In [2]:
```python
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 [3]:
```
!wget -O loan_train.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
```

```
--2019-07-11 00:43:34--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.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

2019-07-11 00:43:34 (12.8 MB/s) - 'loan_train.csv' saved [23101/23101]
```

Load Data From CSV File

In [4]:
```python
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	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 [5]:
```python
df.shape
```

Out[5]:
```
(346, 10)
```

Convert to date time object

In [6]:
```python
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	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

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

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

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

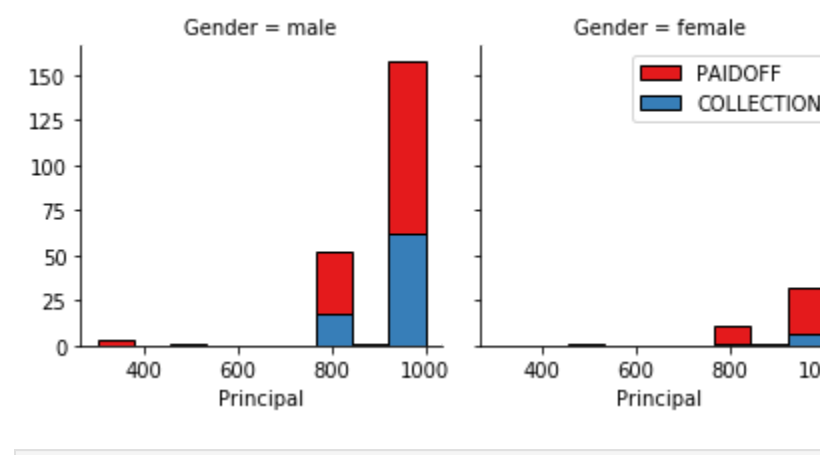
260 people have paid off the loan on time while 86 have gone into collection

Lets plot some columns to underestand data better:

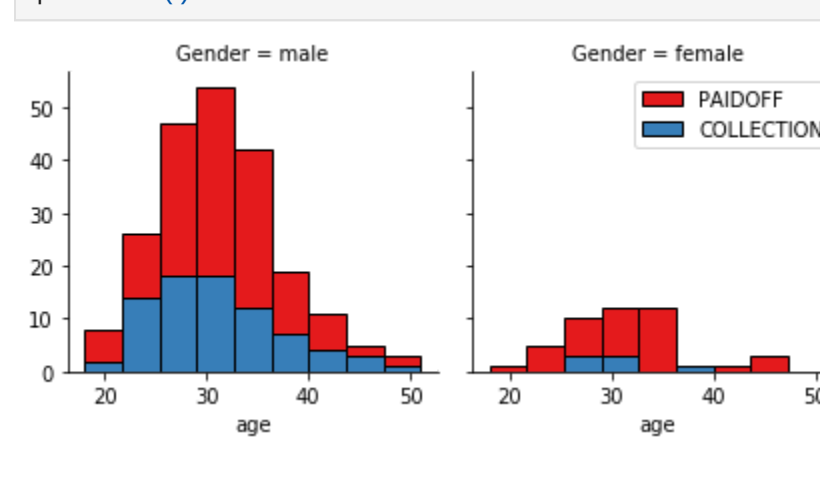
In [7]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
Solving environment: done

## Package Plan ##

  environment location: /opt/conda/envs/Python36

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2019.5.15  |                0         133 KB  anaconda
    openssl-1.1.1              |       h7b6447c_0         5.0 MB  anaconda
    certifi-2019.6.16          |           py36_0         154 KB  anaconda
    seaborn-0.9.0              |           py36_0         379 KB  anaconda
    ------------------------------------------------------------
                                           Total:         5.7 MB

The following packages will be UPDATED:

    ca-certificates: 2019.5.15-0       --> 2019.5.15-0      anaconda
    certifi:         2019.6.16-py36_0  --> 2019.6.16-py36_0 anaconda
    openssl:         1.1.1c-h7b6447c_1 --> 1.1.1-h7b6447c_0 anaconda
    seaborn:         0.9.0-py36_0      --> 0.9.0-py36_0     anaconda


Downloading and Extracting Packages
ca-certificates-2019 | 133 KB    | ##################################### | 100%
openssl-1.1.1        | 5.0 MB    | ##################################### | 100%
certifi-2019.6.16    | 154 KB    | ##################################### | 100%
seaborn-0.9.0        | 379 KB    | ##################################### | 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()
```



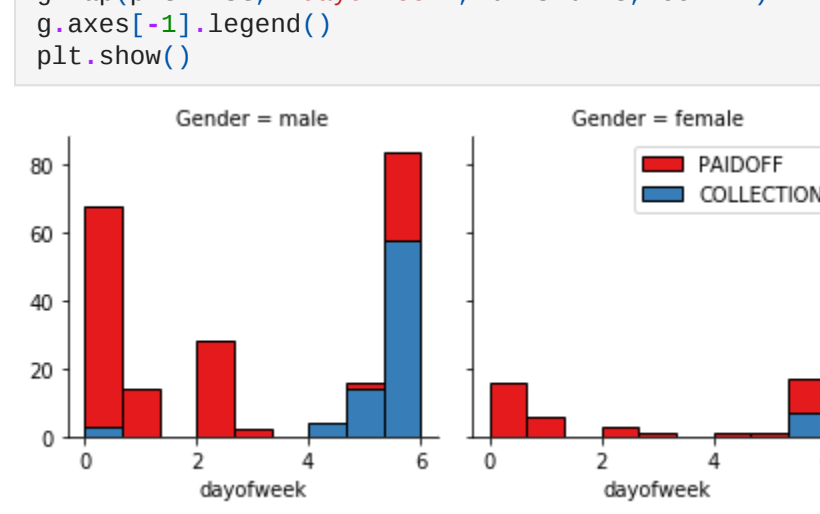
In [9]:
```python
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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



Pre-processing: Feature selection/extraction

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

In [9]:
```python
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 [16]:
```python
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

Out[16]:

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

Convert Categorical features to numerical values

Lets look at gender:

In [17]:
```python
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[17]:
```
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 [18]:
```python
df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
df.head()
```

Out[18]:

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

One Hot Encoding

How about education?

In [19]:
```python
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[19]:
```
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 [20]:
```python
df[['Principal','terms','age','Gender','education']].head()
```

Out[20]:

	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 [21]:
```python
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[21]:

	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 [36]:
```python
X = Feature
X[0:5]
```

Out[36]:

	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 [45]:
```python
y = pd.get_dummies(df['loan_status'])['PAIDOFF'].values
y[0:5]
```

Out[45]:
```
array([1, 1, 1, 1, 1], dtype=uint8)
```

Normalize Data

Data Standardization give data zero mean and unit variance (technically should be done after train test split)

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

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:

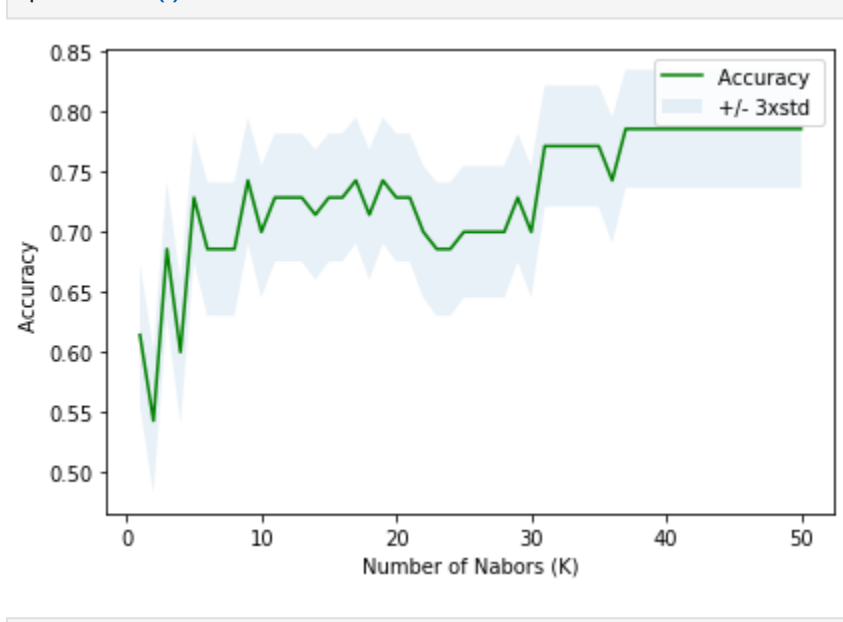
- You can go above and change the pre-processing, feature selection, feature-extraction, and so on, to make a better model.
- You should use either scikit-learn, Scipy or Numpy libraries for developing the classification algorithms.
- You should include the code of the algorithm in the following cells.

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 [48]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
```

In [49]:
```python
mean_acc=np.zeros(50)
std_acc = np.zeros(50)
for n in range(1,51):
    knnmodel=KNeighborsClassifier(n_neighbors=n).fit(X_train,y_train)
    y_pred=knnmodel.predict(X_test)
    mean_acc[n-1]=metrics.accuracy_score(y_test,y_pred)
    std_acc[n-1]=np.std(y_pred==y_test)/np.sqrt(y_pred.shape[0])

plt.plot(range(1,51),mean_acc,'g')
plt.fill_between(range(1,51),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()
plt.show()
```



In [50]:
```python
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.7857142857142857 with k= 37
```

Decision Tree

In [51]:
```python
from sklearn.tree import DecisionTreeClassifier
```

In [79]:
```python
dtmodel = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
dtmodel.fit(X_train,y_train)
```

Out[79]:
```
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 [80]:
```python
y_pred=dtmodel.predict(X_test)
TreeAccuracy=metrics.accuracy_score(y_test,y_pred)
TreeAccuracy
```

Out[80]:
```
0.6142857142857143
```

Support Vector Machine

In [68]:
```python
from sklearn import svm
svmmodel=svm.SVC(kernel='rbf')
svmmodel.fit(X_train,y_train)
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[68]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [75]:
```python
y_pred=svmmodel.predict(X_test)
y_pred
```

Out[75]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1], dtype=uint8)
```

In [76]:
```python
metrics.accuracy_score(y_test,y_pred)
```

Out[76]:
```
0.7571428571428571
```

Logistic Regression

In [81]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
lrmodel=LogisticRegression(C=0.01,solver='liblinear').fit(X_train,y_train)
```

In [59]:
```python
y_pred=lrmodel.predict(X_test)
print(y_pred)
print(LR.predict_proba(X_test))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[[0.32295451 0.67704549]
 [0.25028458 0.74971542]
 [0.18058968 0.81941032]
 [0.31842814 0.68157186]
 [0.23877924 0.76122076]
 [0.20537939 0.79462061]
 [0.27303975 0.72696025]
 [0.24257638 0.75742362]
 [0.31842814 0.68157186]
 [0.29873276 0.70126724]
 [0.30139145 0.69860855]
 [0.31870506 0.68129494]
 [0.29486638 0.70513362]
 [0.29712705 0.70287295]
 [0.13826387 0.86173613]
 [0.16477552 0.83522448]
 [0.3961522  0.6038478 ]
 [0.18361846 0.81638154]
 [0.33624938 0.66375062]
 [0.22107799 0.77892201]
 [0.28403971 0.71596029]
 [0.31011812 0.68988188]
 [0.3587312  0.6412688 ]
 [0.36668443 0.63331557]
 [0.21036655 0.78963345]
 [0.34061581 0.65938419]
 [0.34961925 0.65038075]
 [0.1467569  0.8532431 ]
 [0.34519783 0.65480217]
 [0.1398959  0.8601041 ]
 [0.22240276 0.77759724]
 [0.34397153 0.65602847]
 [0.28135638 0.71864362]
 [0.27709473 0.72290527]
 [0.21009772 0.78990228]
 [0.19606528 0.80393472]
 [0.33624938 0.66375062]
 [0.14255195 0.85744805]
 [0.31313687 0.68686313]
 [0.21412393 0.78587607]
 [0.35425731 0.64574269]
 [0.22801433 0.77198567]
 [0.16927732 0.83072268]
 [0.34061581 0.65938419]
 [0.18966569 0.81033431]
 [0.34010247 0.65989753]
 [0.22936843 0.77063157]
 [0.31870506 0.68129494]
 [0.18986182 0.81013818]
 [0.18629367 0.81370633]
 [0.24092811 0.75907189]
 [0.20179757 0.79820243]
 [0.2389319  0.7610681 ]
 [0.29333468 0.70666532]
 [0.26315166 0.73684834]
 [0.10211024 0.89788976]
 [0.20766284 0.79233716]
 [0.24532558 0.75467442]
 [0.22240276 0.77759724]
 [0.27278662 0.72721338]
 [0.1154932  0.8845068 ]
 [0.27683928 0.72316072]
 [0.19606528 0.80393472]
 [0.40572037 0.59427963]
 [0.23510262 0.76489738]
 [0.31011812 0.68988188]
 [0.27683928 0.72316072]
 [0.20154559 0.79845441]
 [0.16374879 0.83625121]
 [0.23510262 0.76489738]]
```

In [82]:
```python
metrics.accuracy_score(y_test,y_pred)
```

Out[82]:
```
0.6142857142857143
```

Model Evaluation using Test set

In [136…
```python
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 [62]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2019-07-11 02:32:38--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.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

2019-07-11 02:32:38 (638 MB/s) - 'loan_test.csv' saved [3642/3642]
```

Load Test set for evaluation

In [120…
```python
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[120…

	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

In [131…
```python
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)
Feature_test = test_df[['Principal','terms','age','Gender','weekend']]
Feature_test = pd.concat([Feature_test,pd.get_dummies(test_df['education'])], axis=1)
Feature_test.drop(['Master or Above'], axis = 1,inplace=True)
Feature_test.head()
```

Out[131…

	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

In [143…
```python
X_testset=Feature_test
y_testset=pd.get_dummies(test_df['loan_status'])['PAIDOFF'].values
y_testset
```

Out[143…
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=uint8)
```

In [145…
```python
y_pred_knn=knnmodel.predict(X_testset)
y_pred_dt=dtmodel.predict(X_testset)
y_pred_svm=svmmodel.predict(X_testset)
y_pred_lr=lrmodel.predict(X_testset)
y_pred_lr_proba=lrmodel.predict_proba(X_testset)
```

In [151…
```python
print(f1_score(y_testset,y_pred_knn))
print(f1_score(y_testset,y_pred_dt))
print(f1_score(y_testset,y_pred_svm))
print(f1_score(y_testset,y_pred_lr))
```

```
0.851063829787234
0.8157894736842106
0.8602150537634409
0.851063829787234
```

In [152…
```python
print(jaccard_similarity_score(y_testset,y_pred_knn))
print(jaccard_similarity_score(y_testset,y_pred_dt))
print(jaccard_similarity_score(y_testset,y_pred_svm))
print(jaccard_similarity_score(y_testset,y_pred_lr))
```

```
0.7407407407407407
0.7407407407407407
0.7592592592592593
0.7407407407407407
```

In [154…
```python
LR_log_loss=log_loss(y_testset,y_pred_lr_proba)
LR_log_loss
```

Out[154…
```
0.5446922340322017
```

Report

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

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.741	0.851	NA
Decision Tree	0.741	0.816	NA
SVM	0.759	0.860	NA
LogisticRegression	0.741	0.851	0.545