Using Python to to determine who should receive a loan and who does not qualify.

September 24, 2017

1 Assignment_1

```
In [1]: import pandas as pd
            import numpy as np import
            matplotlib as plt
```

```
In [2]: # Read the Load Predictor CSV file of your dataset into a dataframe object

            dataset_df = pd.read_csv("c:\\datasets\insurancedataset.csv")

            # sanity test the content of the dataframe object

            dataset_df.head()
```

```
Out[2]: Loan_ID Gender Married Dependents Education Self_Employed \
            0 LP001002 Male Yes 0 Graduate No 1 LP001003 Male No 1 Graduate No 2
            LP001005 Male Yes 0 Graduate Yes 3 LP001006 Male Yes 0 Not Graduate No 4
            LP001008 Male No 0 Graduate No

      ApplicantIncome CoapplicantIncome LoanAmount Loan_Amount_Term \ 0 5849 0 140 360 1
            4583 1508 128 360 2 3000 0 66 360 3 2583 2358 120 360 4 6000 0 141 360

      Credit_History Property_Area Loan_Status 0 1 Urban Y 1 1
            Rural N 2 1 Urban Y 3 1 Urban Y 4 1 Urban Y
```

```
In [3]: # General stat description of the data in the dataframe

            dataset_df.describe()
```

1

Out[3]: Dependents ApplicantIncome CoapplicantIncome LoanAmount \

```
count 50.000000 50.000000 50.000000 50.000000 mean 1.040000 4177.860000
1948.320000 138.780000 std 0.879703 2487.662119 2322.676179 77.780248 min
0.000000 1299.000000 0.000000 16.000000 25% 0.000000 2600.000000 0.000000
101.750000 50% 1.000000 3516.500000 1521.000000 117.500000 75% 2.000000
5025.000000 2840.000000 157.000000 max 2.000000 12841.000000 10968.000000
349.000000
```

```
Loan_Amount_Term      Credit_History      count
 50.000000 50.000000 mean 340.800000 0.880000
 std 61.140187 0.328261 min 120.000000 0.000000
 25%   360.000000   1.000000   50%   360.000000
 1.000000    75%    360.000000    1.000000    max
 360.000000 1.000000
```

In [4]:
```
# Count how many applicants from every property area

dataset_df['Property_Area'].value_counts()
```

Out[4]: Urban 34 Semiurban 8 Rural 8 Name: Property_Area, dtype: int64

2 Visualize the dataset

2.1 Create histograms for significant features

In [5]:
```
%matplotlib inline
dataset_df['ApplicantIncome'].hist(color='DarkGreen', bins=100)
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x8dbb438> 2 In [6]:
dataset_df['LoanAmount'].hist(color='DarkOrange',bins=100)

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x84b5860> 3

2.2 Create stacked-bar chart for some features

In [7]:
```
stacked_bar_chart = pd.crosstab(dataset_df['Dependents'], dataset_df['Loan_Status'])
stacked_bar_chart.plot(kind='bar', stacked=True, color=['red','green'], grid=False)
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x926d1d0> In [8]:

```
stacked_bar_chart

Out[8]: Loan_Status N Y
Dependents 0 7 11 1 8 4 2 3 17

In [9]: stacked_bar_chart = pd.crosstab(dataset_df['Gender'], dataset_df['Loan_Status'])
               stacked_bar_chart.plot(kind='bar', stacked=True, color=['red','green'], grid=False)

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x996e978>In [10]:
stacked_bar_chart = pd.crosstab(dataset_df['Married'],
dataset_df['Loan_Status'])
                 stacked_bar_chart.plot(kind='bar', stacked=True, color=['red','green'], grid=False)

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x9b18e80>
```

2.3 Create Boxplot for significant features

```
In [11]: dataset_df.boxplot(column='LoanAmount')

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0xa0a4588>



In [12]: dataset_df['Self_Employed'].value_counts()

Out[12]: No 44 Yes 6 Name: Self_Employed, dtype:
int64
```

2.4 Preprocessng and Normalization

```
In [13]: # Lets get the total income and add it to dataset dataframe
                 dataset_df['TotalIncome'] = dataset_df['ApplicantIncome'] + dataset_df['CoapplicantIncome']

In [14]: # Boxplot the total income
                 dataset_df.boxplot(column='TotalIncome')

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0xa1295f8> In [15]: #
What are the data types for the different features and label


                 dataset_df.dtypes

Out[15]:   Loan_ID   object   Gender   object
Married object
```

```
Dependents int64 Education object
Self_Employed                object
ApplicantIncome               int64
CoapplicantIncome             int64
LoanAmount                    int64
Loan_Amount_Term              int64
Credit_History                int64
Property_Area object Loan_Status
object TotalIncome int64 dtype:
object
```

In [16]:
```
# Sanity test the head of the dataframe object

dataset_df.head()
```

Out[16]:
```
Loan_ID Gender Married Dependents Education Self_Employed \
0 LP001002 Male Yes 0 Graduate No 1 LP001003 Male No 1 Graduate No 2
LP001005 Male Yes 0 Graduate Yes 3 LP001006 Male Yes 0 Not Graduate No 4
LP001008 Male No 0 Graduate No

ApplicantIncome CoapplicantIncome LoanAmount Loan_Amount_Term \ 0 5849 0 140 360 1
4583 1508 128 360 2 3000 0 66 360 3 2583 2358 120 360 4 6000 0 141 360

Credit_History Property_Area Loan_Status TotalIncome 0 1 Urban Y 5849 1 1
Rural N 6091 2 1 Urban Y 3000 3 1 Urban Y 4941 4 1 Urban Y 6000
```

In [17]:
```
# Sanity test the head of the dataframe object

dataset_df.tail()
```

Out[17]:
```
Loan_ID Gender Married Dependents Education Self_Employed \
45 LP001112 Male Yes 2 Graduate No 46 LP001113 Male Yes 1 Graduate No 47
LP001114 Male No 0 Graduate No 48 LP001115 Male Yes 2 Graduate No 49
LP001116 Male Yes 2 Graduate Yes
```

```
ApplicantIncome CoapplicantIncome LoanAmount Loan_Amount_Term \ 45 3995 1600 159 360
46 9200 0 338 360 47 1785 2840 116 360 48 1320 1086 18 120 49 5375 3970 259 360
```

```
Credit_History Property_Area Loan_Status TotalIncome 45 1 Urban Y 5595 46
    1 Semiurban N 9200 47 1 Rural N 4625 48 1 Urban Y 2406 49 1 Urban Y
    9345
```

In [19]:
```
# Identift the features and the label to be included in th emodel construction


    # 4 features considered features_considered =
    ['Credit_History','Gender','Married','Education']

    target_label = 'Loan_Status'
```

In [20]:
```
from sklearn.preprocessing import LabelEncoder

    list_of_feaatures_to_encode = ['Gender','Married','Education','Self_Employed','Property_Area','  le
    LabelEncoder()

    for i in list_of_feaatures_to_encode:
        enc = le.fit(np.unique(dataset_df[i].values))
        print(enc.classes_) dataset_df[i] =
        le.fit_transform(dataset_df[i])


    features_df = dataset_df[features_considered] target_df =
    dataset_df[target_label]
```

```
['Female'  'Male'] ['No'
'Yes'] ['Graduate'  'Not
Graduate'] ['No'  'Yes']
['Rural'  'Semiurban'  'Urban']
['N'  'Y']
```

In [21]:
```
# What are the types of the features after label encoding transformation

    dataset_df.dtypes
```

Out[21]:
```
Loan_ID object Gender int64
```

```
    Married   int64   Dependents   int64
    Education    int64    Self_Employed
```

```
int64     ApplicantIncome     int64
CoapplicantIncome             int64
LoanAmount                    int64
Loan_Amount_Term              int64
Credit_History                int64
Property_Area  int64  Loan_Status
int64  TotalIncome  int64  dtype:
object
```

```
In [22]: # sanity test

         dataset_df[['Credit_History', 'Gender', 'Married', 'Education', 'Loan_Status']].head()
```

```
Out[22]: Credit_History Gender Married Education Loan_Status 0 1 1 1 0 1 1 1 1 0 0 0
2 1 1 1 0 1 3 1 1 1 1 1 4 1 1 0 0 1
```

```
In [23]: features_df.head(2)
```

```
Out[23]: Credit_History Gender Married Education 0 1 1 1 0 1 1 1 0 0
```

```
In [24]: target_df.head(2)
```

```
Out[24]: 0 1 1 0 Name: Loan_Status, dtype: int64
```

```
In [25]: #convert dataframe to ndarray
         features = features_df.values target =
         target_df.values
```

```
In [26]: type(features)
```

```
Out[26]: numpy.ndarray
```

```
In [27]: features[0]
```

```
Out[27]: array([1, 1, 1, 0], dtype=int64)
```

```
In [28]: target[0]
```

1

0

```
Out[28]: 1
```

```
In [29]: # Build the prective model
```

```
          # default number of n_neighbors=5

          from sklearn.neighbors import KNeighborsClassifier

          # fit a k-nearest neighbor model to the data model =
          KNeighborsClassifier() model.fit(features, target)

          # print the default parameter settings for the model print(model)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

```
     In [30]: # predict the label of an observation:
                model.predict([[1, 1, 0, 1]])

Out[30]: array([1], dtype=int64)

     In [31]: # predict the label of an observation:
                model.predict([[0, 0, 1, 0]])

Out[31]: array([0], dtype=int64)

     In [32]: # predict the label of an observation:
                model.predict([[0, 1, 1, 0]])

Out[32]: array([1], dtype=int64)

     In [33]: # predict the label of an observation:
                model.predict([[0, 0, 1, 1]])

Out[33]: array([0], dtype=int64)
```

```
In [34]: # Fine-tune the predtive model
          # Lets change the n_neighbors to 10


          model = KNeighborsClassifier(n_neighbors=10)
          model.fit(features, target) print(model)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=10, p=2, weights='uniform')
```

```
1
1
```

```
In [35]: # predict the label of an observation:
         model.predict([[1, 1, 0, 1]])
```

```
Out[35]: array([1], dtype=int64)
```

```
In [36]: # predict the label of an observation:
         model.predict([[0, 0, 1, 0]])
```

```
Out[36]: array([1], dtype=int64)
```

```
In [37]: # predict the label of an observation:
         model.predict([[0, 1, 1, 0]])
```

```
Out[37]: array([1], dtype=int64)
```

```
In [38]: # predict the label of an observation:
         model.predict([[0, 0, 1, 1]])
```

```
Out[38]: array([0], dtype=int64)
```

2.5 Requirement #1: Create histograms for the CoApplicantIncome feature

```
In [39]: # Add your code for Requiremen #1 in this cell
         %matplotlib inline dataset_df['CoapplicantIncome'].hist(color='DarkGreen',
         bins=100)

         %matplotlib inline dataset_df['CoapplicantIncome'].hist(color='Red',
         bins=50)
```

```
Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0xe37d048>
```

2.6 Requirement #2: Create a stackchart for Credit_History and Loan_Status features

```
In [40]: # Add your code for Requiremen #2 in this cell
         stacked_bar_chart = pd.crosstab(dataset_df['Credit_History'], dataset_df['Loan_Status'])
         stacked_bar_chart.plot(kind='bar', stacked=True, color=['red','green'], grid=False)
```

```
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0xe4ffd68>
```

2.7 Requirement #3: Create boxplot chart for ApplicantIncome feature

In [41]:

```
# Add your code for Requiremen #3 in this cell

dataset_df.boxplot(column='ApplicantIncome')
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdff4668>
```

2.8 Requirement #4: Create KNeighborsClassifier predictive model considering the following features:

- ['Credit_History', 'Education', 'Married', 'Self_Employed', 'Property_Area']

In [61]: *# Add your code for Requiremen #4 in this cell*

```
features_considered1 = ['Credit_History', 'Education', 'Married', 'Self_Employed', 'Property_Ar

target_label1 = 'Loan_Status'


list_of_feaatures_to_encode = ['Credit_History','Married','Education','Self_Employed','Property le =
LabelEncoder()

for i in list_of_feaatures_to_encode:
    enc = le.fit(np.unique(dataset_df[i].values))
    print(enc.classes_) dataset_df[i] =
    le.fit_transform(dataset_df[i])


features_df = dataset_df[features_considered1] target_df =
dataset_df[target_label1]

features = features_df.values target1 =
target_df.values

model1 = KNeighborsClassifier(n_neighbors=3)
model1.fit(features, target) print(model1)
```

```
[0 1] [0 1] [0 1] [0 1] [0 1 2] [0 1] KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
    metric_params=None, n_jobs=1, n_neighbors=3, p=2,
    weights='uniform')
```

2.9 Requirement #5: Run your model with n_neighbors equals 3 for the obersevations testing_data.csv :

In [62]:
```
testing_dataset_df = pd.read_csv("c:\\datasets\Hmw1FDSTestDataset.csv")
testing_dataset_df
```

Out[62]:
```
Loan_ID Gender Married Dependents Education Self_Employed \
    0 LP001054 Male Yes 0 Not Graduate Yes 1 LP001055 Female No 1 Not Graduate No
    2 LP001056 Male Yes 2 Not Graduate No 3 LP001059 Male Yes 2 Graduate Yes 4
    LP001067 Male No 0 Not Graduate No

ApplicantIncome CoapplicantIncome LoanAmount Loan_Amount_Term \ 0 2165 3422 152 360 1
    2226 0 59 360 2 3881 0 147 360 3 13633 0 280 240 4 2400 2400 123 360

Credit_History Property_Area 0 1 Urban 1 1
    Semiurban 2 0 Rural 3 1 Urban 4 1
    Semiurban
```

In [63]:
```
testing_dataset_df.dtypes
```

Out[63]:
```
Loan_ID object Gender object
Married object Dependents int64
    Education object Self_Employed
    object    ApplicantIncome    int64
    CoapplicantIncome            int64
    LoanAmount                   int64
    Loan_Amount_Term             int64
    Credit_History               int64
```

```
            Property_Area object dtype: object
```

```
In [64]: features_considered2 = ['Credit_History', 'Education', 'Married', 'Self_Employed', 'Property_Ar
```

```
In [65]: from sklearn.preprocessing import LabelEncoder

            list_of_feaatures_to_encode = ['Credit_History','Married','Education','Self_Employed','Property le =
            LabelEncoder()

            for i in list_of_feaatures_to_encode:
                enc = le.fit(np.unique(testing_dataset_df[i].values)) print(enc.classes_)
                testing_dataset_df[i] = le.fit_transform(testing_dataset_df[i])


            features_df = testing_dataset_df[features_considered2] features2 =
            features_df.values
```

```
[0 1] ['No'  'Yes'] ['Graduate'
'Not Graduate'] ['No'  'Yes']
['Rural'  'Semiurban'  'Urban']
```

```
In [67]: pred=model1.predict(features2)
            print(pred)
```

```
[1 1 0 1 1]
```