```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

Let's download the dataset

```
!wget -O loan_train.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c
```

```
--2022-08-25 14:17:15--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/FinalModule_Coursera/data/
loan_train.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.
s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-d
ata.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23101 (23K) [text/csv]
Saving to: 'loan_train.csv'

     0K .......... .......... ..                              100% 3,12M=0,007s

2022-08-25 14:17:16 (3,12 MB/s) - 'loan_train.csv' saved [23101/23101]
```

Load Data From CSV File

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

	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

```python
df.shape
```

```
(346, 10)
```

Convert to date time object

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

	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

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

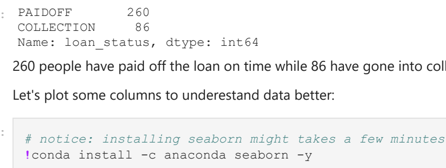
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

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

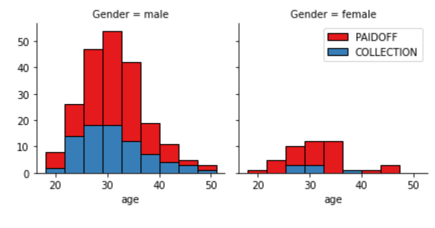
Let's plot some columns to underestand data better:

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

```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()
```



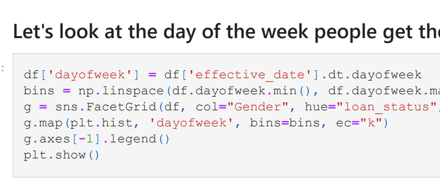

```python
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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



Pre-processing: Feature selection/extraction

Let's look at the day of the week people get the loan

```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 don't pay it off, so let's use Feature binarization to set a threshold value less than day 4

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

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	Gender	day
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

Let's look at gender:

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

```
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

Let's convert male to 0 and female to 1:

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

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	Gender	day
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

How about education?

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

```
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
```

Features before One Hot Encoding

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

	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

```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()
```

	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

Let's define feature sets, X:

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

	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?

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

```
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)

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

```
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

K Nearest Neighbor(KNN)

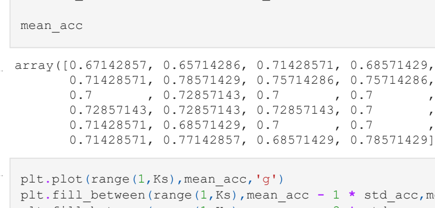
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

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

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

Ks = 30
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

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])

mean_acc
```

```
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       , 0.68571429,
       0.72857143, 0.72857143, 0.72857143, 0.7       , 0.68571429,
       0.71428571, 0.68571429, 0.7       , 0.7       , 0.72857143,
       0.71428571, 0.77142857, 0.68571429, 0.78571429])
```

```python
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=0.1(
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.1(
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```




```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= 7
```

```python
k = 7
neigh7 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat7 = neigh6.predict(X_test)
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh6.predict(X_train))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat6))
```

```
Train set Accuracy:  0.8079710144927537
Test set Accuracy:  0.7857142857142857
```

```python
from sklearn.metrics import log_loss
y_hat_probKNN = neigh7.predict_proba(X_test)
logKNN = log_loss(y_test, y_hat_probKNN)
```

```python
from sklearn.metrics import f1_score
f1KNN = f1_score(y_test, yhat7, average='weighted')
```

```python
from sklearn.metrics import jaccard_score
jacKNN = jaccard_score(y_test, yhat7, pos_label='PAIDOFF')
```

Decision Tree

```python
import sys
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import preprocessing
%matplotlib inline
```

```python
loanTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
loanTree # it shows the default parameters
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```python
loanTree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```python
predTree = loanTree.predict(X_test)
```

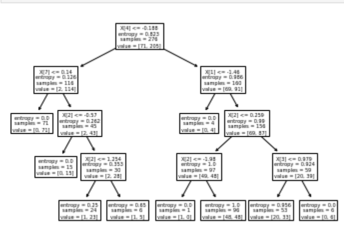
```python
#visual predictions of the values
print (predTree [0:5])
print (y_test [0:5])
```

```
['COLLECTION' 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF']
['PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF']
```

```python
#evaluation
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, predTree))
```

```
DecisionTrees's Accuracy:  0.6142857142857143
```

```python
tree.plot_tree(loanTree)
plt.show()
```



```python
from sklearn.metrics import log_loss
y_hat_probTREE = loanTree.predict_proba(X_test)
logTREE = log_loss(y_test, y_hat_probTREE)
```

```python
from sklearn.metrics import f1_score
f1TREE = f1_score(y_test, predTree, average='weighted')
```

```python
from sklearn.metrics import jaccard_score
jacTREE = jaccard_score(y_test, predTree, pos_label='PAIDOFF')
```

Support Vector Machine

```python
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

```python
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

```
SVC()
```

```python
yhat = clf.predict(X_test)
yhat [0:5]
```

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

```python
#evaluation
from sklearn import metrics
import matplotlib.pyplot as plt
print("SVM Accuracy rbf: ", metrics.accuracy_score(y_test, yhat))
```

```
SVM Accuracy rbf:  0.7428571428571429
```

```python
clf = svm.SVC(kernel='sigmoid')
clf.fit(X_train, y_train)
yhat = clf.predict(X_test)
yhat [0:5]
#evaluation

print("SVM Accuracy sigmoid: ", metrics.accuracy_score(y_test, yhat))
```

```
SVM Accuracy sigmoid:  0.7428571428571429
```

```python
clf = svm.SVC(kernel='poly')
clf.fit(X_train, y_train)
yhat = clf.predict(X_test)
yhat [0:5]
#evaluation

print("SVM Accuracy polynomial: ", metrics.accuracy_score(y_test, yhat))
```

```
SVM Accuracy polynomial:  0.7714285714285715
```

```python
clf = svm.SVC(kernel='linear', probability=True)
clf.fit(X_train, y_train)
yhat = clf.predict(X_test)
yhat [0:5]
#evaluation

print("SVM Accuracy linear: ", metrics.accuracy_score(y_test, yhat))
```

```
SVM Accuracy linear:  0.7857142857142857
```

```python
from sklearn.metrics import f1_score
f1SVM = f1_score(y_test, yhat, average='weighted')
```

```python
from sklearn.metrics import jaccard_score
jacSVM = jaccard_score(y_test, yhat, pos_label='PAIDOFF')
```

```python
from sklearn.metrics import log_loss
y_hat_probSVM = clf.predict_proba(X_test)
logSVM = log_loss(y_test, y_hat_probSVM)
```

Logistic Regression

```python
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
```

```python
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
```

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

```python
y_hat = LR.predict(X_test)
y_hat
```

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

```python
y_hat_prob = LR.predict_proba(X_test)
```

```python
from sklearn.metrics import jaccard_score
jacLOG = jaccard_score(y_test, y_hat,pos_label='PAIDOFF')
```

```python
from sklearn.metrics import f1_score
f1LOG = f1_score(y_test, y_hat, average='weighted')
```

```python
from sklearn.metrics import log_loss
logLOG = log_loss(y_test, y_hat_prob)
```

Model Evaluation using Test set

```python
dt = {'Algorithm': ['KNN', 'Decision Tree', 'SVM', 'LogisticRegression'],
        'Jaccard': [jacKNN, jacTREE, jacSVM, jacLOG],
      'F1-score': [f1KNN, f1TREE, f1SVM, f1LOG],
      'LOGloss': [logKNN, logTREE, logSVM, logLOG]}
df_results = pd.DataFrame(dt)
df_results
```

	Algorithm	Jaccard	F1-score	LOGloss
0	KNN	0.765625	0.776654	0.467195
1	Decision Tree	0.571429	0.644599	1.375281
2	SVM	0.785714	0.691429	0.523961
3	LogisticRegression	0.676471	0.667052	0.577229