True  True
 False  True  True  True  True  True  True  True  True False  True  True
 False  True False  True  True False  True  True False  True]
The best accuracy is  0.785714285714  where k is equal to 7 as clearly demonstrated
in the plot below:
```



Decision Tree

In [63]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

x_train, x_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_sta
print ('Training set:', x_train.shape,  y_train.shape)
print ('Test set:', x_test.shape,  y_test.shape)

DecTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
DecTree.fit(x_train,y_train)
print(DecTree)
```

```
Training set: (276, 8) (276,)
Test set: (70, 8) (70,)
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            max_features=None, 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=None,
            splitter='best')
```

In [78]:

```
yhat = DecTree.predict(x_test)
print(yhat)
print(np.mean(yhat==y_test))
```

```
['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']
0.785714285714
```

In [ ]:

Support Vector Machine

In [79]:

```
from sklearn import svm
SVM = svm.SVC()
SVM.fit(x_train, y_train)

yhat = SVM.predict(x_test)
print(yhat)
print(np.mean(yhat==y_test))
```

```
['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']
0.742857142857
```

In [ ]:

In [ ]:

Logistic Regression

In [80]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LogReg = LogisticRegression(C=0.01).fit(x_train,y_train)
print(LogReg)

yhat = LogReg.predict(x_test)
print(yhat)
print(np.mean(yhat==y_test))
```

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
['COLLECTION' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'COLLECTION'
 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'COLLECTION' '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']
0.685714285714
```

In [ ]:

In [ ]:

Model Evaluation using Test set

In [106…

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

First, download and load the test set:

In [82]:

```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-
```

```
--2019-06-07 18:50:55--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-course
s-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.sof
tlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorag
e.softlayer.net)|67.228.254.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3642 (3.6K) [text/csv]
Saving to: ‘loan_test.csv’

100%[======================================>] 3,642       --.-K/s   in 0s

2019-06-07 18:50:55 (673 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

Load Test set for evaluation

In [119…

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

Out[119…

	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

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
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 [120…

```python
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'])], axis=1
test_feature.drop(['Master or Above'], axis = 1,inplace=True)

print(test_feature.head())


test_x = preprocessing.StandardScaler().fit(test_feature).transform(test_feature)
test_x[0:5]

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

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

   college
0        0
1        0
2        0
3        1
4        0
```

Out[120…

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

In [126…

```python
#Find all the Jaccard similarity scores for the 4 models we tested and add to an arr
knn_yhat = KWithSeven.predict(test_x)
jaccard1 = round(jaccard_similarity_score(test_y, knn_yhat),2)

dt_yhat = DecTree.predict(test_x)
jaccard2 = round(jaccard_similarity_score(test_y, dt_yhat),2)

svm_yhat = SVM.predict(test_x)
jaccard3 = round(jaccard_similarity_score(test_y, svm_yhat),2)

lr_yhat = LogReg.predict(test_x)
jaccard4 = round(jaccard_similarity_score(test_y, lr_yhat),2)

list_jc = [jaccard1, jaccard2, jaccard3, jaccard4]
list_jc
print(list_jc)
```

```
fs1 = round(f1_score(test_y, knn_yhat, average='weighted'), 2)
fs2 = round(f1_score(test_y, dt_yhat, average='weighted'), 2)
fs3 = round(f1_score(test_y, svm_yhat, average='weighted'), 2)
fs4 = round(f1_score(test_y, lr_yhat, average='weighted'), 2)

list_fs = [fs1, fs2, fs3, fs4]
print (list_fs)

LogReg_prob = LogReg.predict_proba(test_x)
LogReg_yhat_prob = LogReg.predict_proba(test_x)

list_ll = ['NA', 'NA', 'NA', round(log_loss(test_y, LogReg_yhat_prob), 2)]
list_ll
print (list_ll)
```

```
[0.6700000000000004, 0.7399999999999999, 0.8000000000000004, 0.7399999999999999]
[0.63, 0.63, 0.7600000000000001, 0.6600000000000003]
['NA', 'NA', 'NA', 0.5699999999999995]
```

```
/opt/conda/envs/DSX-Python35/lib/python3.5/site-packages/sklearn/metrics/classificat
ion.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in
labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [128…

```
df = pd.DataFrame(list_jc, index=['KNN','Decision Tree','SVM','Logistic Regression']
df.columns = ['Jaccard']
df.insert(loc=1, column='F1 Score', value=list_fs)
df.insert(loc=2, column='LogLoss', value=list_ll)
df.columns.name = 'Algorithm'
df
```

Out[128…

Algorithm	Jaccard	F1 Score	LogLoss
KNN	0.67	0.63	NA
Decision Tree	0.74	0.63	NA
SVM	0.80	0.76	NA
Logistic Regression	0.74	0.66	0.57

Report

It is observable during testing the highest accuracy came from using a Support Vector Model with a Jaccard score of 0.80 and F1-score of 0.76.

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.67	0.63	NA
Decision Tree	0.74	0.63	NA
SVM	0.80	0.76	NA
LogisticRegression	0.74	0.66	0.57