Bank_Loans_Modelling

September 3, 2020

(1)Importing Library and Load data

```
In [3]: #importing required library
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import pandas.util.testing as tm
```

```
In [15]: #upload dataset from my local machine
         from google.colab import files
         uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Bank_Personal_Loan_Modelling.xlsx to Bank_Personal_Loan_Modelling (1).xlsx
```

```
In [4]: #read excel file and store into a dataframe
        df = pd.read_excel( 'Bank_Personal_Loan_Modelling.xlsx' , 'Data' )
```

```
In [5]: #used for calculating some statistical data like percentile, mean and std of the numer
        df.describe()
```

```
Out[5]:                 ID          Age  ...       Online   CreditCard
        count  5000.000000  5000.000000  ...  5000.000000  5000.000000
        mean   2500.500000    45.338400  ...     0.596800     0.294000
        std    1443.520003    11.463166  ...     0.490589     0.455637
        min       1.000000    23.000000  ...     0.000000     0.000000
        25%    1250.750000    35.000000  ...     0.000000     0.000000
        50%    2500.500000    45.000000  ...     1.000000     0.000000
        75%    3750.250000    55.000000  ...     1.000000     1.000000
        max    5000.000000    67.000000  ...     1.000000     1.000000

        [8 rows x 14 columns]
```

```
In [6]: #show the starting five rows of the datasets
        df.head()
```

```
Out[6]:    ID  Age  Experience  ...  CD Account  Online  CreditCard
        0   1   25           1  ...           0       0           0
        1   2   45          19  ...           0       0           0
        2   3   39          15  ...           0       0           0
        3   4   35           9  ...           0       0           0
        4   5   35           8  ...           0       0           1

        [5 rows x 14 columns]
```

```
In [7]: #show the last five rows of the datasets
        df.tail()
```

```
Out[7]:         ID  Age  Experience  ...  CD Account  Online  CreditCard
        4995  4996   29           3  ...           0       1           0
        4996  4997   30           4  ...           0       1           0
        4997  4998   63          39  ...           0       0           0
        4998  4999   65          40  ...           0       1           0
        4999  5000   28           4  ...           0       1           1

        [5 rows x 14 columns]
```

```
In [8]: #show the non-null,count and datatypes
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  5000 non-null   int64
 1   Age                 5000 non-null   int64
 2   Experience          5000 non-null   int64
 3   Income              5000 non-null   int64
 4   ZIP Code            5000 non-null   int64
 5   Family              5000 non-null   int64
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64
 8   Mortgage            5000 non-null   int64
 9   Personal Loan       5000 non-null   int64
 10  Securities Account  5000 non-null   int64
 11  CD Account          5000 non-null   int64
 12  Online              5000 non-null   int64
 13  CreditCard          5000 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

```
In [9]: #show the shape of dataset
        df.shape
```

Out[9]: (5000, 14)

In [10]:
```
#this used to get an int representing the number of elements in this object
df.size
```

Out[10]: 70000

In [19]:
```
#check if their is null value or not and we found that Experience column contain null
df.isnull().sum().sort_values(ascending = False)
df = df.dropna(subset=['Experience'])

df.isnull().sum().sort_values(ascending = False)
```

Out[19]:
```
CreditCard            0
Online                0
CD Account            0
Securities Account    0
Personal Loan         0
Mortgage              0
Education             0
CCAvg                 0
Family                0
ZIP Code              0
Income                0
Experience            0
Age                   0
ID                    0
dtype: int64
```

In [21]:
```
#tells us about datatypes
df.dtypes
```

Out[21]:
```
ID                      int64
Age                     int64
Experience              int64
Income                  int64
ZIP Code                int64
Family                  int64
CCAvg                 float64
Education               int64
Mortgage                int64
Personal Loan           int64
Securities Account      int64
CD Account              int64
Online                  int64
CreditCard              int64
dtype: object
```

In [22]:
```
df.apply(lambda x : sum(x.isnull()))
```

Out[22]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

(2)Check if you need to clean the data for any of the variables

In [24]:
```
#check that how many negative value is their for Expeience column , because Expeience
df[df['Experience'] < 0]['Experience'].value_counts()
```

Out[24]:
```
-1    33
-2    15
-3     4
Name: Experience, dtype: int64
```

In [25]:
```
#clean the negative variable
dfExp = df.loc[df['Experience'] >0]
negExp = df.Experience < 0
newlist = df.loc[negExp]['ID'].tolist() # getting the customer ID who has negative ex
negExp.value_counts()
```

Out[25]:
```
False    4948
True       52
Name: Experience, dtype: int64
```

In [26]:
```
import numpy as np
```

In [27]:
```
#Get the value of Age and Education columns
#Filter the records which has positive experience and take the median
#Apply the median to the location which had negative experience
```

In [28]:
```
for id in newlist:
    age = df.loc[np.where(df['ID']==id)]["Age"].tolist()[0]
    df_filtered = dfExp[(dfExp.Age == age)]
    exp = df_filtered['Experience'].median()
    df.loc[df.loc[np.where(df['ID']==id)].index, 'Experience'] = exp
```

In [29]:
```
df[df['Experience'] < 0]['Experience'].count()
```

```
Out[29]: 0
```

```
In [30]: df.describe()
```

```
Out[30]:                 ID          Age  ...       Online   CreditCard
         count  5000.000000  5000.000000  ...  5000.000000  5000.000000
         mean   2500.500000    45.338400  ...     0.596800     0.294000
         std    1443.520003    11.463166  ...     0.490589     0.455637
         min       1.000000    23.000000  ...     0.000000     0.000000
         25%    1250.750000    35.000000  ...     0.000000     0.000000
         50%    2500.500000    45.000000  ...     1.000000     0.000000
         75%    3750.250000    55.000000  ...     1.000000     1.000000
         max    5000.000000    67.000000  ...     1.000000     1.000000

         [8 rows x 14 columns]
```

(3) Study the data distribution in each attribute and target variable

```
In [31]: import seaborn as sns

         df.nunique() #Number of unique in each column
```

```
Out[31]: ID                    5000
         Age                     45
         Experience              44
         Income                 162
         ZIP Code               467
         Family                   4
         CCAvg                  108
         Education                3
         Mortgage               347
         Personal Loan            2
         Securities Account       2
         CD Account               2
         Online                   2
         CreditCard               2
         dtype: int64
```

```
In [32]: zm = df[df['Mortgage'] == 0]['Mortgage'].count() #Number of people with zero mortgage
```

```
In [33]: #Number of people with zero credit card spending per month
         df[df['CCAvg'] == 0]['CCAvg'].count()
```

```
Out[33]: 106
```

```
In [34]: df.count() #Value counts of all categorical columns
```

```
Out[34]: ID                    5000
         Age                   5000
```

```
Experience           4971
Income               5000
ZIP Code             5000
Family               5000
CCAvg                5000
Education            5000
Mortgage             5000
Personal Loan        5000
Securities Account   5000
CD Account           5000
Online               5000
CreditCard           5000
dtype: int64
```

In [35]:
```python
df.Family.value_counts()
```

Out[35]:
```
1    1472
2    1296
4    1222
3    1010
Name: Family, dtype: int64
```

In [36]:
```python
df.Education.value_counts()
```

Out[36]:
```
1    2096
3    1501
2    1403
Name: Education, dtype: int64
```

In [37]:
```python
l=df.Online.value_counts()
```

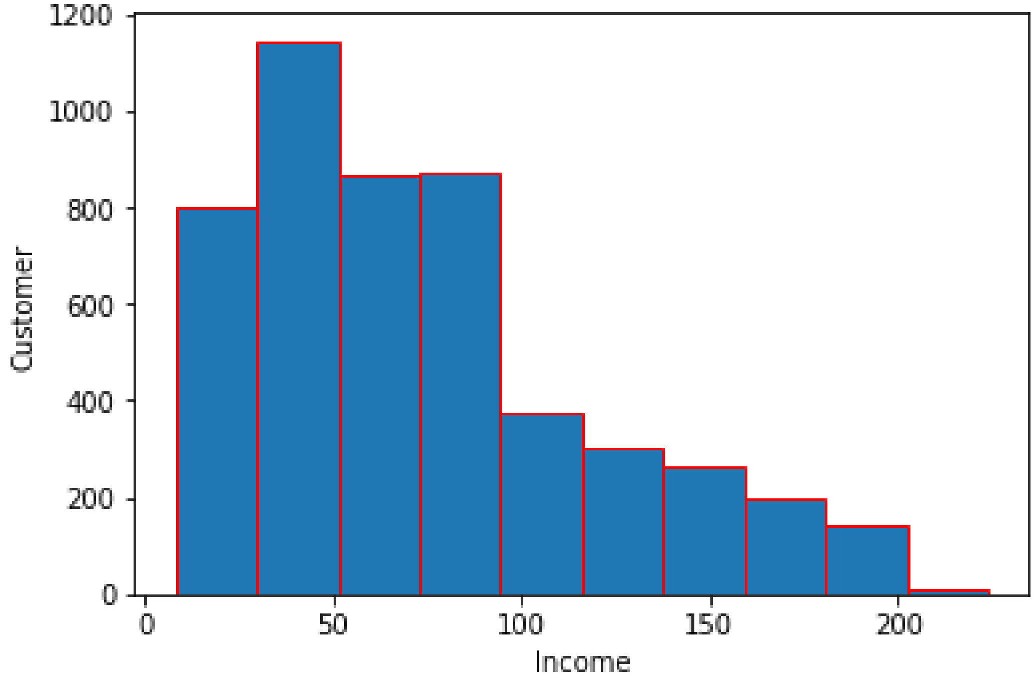
In [38]:
```python
df.CreditCard.value_counts()
```

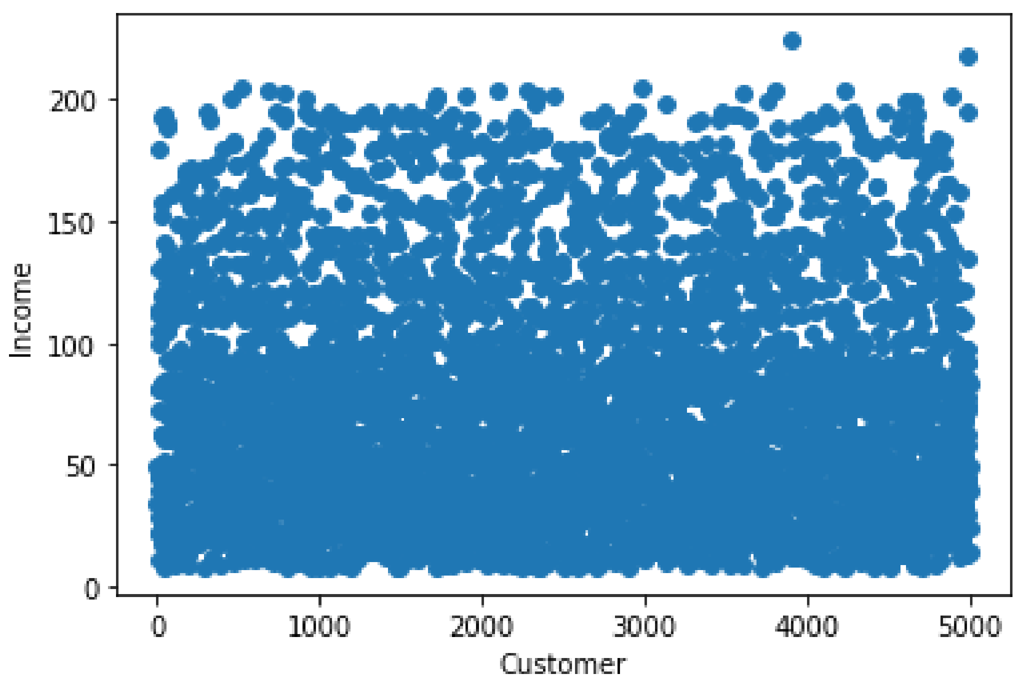
Out[38]:
```
0    3530
1    1470
Name: CreditCard, dtype: int64
```

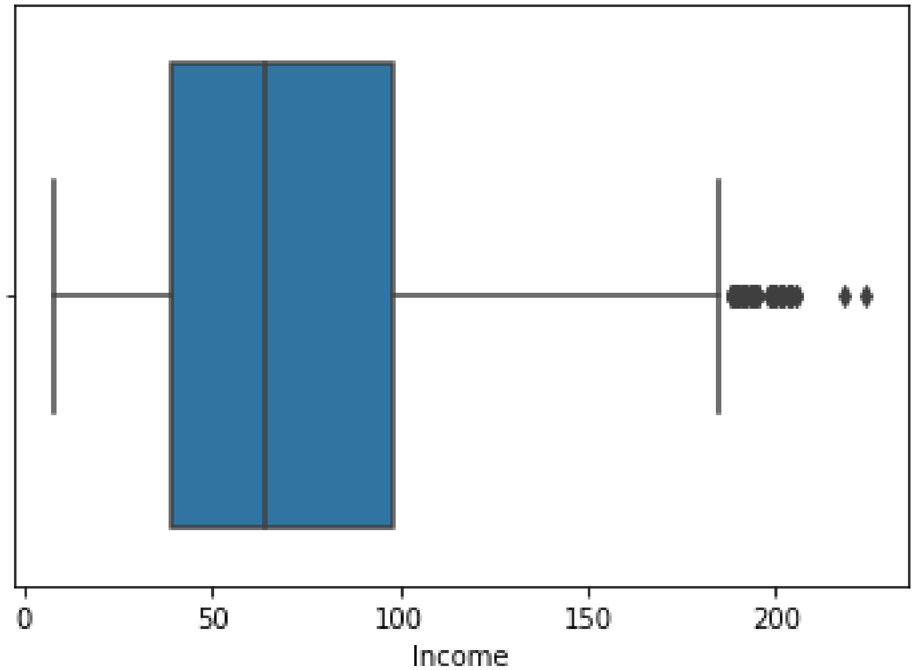
In [39]:
```python
#Univariate analysis
```

In [40]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

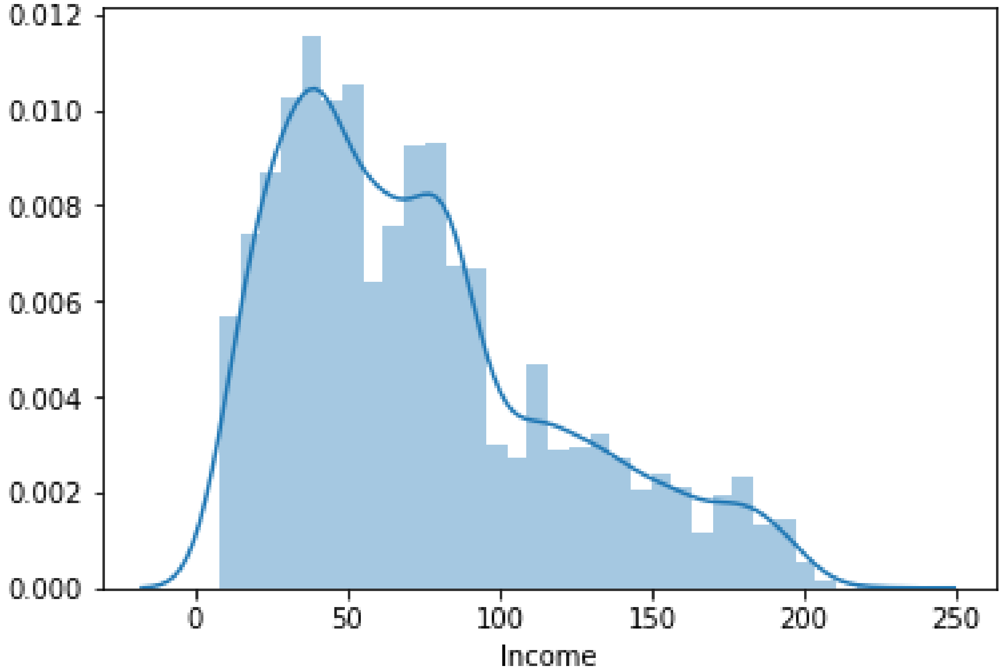
In [154]:
```python
plt.hist(df.Income, edgecolor='Red')
plt.xlabel('Income')
plt.ylabel('Customer')
plt.show()
plt.scatter(df.index,df['Income'])
plt.xlabel('Customer')
plt.ylabel('Income')
plt.show()
sns.boxplot(df['Income'])
plt.show()
```

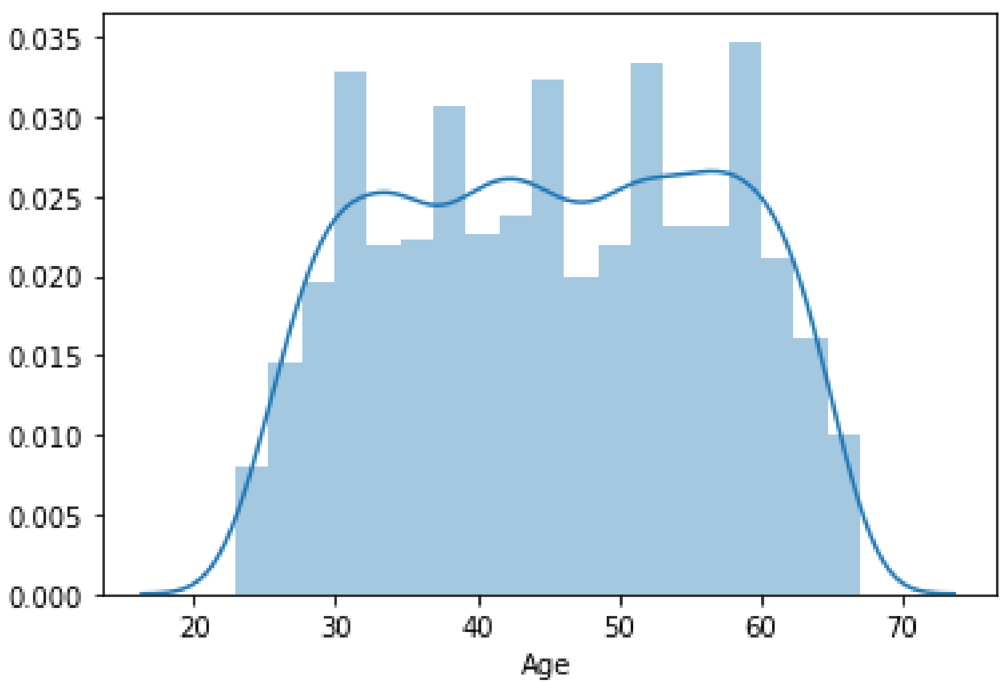

1200
1000
800
600
400
200
0
Customer
0
50
100
150
200
Income

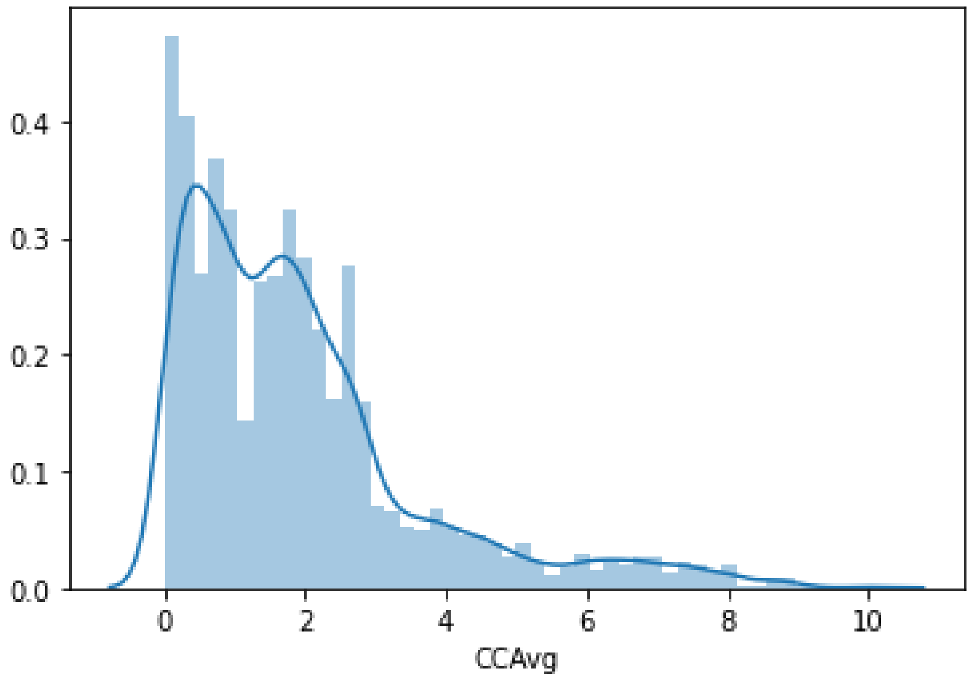

200
150
100
50
0
Income
0
1000
2000
3000
4000
5000
Customer



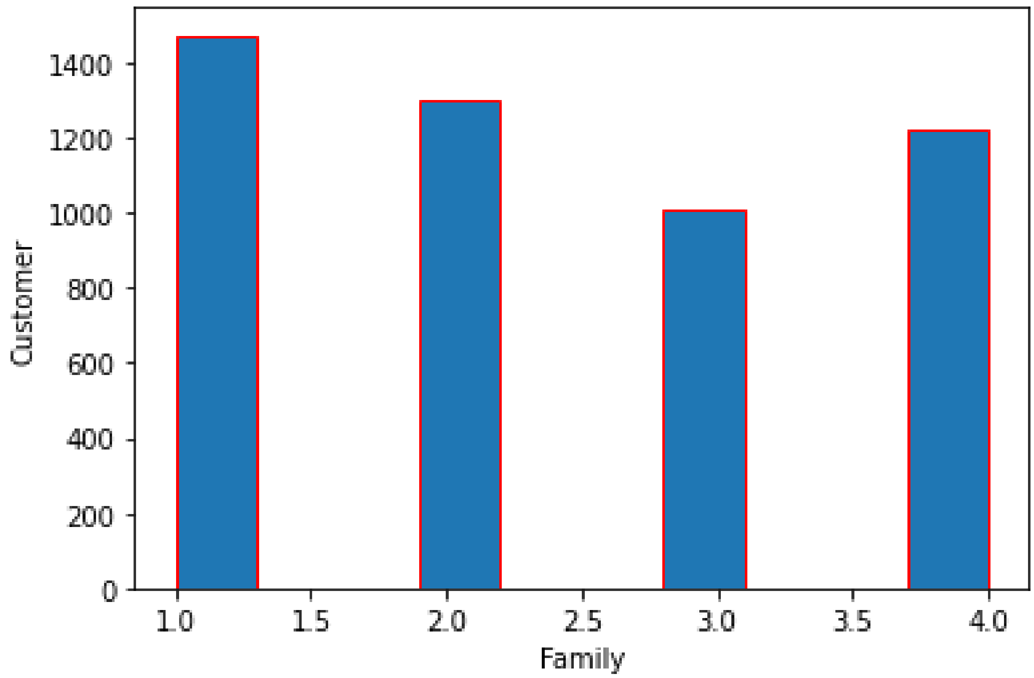

```
In [42]: sns.distplot(df.Income)
         plt.show()
         sns.distplot(df.Age)
         plt.show()
         sns.distplot(df.CCAvg)
         plt.show()
```

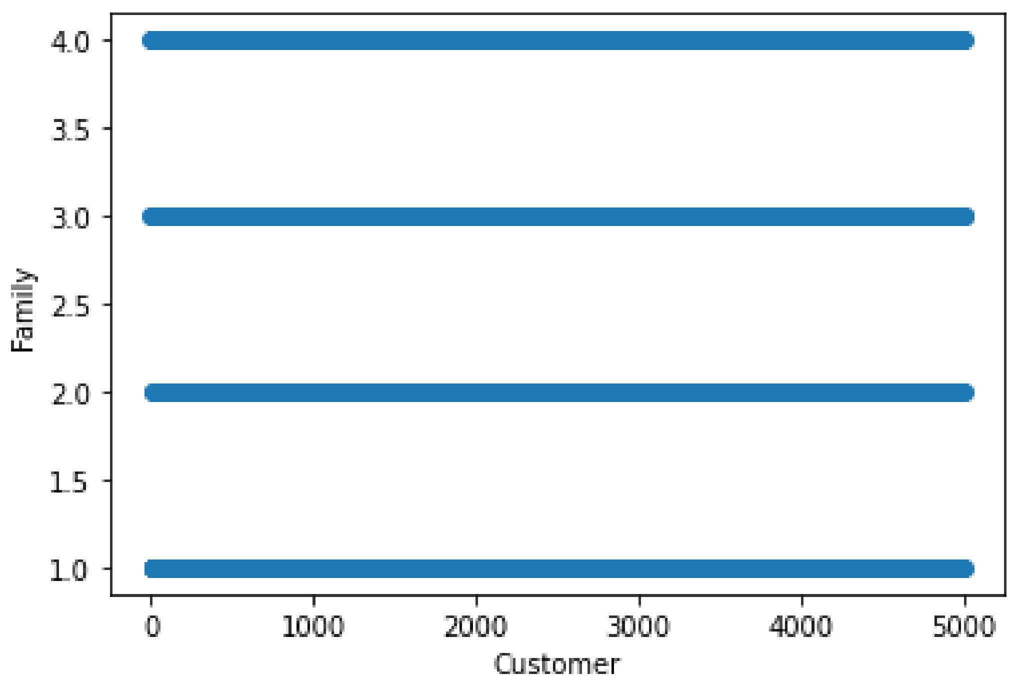

0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
50
100
150
200
250
Income

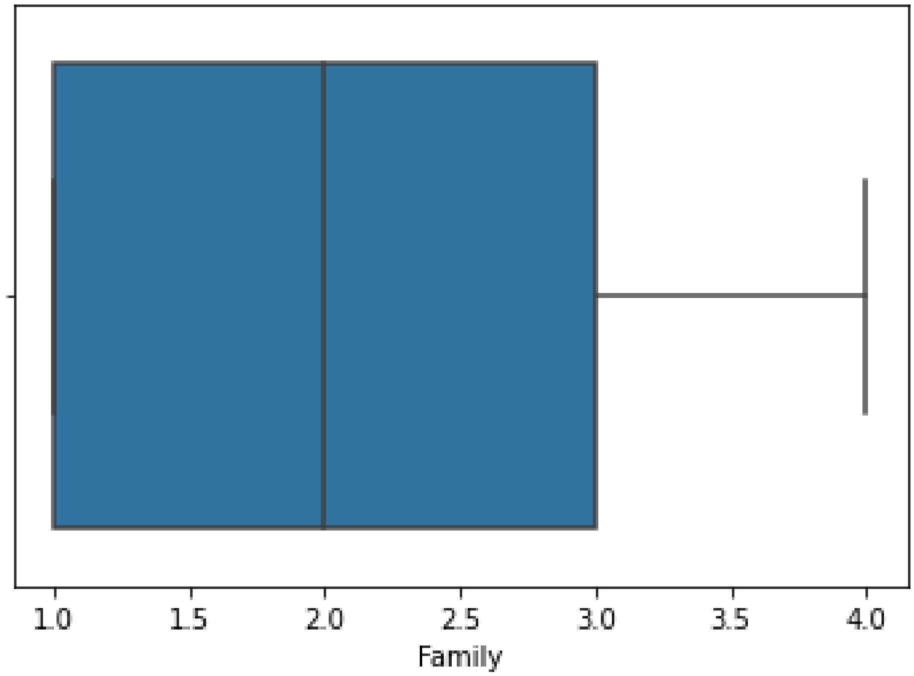

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
20
30
40
50
60
70
Age



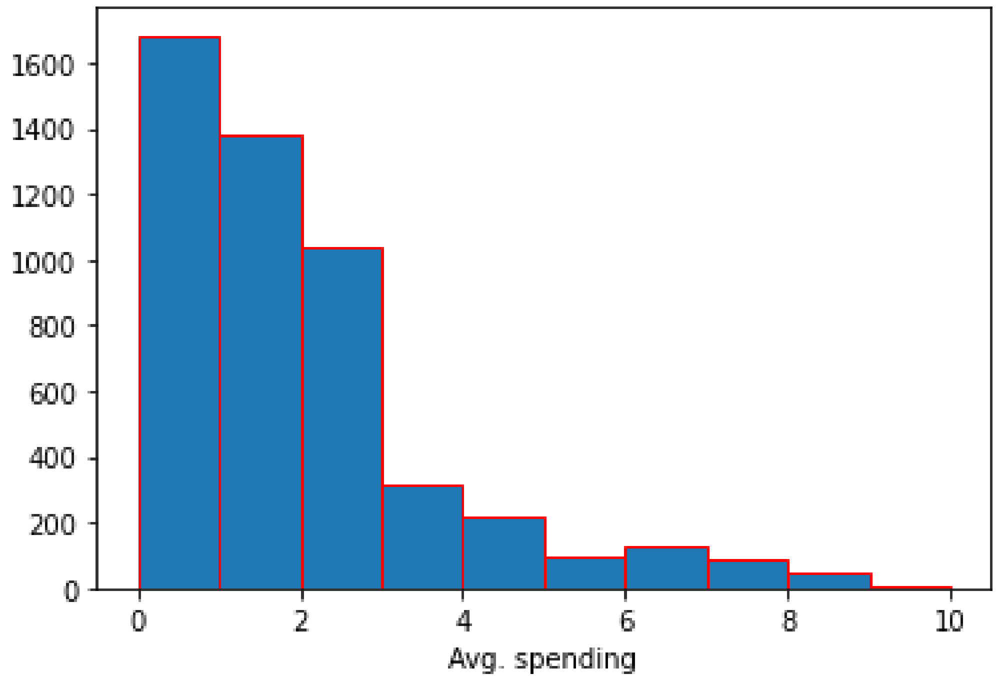

```
In [47]: plt.hist(df.Family, edgecolor='Red')
         plt.xlabel('Family')
         plt.ylabel('Customer')
         plt.show()
         plt.scatter(df.index,df['Family'])
         plt.xlabel('Customer')
         plt.ylabel('Family')
         plt.show()
         sns.boxplot(df['Family'])
         plt.show()
```

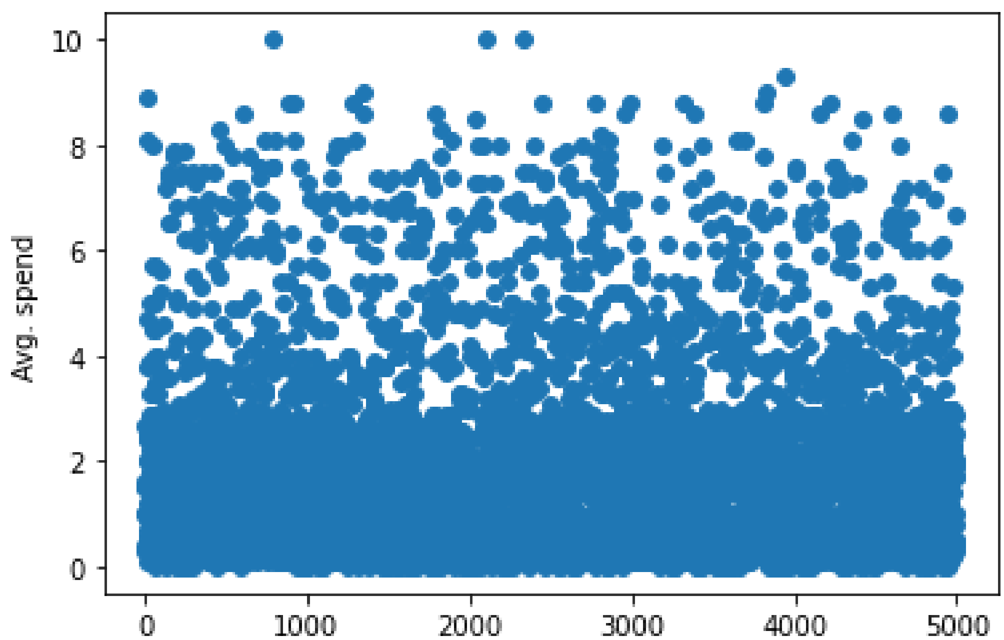

1400
1200
1000
800
600
400
200
0
Customer
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Family


4.0
3.5
3.0
2.5
2.0
1.5
1.0
Family
0
1000
2000
3000
4000
5000
Customer



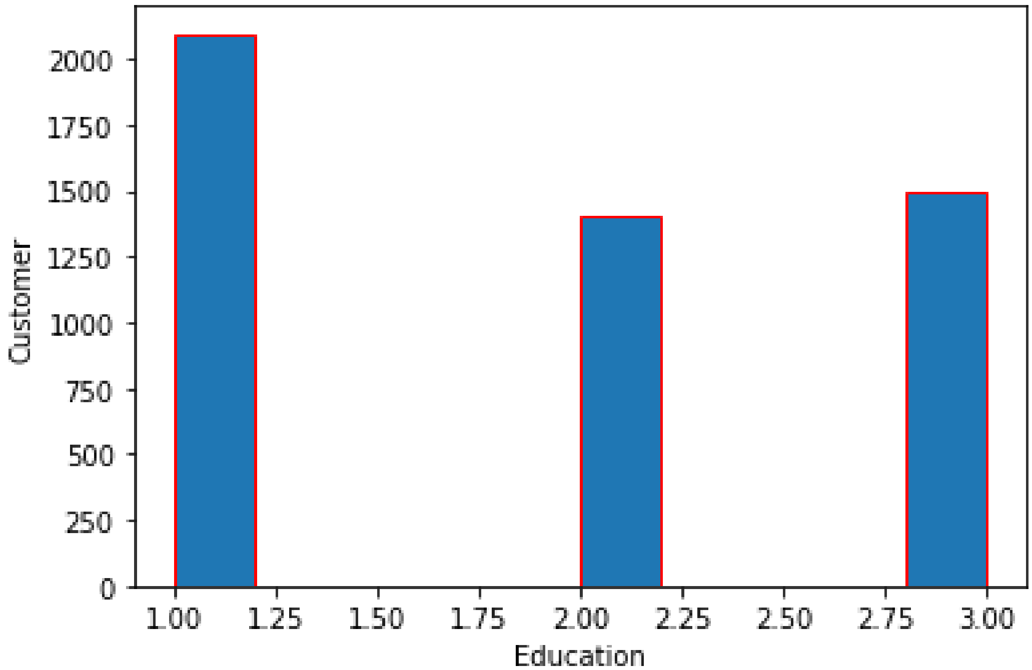

```
In [48]: plt.hist(df['CCAvg'] , edgecolor='Red')
         plt.xlabel('Avg. spending')
         plt.show()
         plt.scatter(df.index,df['CCAvg'])
         plt.ylabel('Avg. spend')
         plt.show()
```

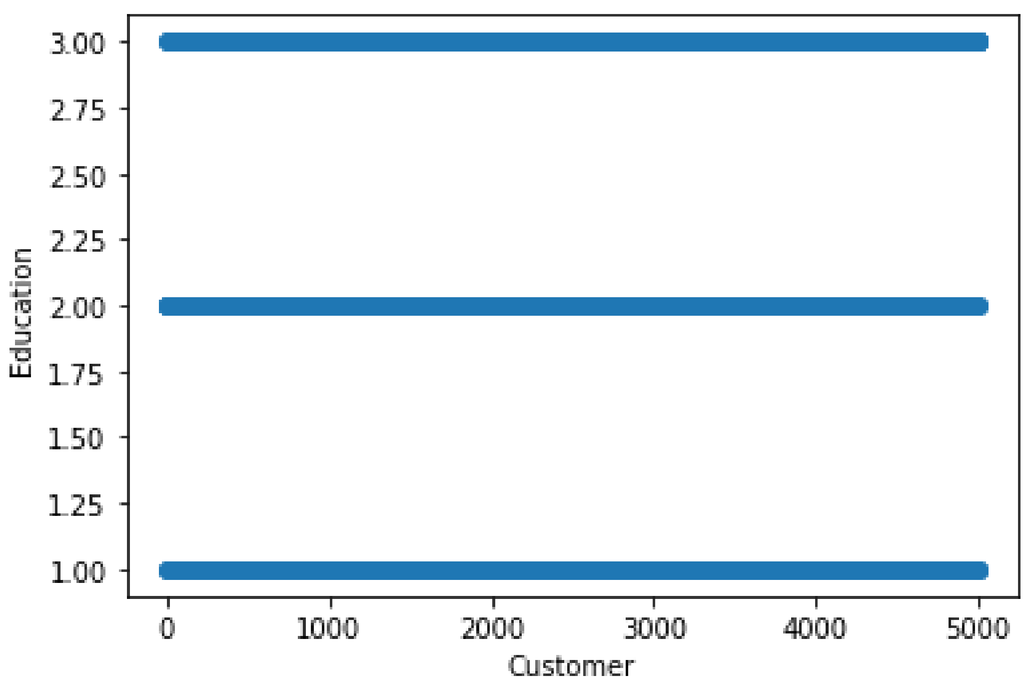




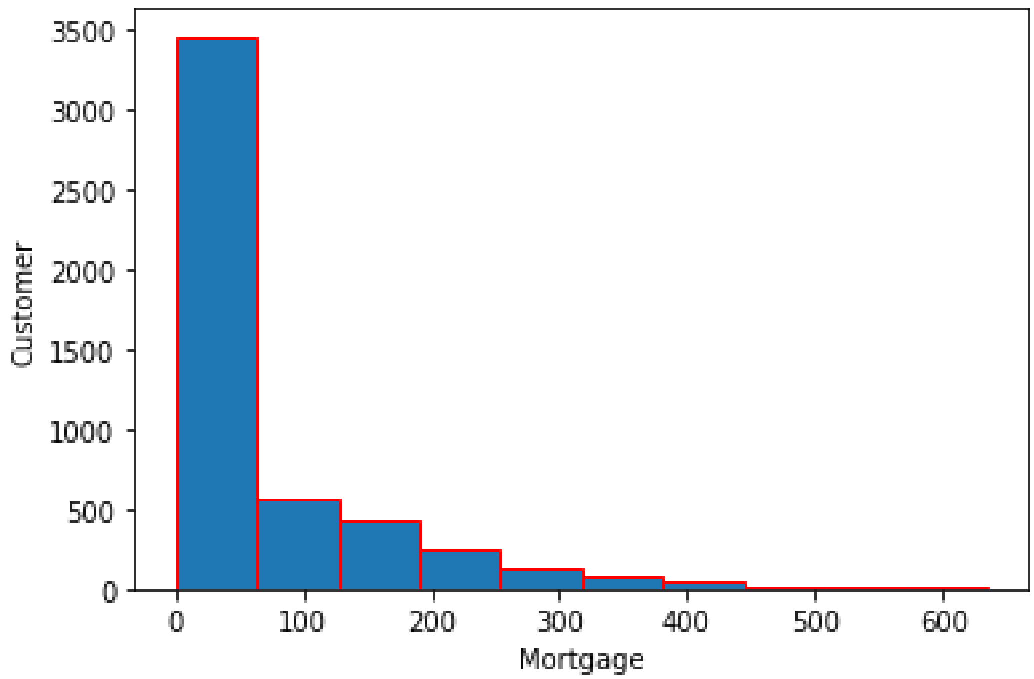

```
In [49]: plt.hist(df.Education, edgecolor='Red')
         plt.xlabel('Education')
         plt.ylabel('Customer')
```

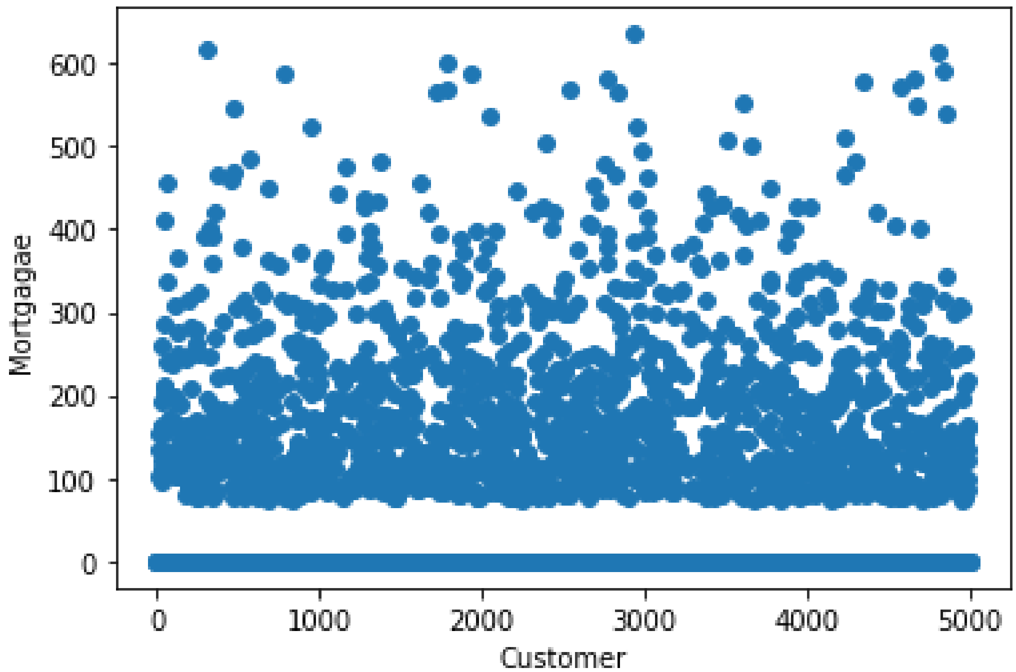
```
plt.show()
plt.scatter(df.index,df['Education'])
plt.xlabel('Customer')
plt.ylabel('Education')
plt.show()
```

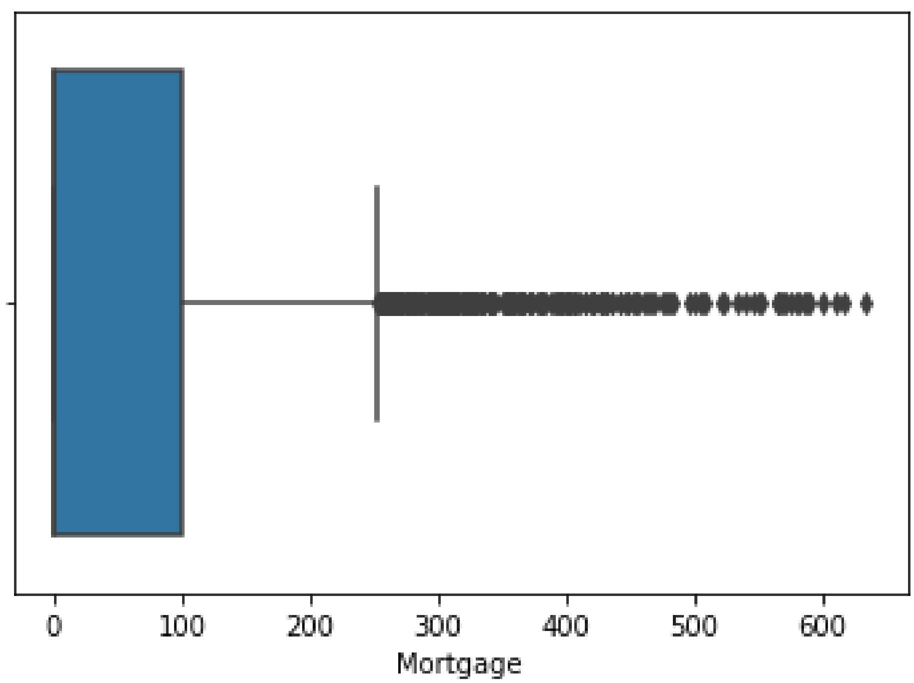




```
In [50]: plt.hist(df.Mortgage, edgecolor='Red')
         plt.xlabel('Mortgage')
         plt.ylabel('Customer')
         plt.show()
         plt.scatter(df.index,df['Mortgage'])
         plt.xlabel('Customer')
         plt.ylabel('Mortgagae')
         plt.show()
         sns.boxplot(df['Mortgage'])
         plt.show()
```

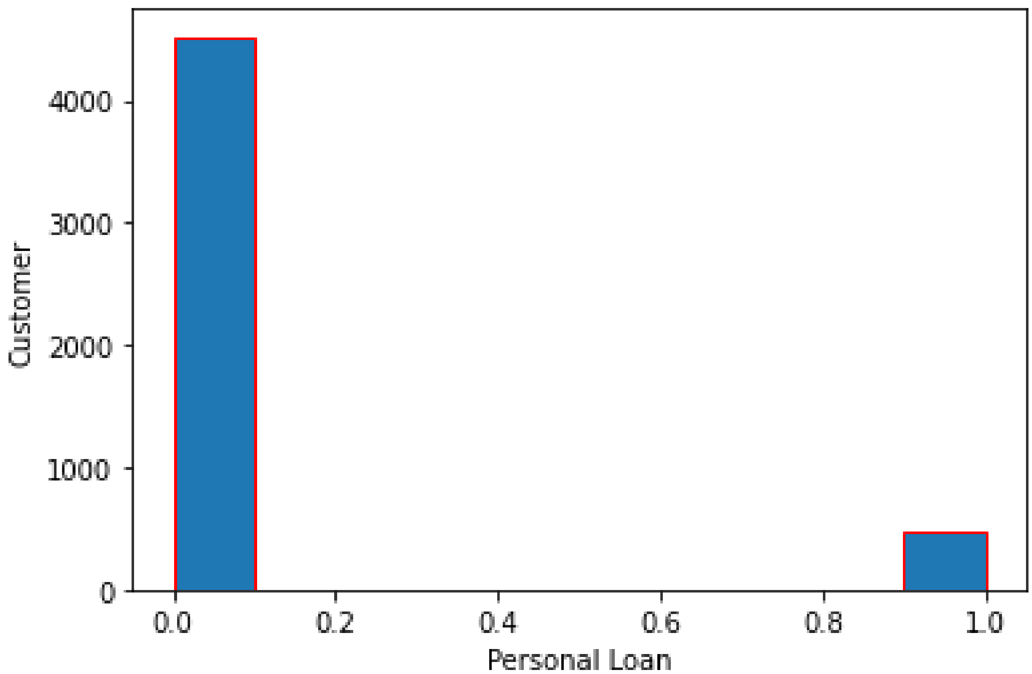


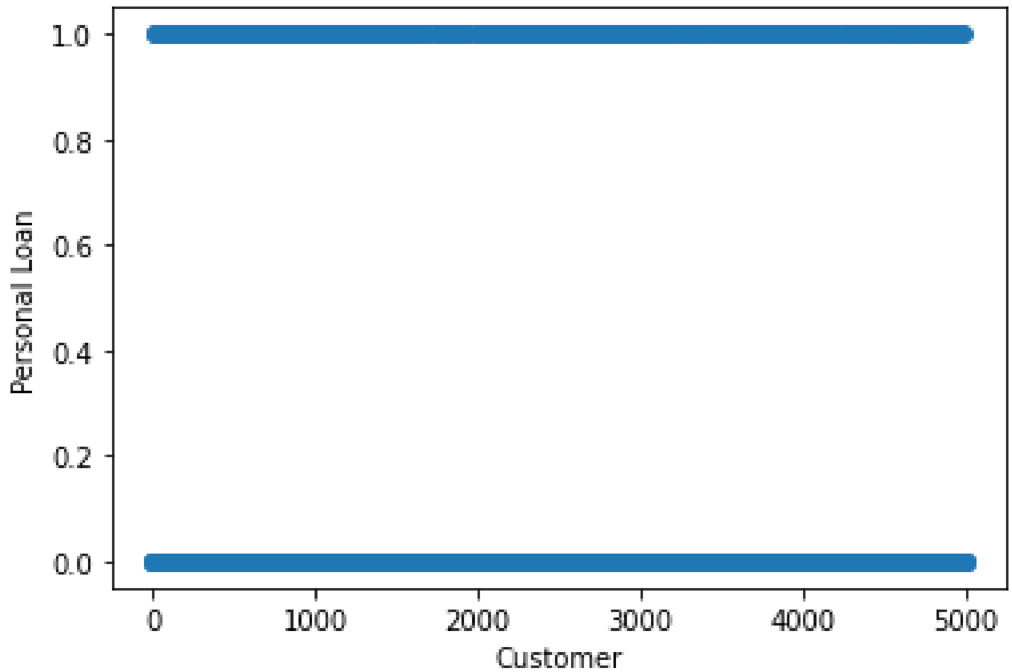




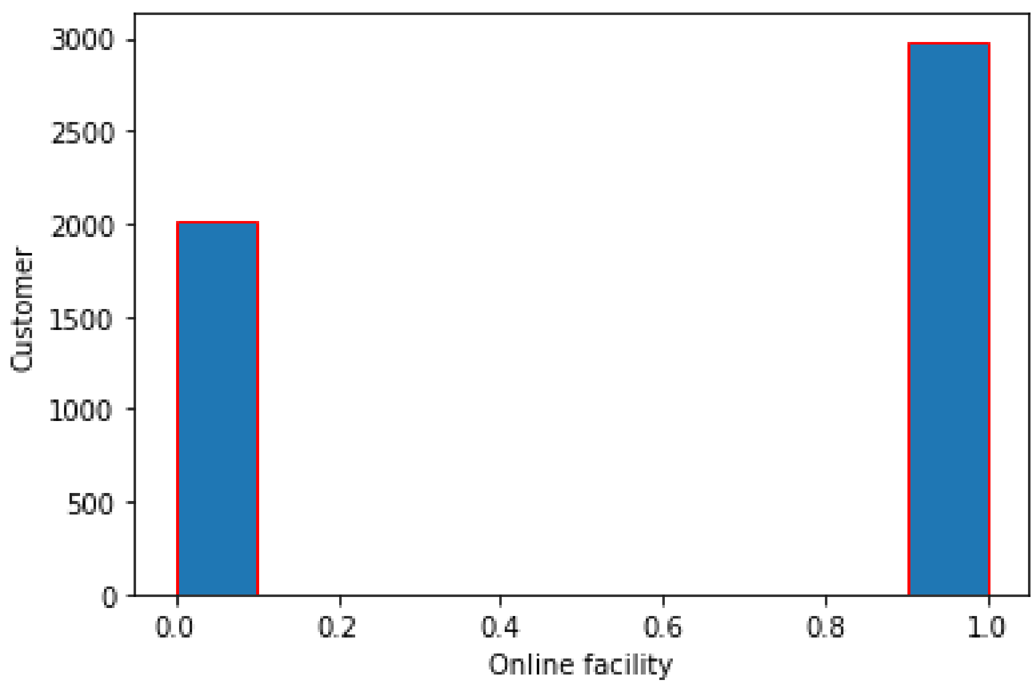

```
In [51]: plt.hist(df['Personal Loan'], edgecolor='Red')
         plt.xlabel('Personal Loan')
```

```
plt.ylabel('Customer')
plt.show()
plt.scatter(df.index,df['Personal Loan'])
plt.xlabel('Customer')
plt.ylabel('Personal Loan')
plt.show()
```

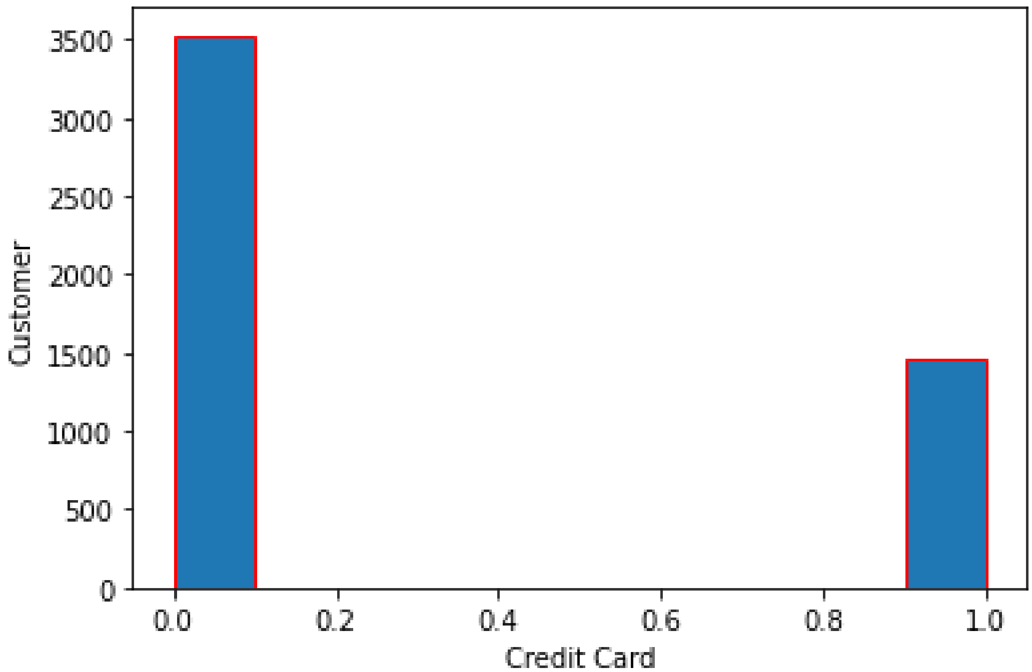




In [52]:
```
plt.hist(df['Online'], edgecolor='Red')
plt.xlabel('Online facility')
plt.ylabel('Customer')
plt.show()
```



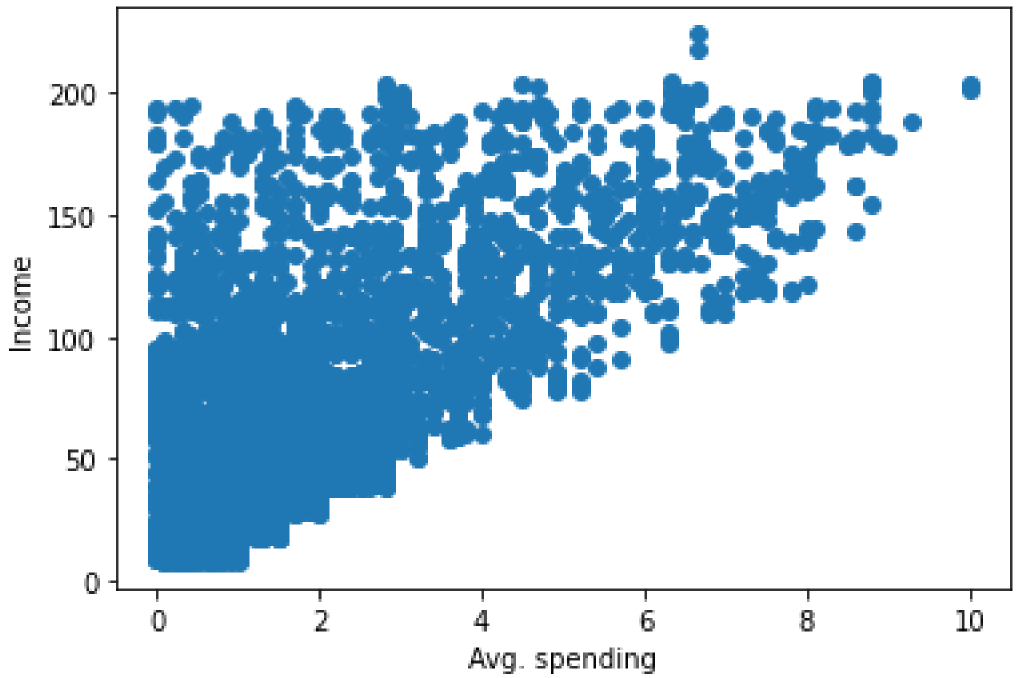
In [53]:
```
plt.hist(df['CreditCard'], edgecolor='Red')
plt.xlabel('Credit Card')
plt.ylabel('Customer')
plt.show()
```

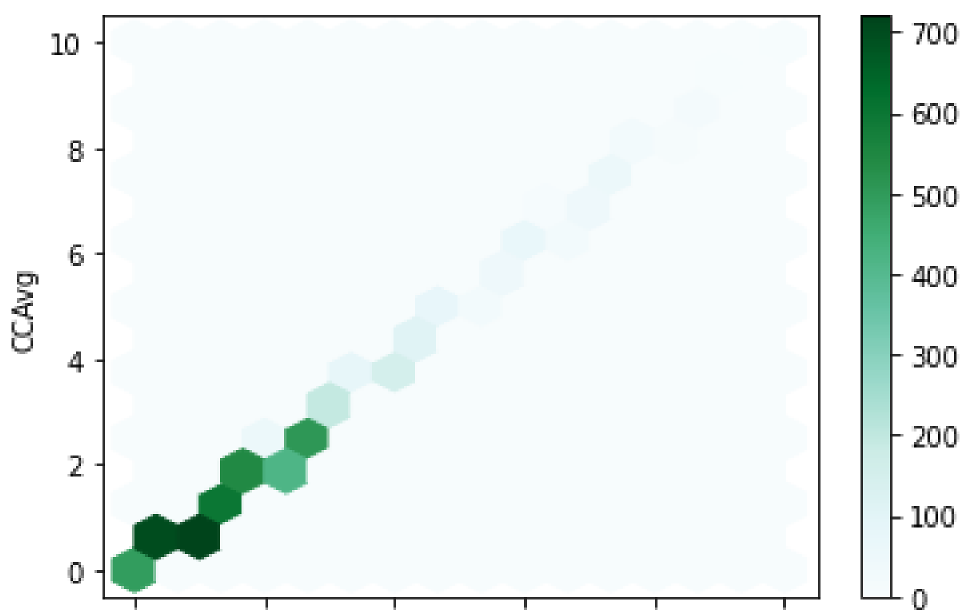


In [54]:
```
# Bivariate Analysis
```

In [55]:
```
plt.scatter(df['CCAvg'],df['Income'])
plt.xlabel('Avg. spending')
plt.ylabel('Income')
df.plot.hexbin(x='CCAvg', y='CCAvg', gridsize=15)
```

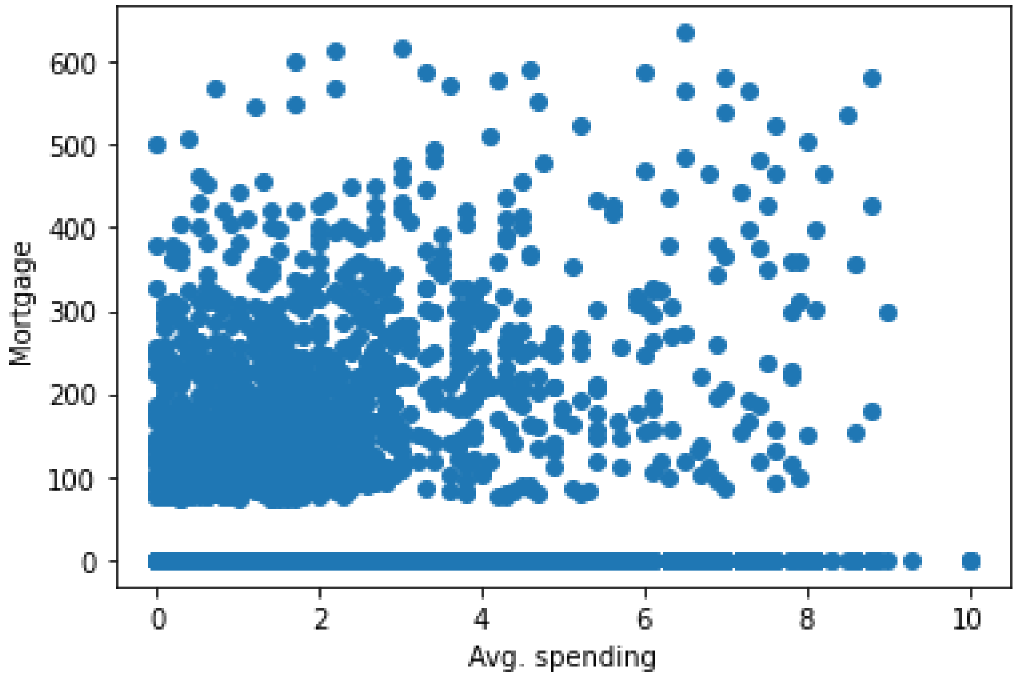
Out[55]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d6403fd30>
```





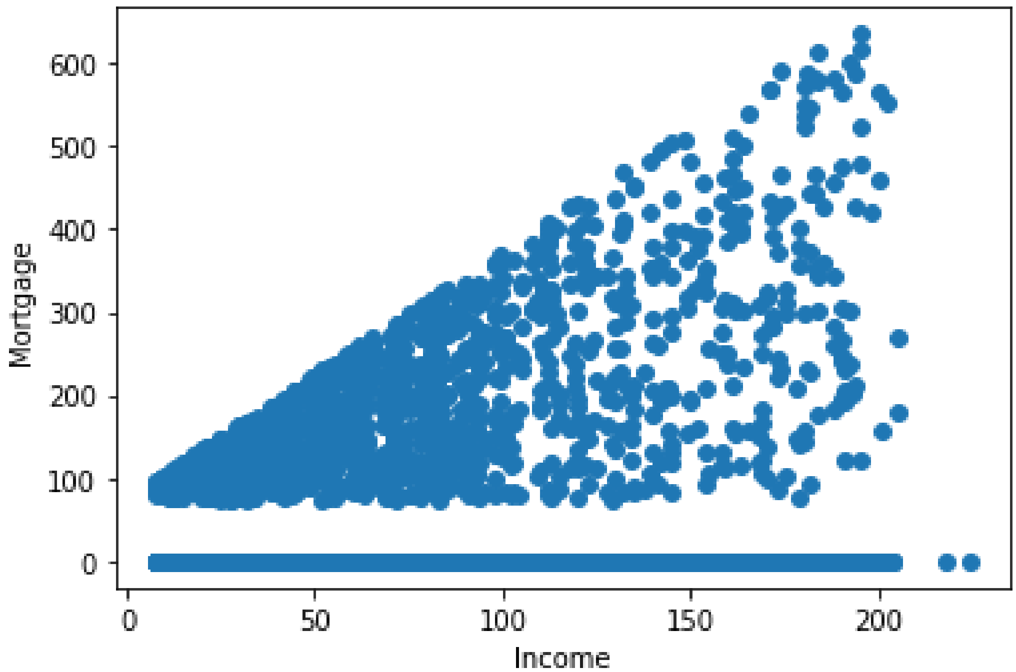

```
In [56]: plt.scatter(df['CCAvg'],df['Mortgage'])
         plt.xlabel('Avg. spending')
         plt.ylabel('Mortgage')
```

Out[56]: Text(0, 0.5, 'Mortgage')

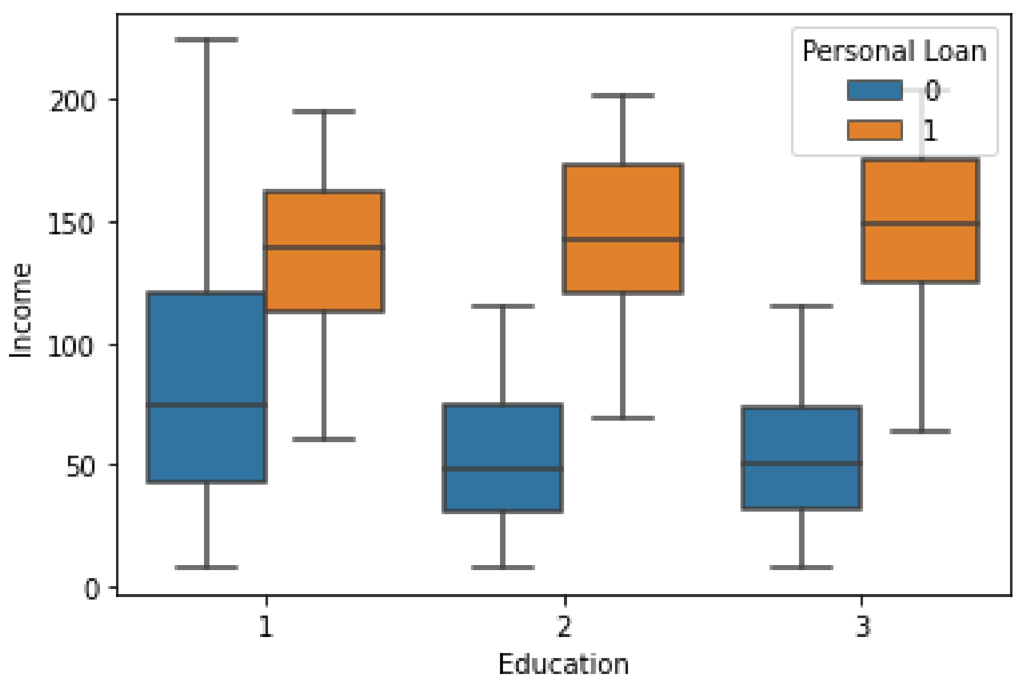


In [57]:
```
plt.scatter(df['Income'],df['Mortgage'])
plt.xlabel('Income')
plt.ylabel('Mortgage')
```

Out[57]: Text(0, 0.5, 'Mortgage')

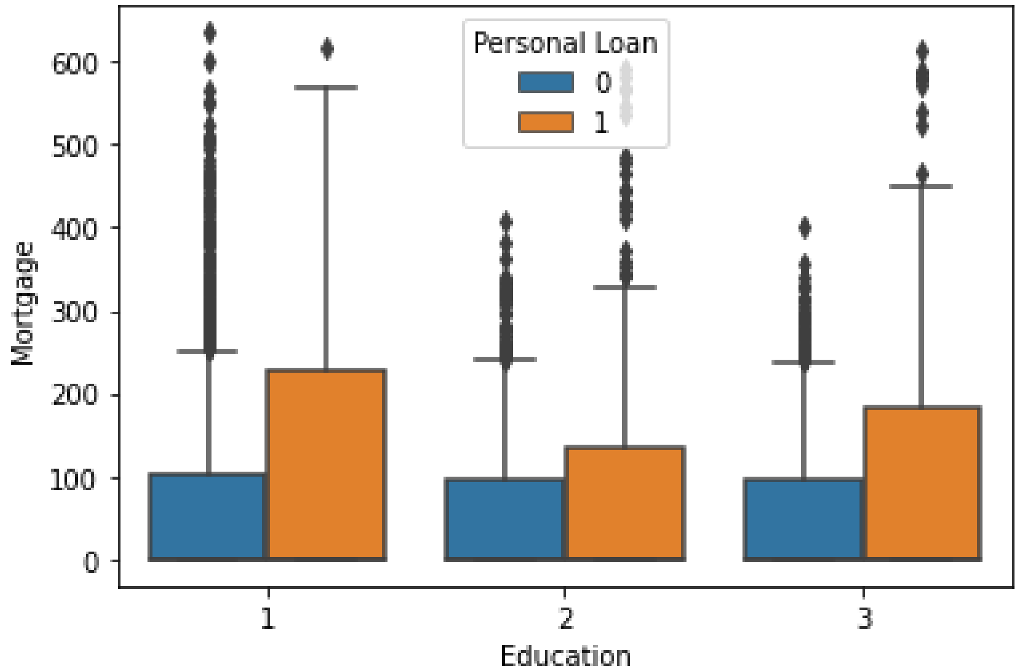



```
In [58]: sns.boxplot(x='Education',y='Income',hue='Personal Loan',data=df)

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d645b1b00>
```



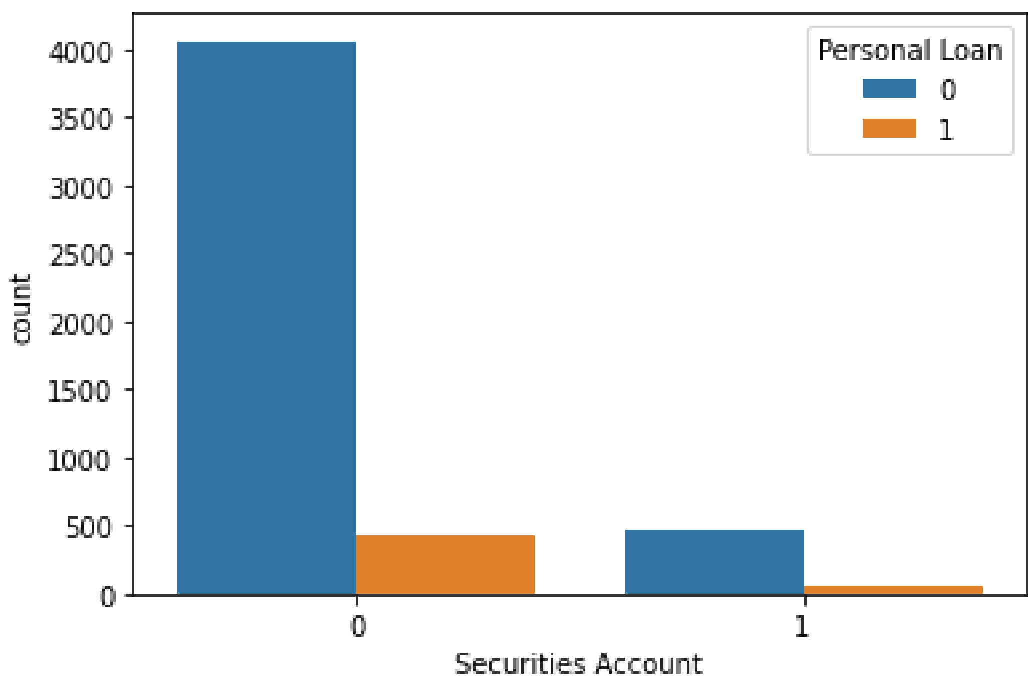
```
In [59]: sns.boxplot(x="Education", y='Mortgage', hue="Personal Loan", data=df)
```

```
Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d64898dd8>
```



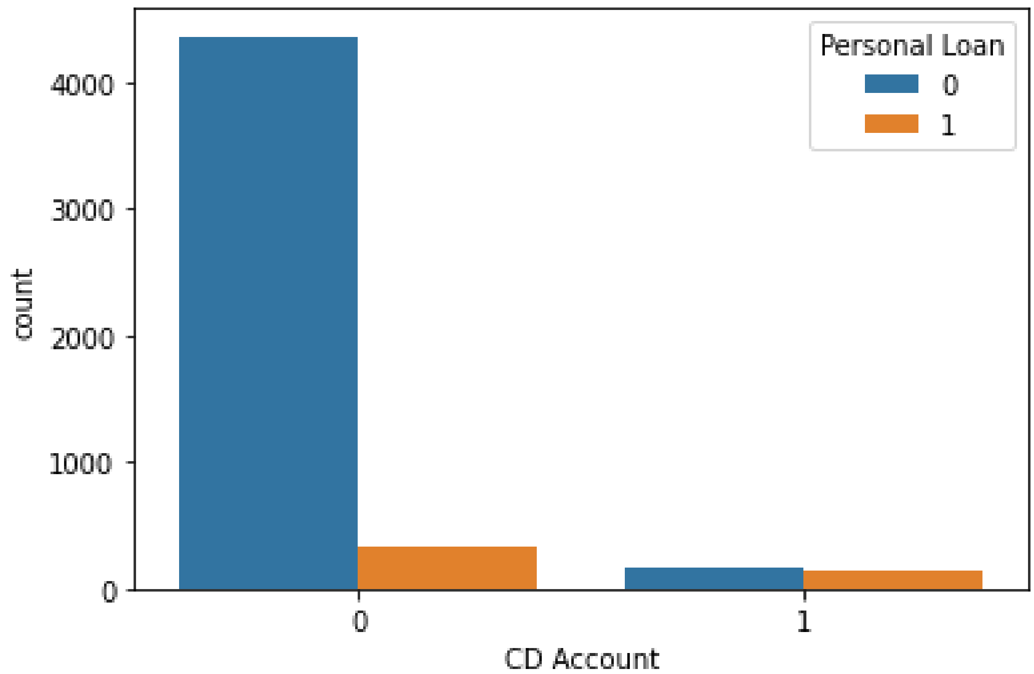

```
In [60]: sns.countplot(x="Securities Account", data=df,hue="Personal Loan")
```

```
Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d63ed2f60>
```



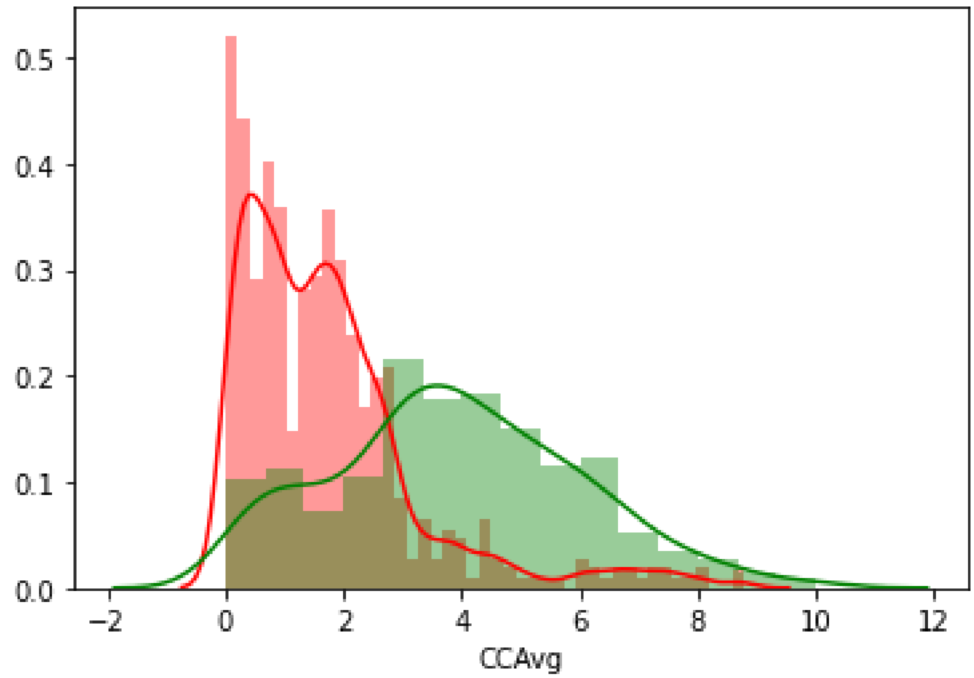
```
In [61]: sns.countplot(x='CD Account',data=df,hue='Personal Loan')
```

```
Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d647ba080>
```



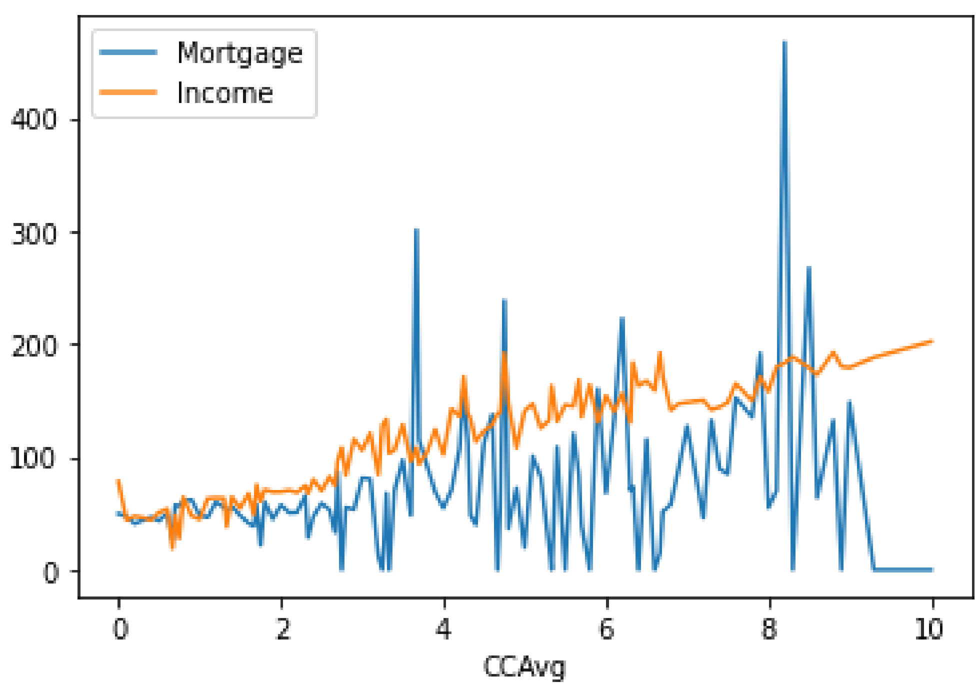

```
In [62]: sns.distplot( df[df['Personal Loan'] == 0]['CCAvg'], color = 'r')
         sns.distplot( df[df['Personal Loan'] == 1]['CCAvg'], color = 'g')
```

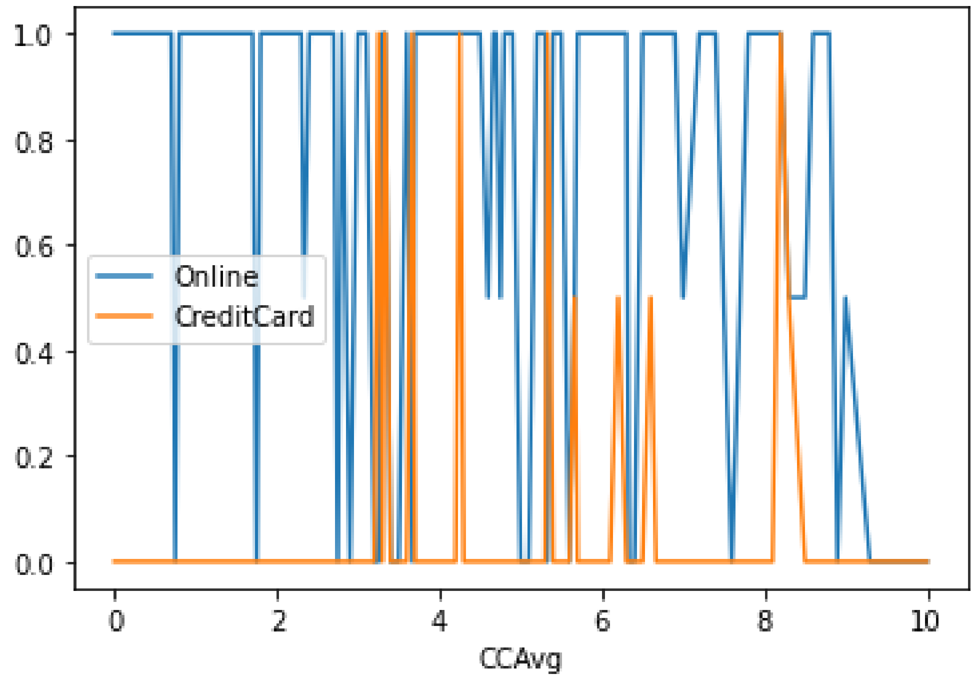
```
Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d63d14b70>
```



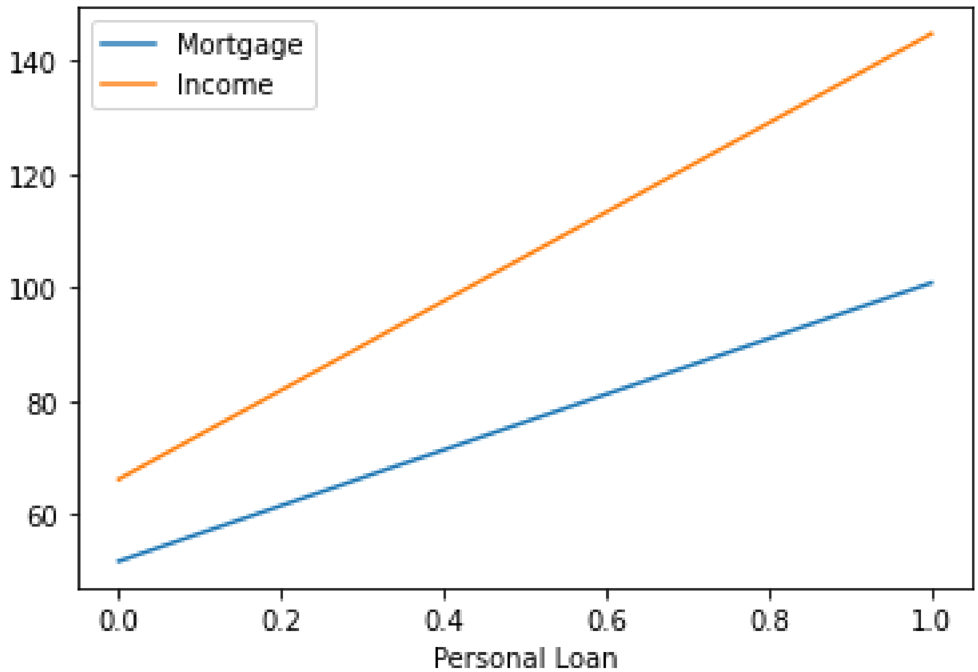
In [63]:
```
df_group = df.groupby('CCAvg').mean()[['Mortgage','Income']]
df_group.plot.line()
df_group2 = df.groupby('CCAvg').median()[['Online','CreditCard']]
df_group2.plot.line()
```

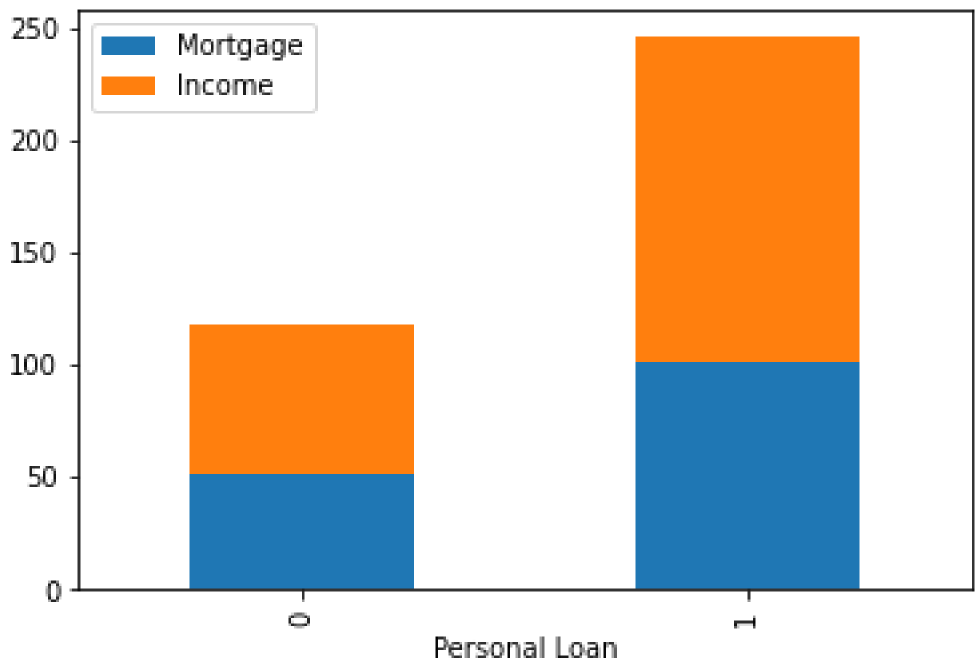
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d646d5be0>





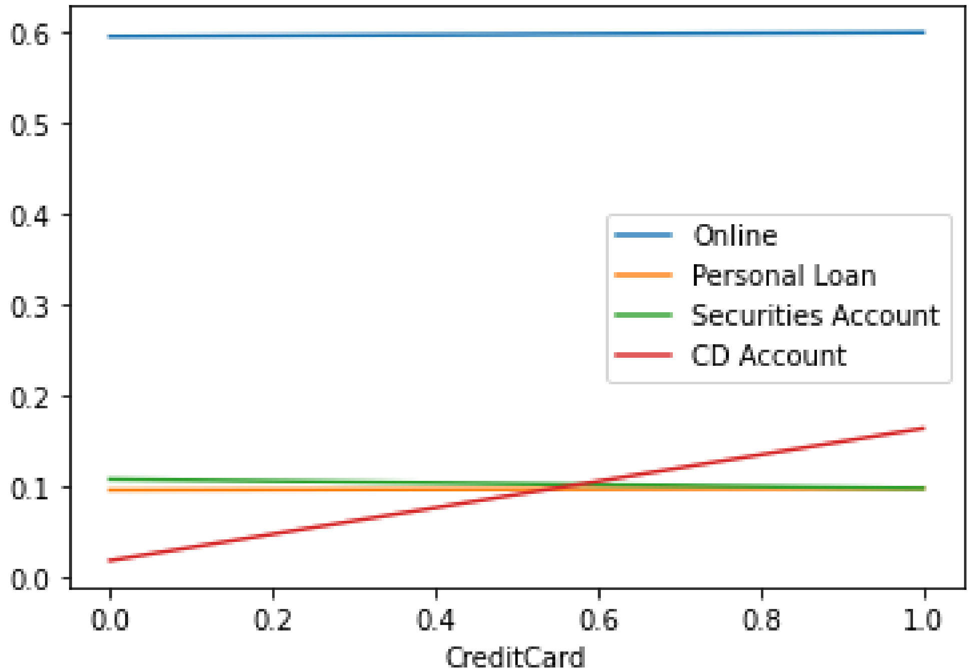

```
In [64]: df_group3 = df.groupby('Personal Loan').mean()[['Mortgage','Income']]
         df_group3.plot.line()
         df_group3.plot.bar(stacked=True)

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d63e50cc0>
```





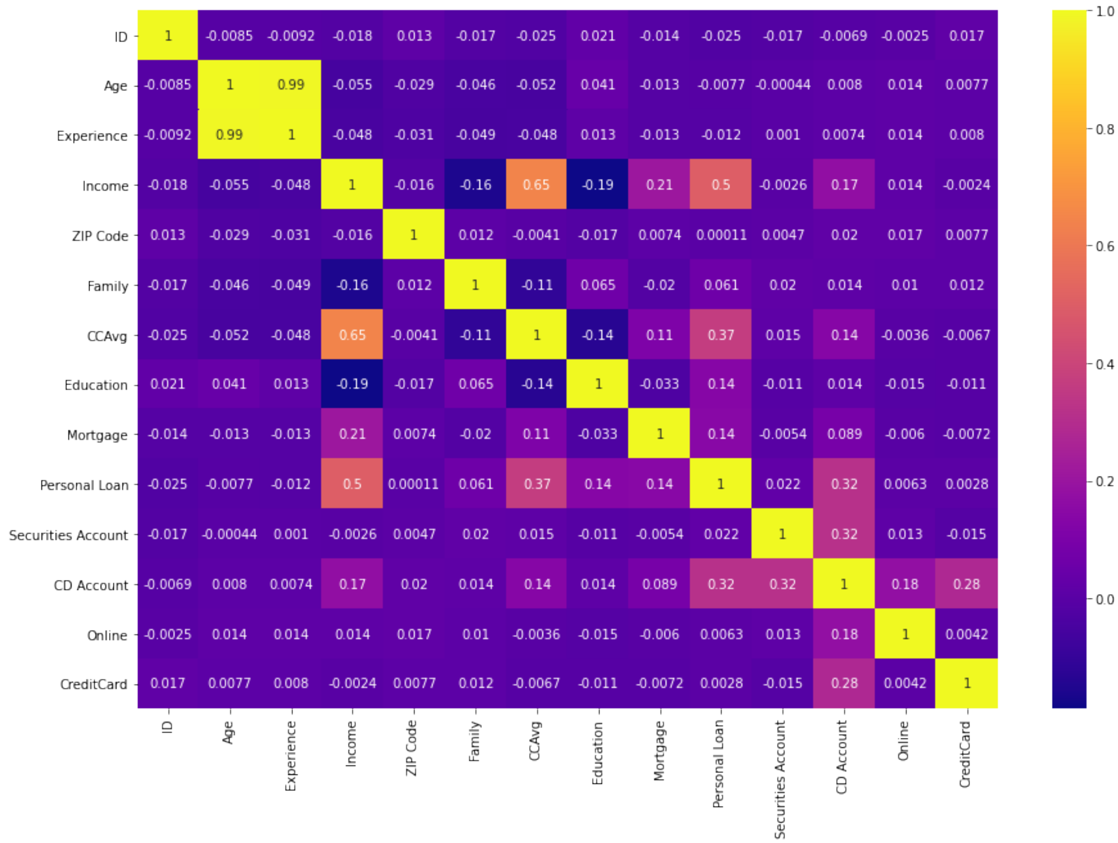

```
In [65]: df_group4 = df.groupby('CreditCard').mean()[['Online','Personal Loan','Securities Acc
         df_group4.plot.line()
         df_group4.plot.bar(stacked=True)
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d63c1e128>



```
In [66]: fig,ax = plt.subplots(figsize=(15,10))
         sns.heatmap(df.corr() , cmap='plasma' , annot=True)
```

```
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d63b49518>
```



(5) Normalise your data and split the data into training and test set in the ratio of 70:30 respectively

```
In [67]: from sklearn.model_selection import train_test_split
         target = 'Personal Loan'
         df_x = df.drop(target,axis='columns',inplace=False)
         df_y = df[target]

         x_train,x_test,y_train,y_test = train_test_split(df_x, df_y,test_size=0.30,random_sta
```

(4) Apply necessary transformations for the feature variables

```
In [68]: def quick_analysis(df):
          print('Data Types:')
          print(df.dtypes)
          print('Rows and Columns:')
          print(df.shape)
```

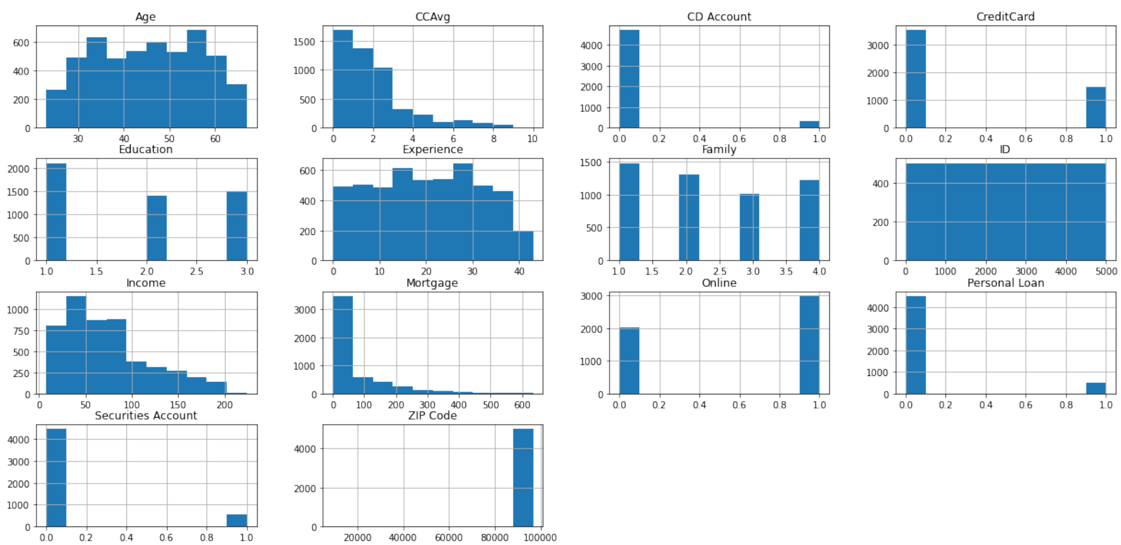
```python
    print('Column Names:')
    print(df.columns)
    print('Null Values:')
    print(df.apply(lambda x: sum(x.isnull()) / len(df)))


quick_analysis(df)
```

```
Data Types:
ID                      int64
Age                     int64
Experience            float64
Income                  int64
ZIP Code                int64
Family                  int64
CCAvg                 float64
Education               int64
Mortgage                int64
Personal Loan           int64
Securities Account      int64
CD Account              int64
Online                  int64
CreditCard              int64
dtype: object
Rows and Columns:
(5000, 14)
Column Names:
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard'],
      dtype='object')
Null Values:
ID                    0.0000
Age                   0.0000
Experience            0.0058
Income                0.0000
ZIP Code              0.0000
Family                0.0000
CCAvg                 0.0000
Education             0.0000
Mortgage              0.0000
Personal Loan         0.0000
Securities Account    0.0000
CD Account            0.0000
Online                0.0000
CreditCard            0.0000
dtype: float64
```

```
In [69]: df.describe()
```

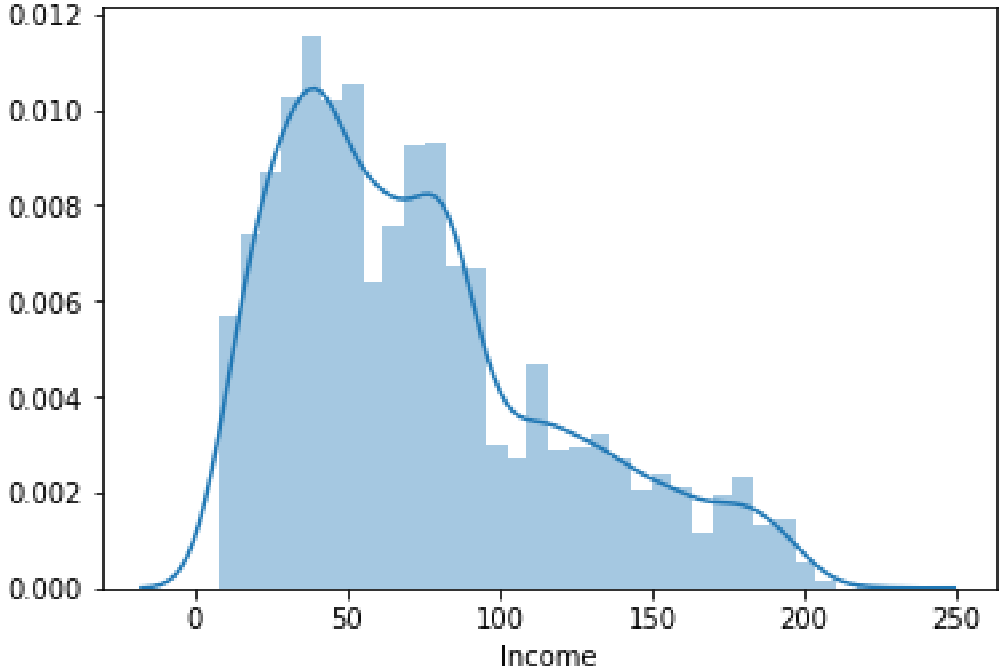
```
Out[69]:                 ID          Age  ...       Online   CreditCard
         count  5000.000000  5000.000000  ...  5000.000000  5000.000000
         mean   2500.500000    45.338400  ...     0.596800     0.294000
         std    1443.520003    11.463166  ...     0.490589     0.455637
         min       1.000000    23.000000  ...     0.000000     0.000000
         25%    1250.750000    35.000000  ...     0.000000     0.000000
         50%    2500.500000    45.000000  ...     1.000000     0.000000
         75%    3750.250000    55.000000  ...     1.000000     1.000000
         max    5000.000000    67.000000  ...     1.000000     1.000000

         [8 rows x 14 columns]
```

```
In [70]: df[df.dtypes[(df.dtypes=="float64")|(df.dtypes=="int64")].index.values].hist(figsize=
         plt.show()
```



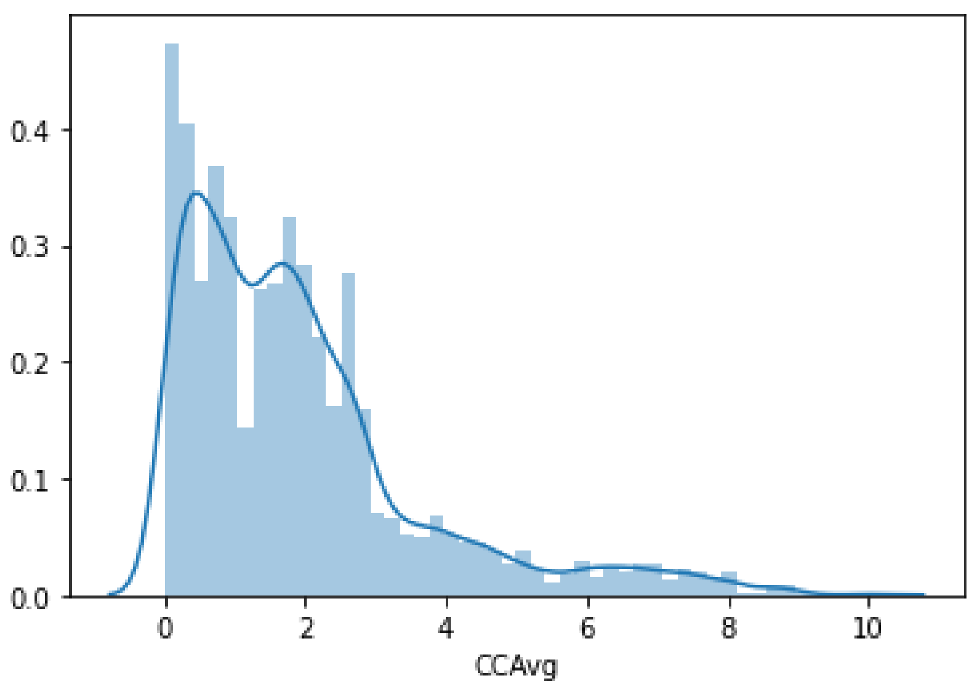

```
In [71]: sns.distplot(df.Income)
```

```
Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d5feb2780>
```



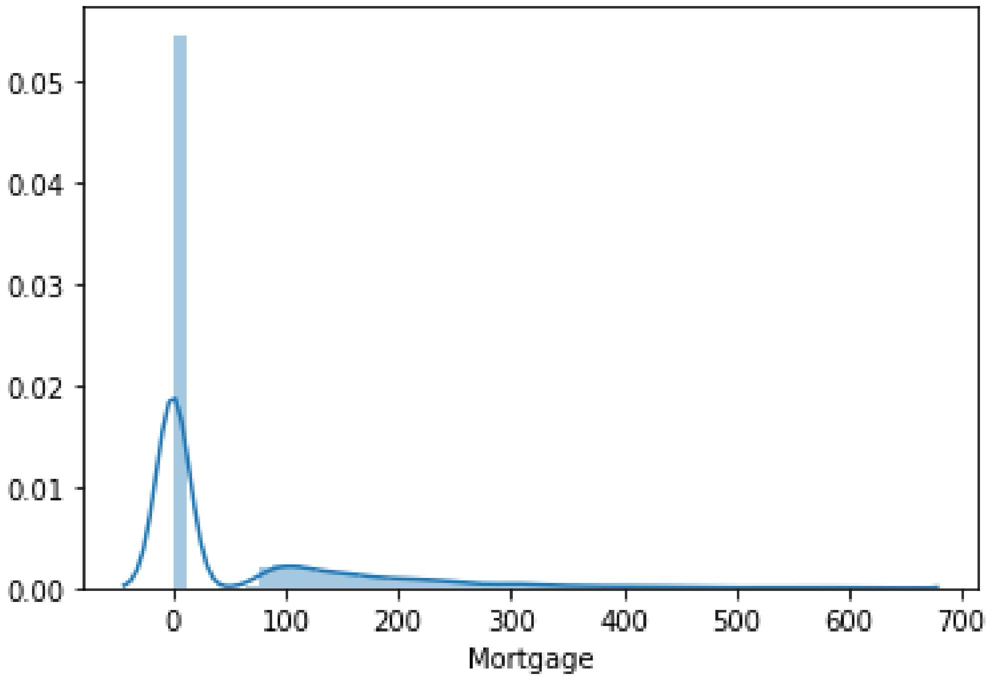

```
In [72]: sns.distplot(df.CCAvg)
```

```
Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d5ca23f60>
```

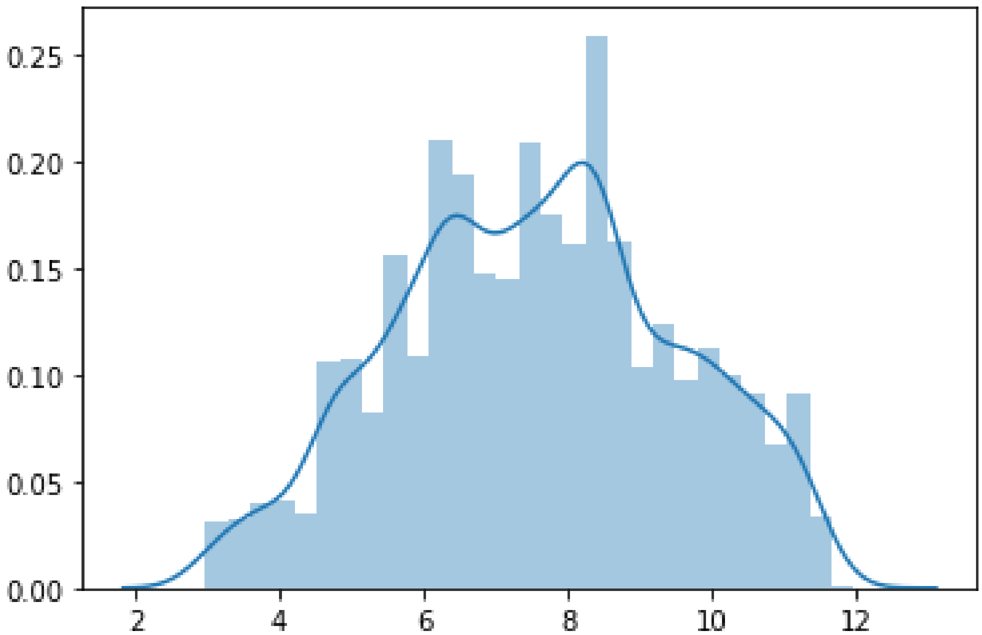


In [73]:
```
sns.distplot(df.Mortgage)
```

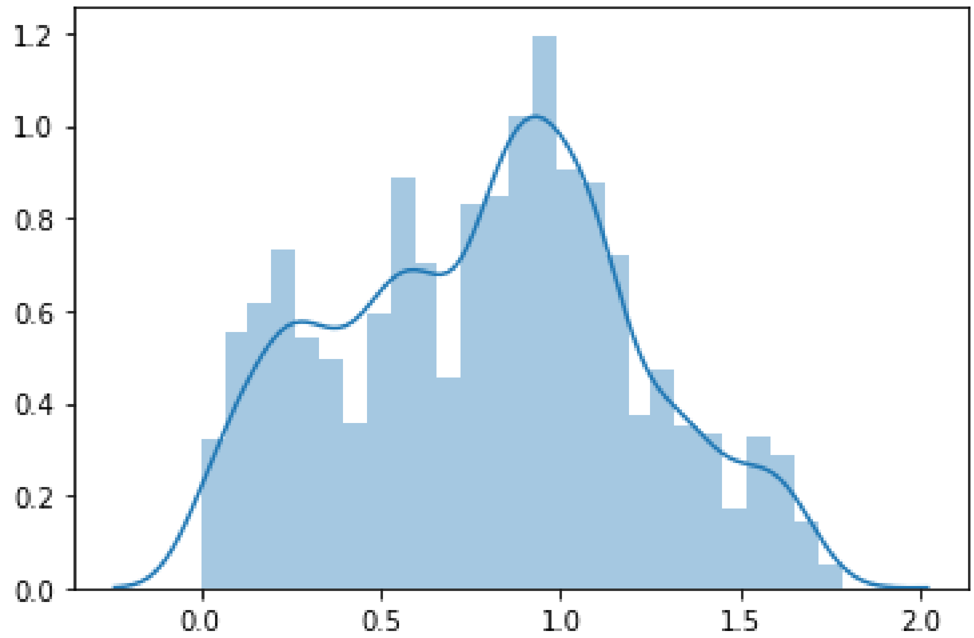
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d5c93e198>



In [74]:
```
from sklearn.preprocessing import PowerTransformer as pt

pwt = pt(method="yeo-johnson" , standardize=False)

pwt.fit(df_x["Income"].values.reshape(-1,1))
temp = pwt.transform(df_x["Income"].values.reshape(-1,1))
sns.distplot(temp)
plt.show()
```




```
In [75]: from sklearn.preprocessing import PowerTransformer as pt

         pwt = pt(method="yeo-johnson" , standardize=False)

         pwt.fit(df_x["CCAvg"].values.reshape(-1,1))
         temp = pwt.transform(df_x["CCAvg"].values.reshape(-1,1))
         sns.distplot(temp)

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d5c7b9b70>
```



(6) Use the Logistic Regression model to predict the likelihood of a customer buying personal loans

```
In [76]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score
         from sklearn.model_selection import train_test_split

In [84]: #In it our target is Personal Loan column so for x_dataset just drop this and for y_d
         target = 'Personal Loan'
         df_x = df.drop(target,axis='columns',inplace=False)
         df_y = df[target]

         x_train,x_test,y_train,y_test = train_test_split(df_x, df_y,test_size=0.30,random_sta

In [79]: #check if their is null value exist
         df.isnull().sum().sort_values(ascending = False)

Out[79]: Experience            29
         CreditCard             0
         Online                 0
         CD Account             0
         Securities Account     0
         Personal Loan          0
         Mortgage               0
         Education              0
         CCAvg                  0
         Family                 0
```

```
ZIP Code              0
Income                0
Age                   0
ID                    0
dtype: int64
```

```
In [82]: #if null value is their then drop
         df = df.dropna(subset=['Experience'])
         df.isnull().sum().sort_values(ascending = False)
```

```
Out[82]: CreditCard            0
         Online                0
         CD Account            0
         Securities Account    0
         Personal Loan         0
         Mortgage              0
         Education             0
         CCAvg                 0
         Family                0
         ZIP Code              0
         Income                0
         Experience            0
         Age                   0
         ID                    0
         dtype: int64
```

```
In [140]: target = 'Personal Loan'
          df_x = df.drop(target,axis='columns',inplace=False)
          df_y = df[target]


          #so perform Logistic Regression with the split of 70:30 in training and testing data
          x_train,x_test,y_train,y_test = train_test_split(df_x, df_y,test_size=0.30,random_sta
          L_R = LogisticRegression(max_iter=1000)
          L_R.fit(x_train,y_train)
```

```
Out[140]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, l1_ratio=None, max_iter=1000,
                             multi_class='auto', n_jobs=None, penalty='l2',
                             random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                             warm_start=False)
```

```
In [122]: #Accuracy for the model
          y_pred = L_R.predict(x_test)
          print("Accuracy of logistic regression classifier on test set :",accuracy_score(y_tes
```

```
Accuracy of logistic regression classifier on test set : 94.16890080428955
```

```
In [123]: print("Testing Accuracy",L_R.fit(x_train,y_train).score(x_test,y_test)*100)
```

```
Testing Accuracy 94.16890080428955
```

```
In [124]: print("Training Accuracy" , L_R.fit(x_train,y_train).score(x_train,y_train)*100)
```

```
Training Accuracy 93.7625754527163
```

(7) Print all the metrics related for evaluating the model performance

```
In [125]: #classification report contains all major aspect which are the very useful for analy
          from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1348
           1       0.76      0.58      0.66       144

    accuracy                           0.94      1492
   macro avg       0.86      0.78      0.81      1492
weighted avg       0.94      0.94      0.94      1492
```

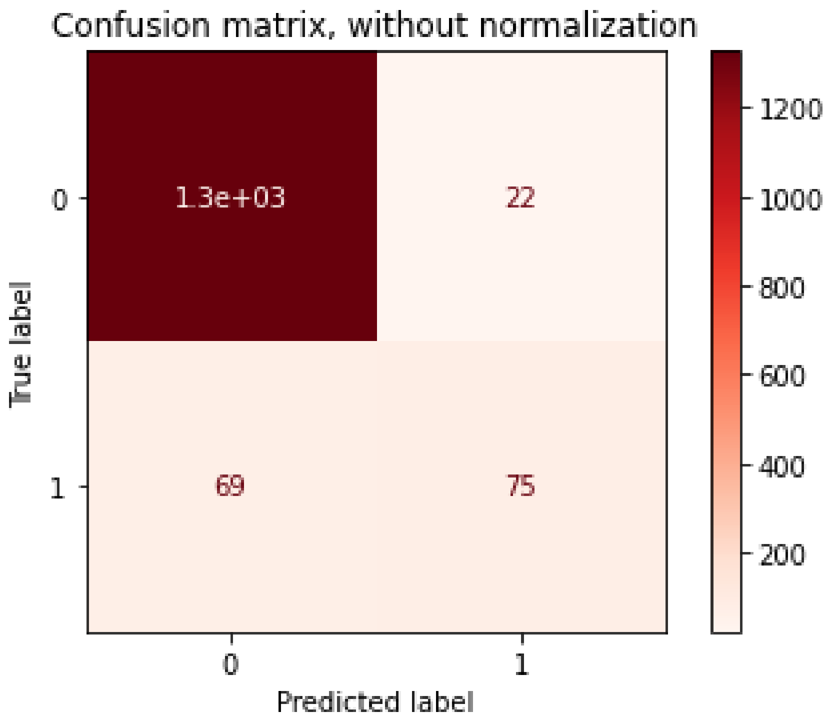
```
In [126]: from sklearn.metrics import confusion_matrix
          confusion_matrix = confusion_matrix(y_test, y_pred)
          print(confusion_matrix)
```

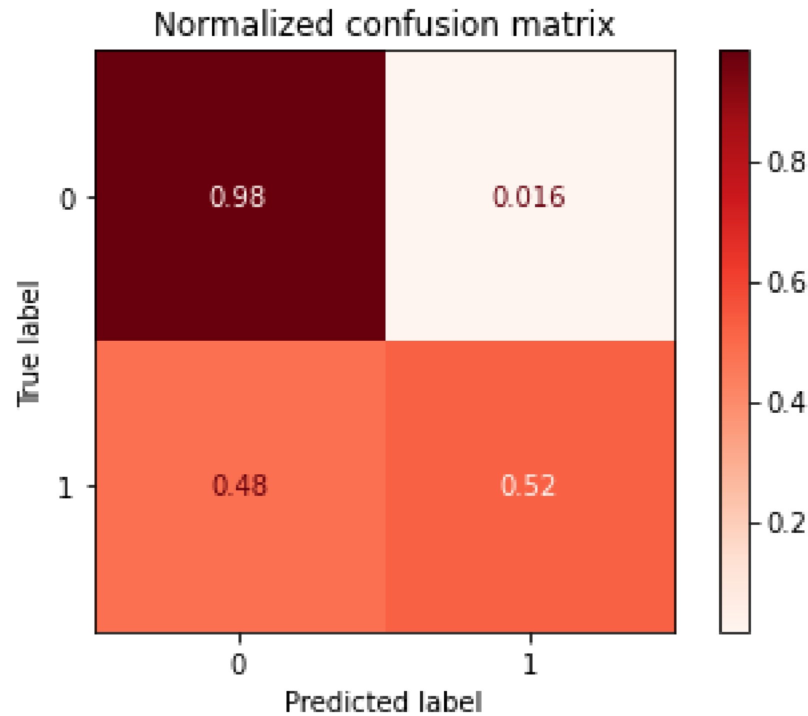
```
[[1321   27]
 [  60   84]]
```

```
In [138]: #show the confusion matrix with and without normalization
          from sklearn.metrics import plot_confusion_matrix
          classifier = LogisticRegression(max_iter=1000,C=0.01).fit(x_train, y_train)

          np.set_printoptions(precision=2)

          # Plot non-normalized confusion matrix
          titles_options = [("Confusion matrix, without normalization", None),
                            ("Normalized confusion matrix", 'true')]
          for title, normalize in titles_options:
              disp = plot_confusion_matrix(classifier, x_test, y_test,
                                           cmap=plt.cm.Reds,
                                           normalize=normalize)
              disp.ax_.set_title(title)

              print(title)
```

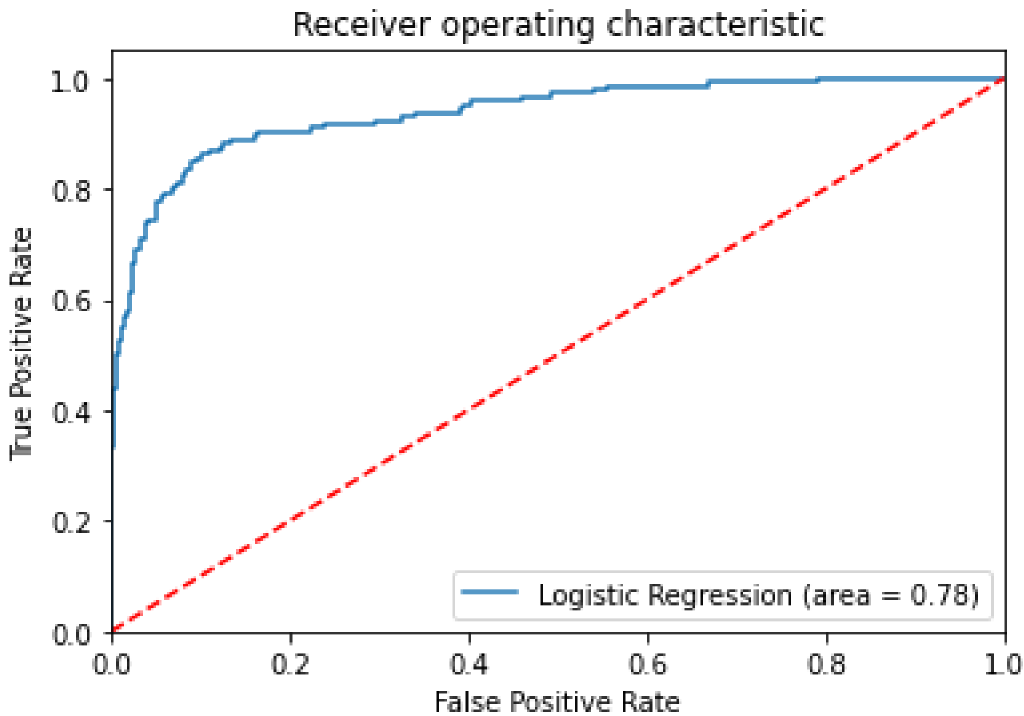
```
        print(disp.confusion_matrix)

    plt.show()
```

```
Confusion matrix, without normalization
[[1326   22]
 [  69   75]]
Normalized confusion matrix
[[0.98 0.02]
 [0.48 0.52]]
```






```
In [153]: #ROC ( Receiver Operating Characteristic ) Curve

          #a good classifier stays as far away from that line as possible (toward the top-left
          from sklearn.metrics import roc_auc_score
          from sklearn.metrics import roc_curve
          logit_roc_auc = roc_auc_score(y_test, L_R.predict(x_test))
          fpr, tpr, thresholds = roc_curve(y_test, L_R.predict_proba(x_test)[:,1])
          plt.figure()
          plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
          plt.plot([0, 1], [0, 1],'r--')
          plt.xlim([0.0, 1.0])
          plt.ylim([0.0, 1.05])
          plt.xlabel('False Positive Rate')
          plt.ylabel('True Positive Rate')
          plt.title('Receiver operating characteristic')
          plt.legend(loc="lower right")
          plt.savefig('Log_ROC')
          plt.show()
```



In [129]:
```
pred_1_test_x = L_R.predict(x_test)
pred_1_train_x = L_R.predict(x_train)

from sklearn.metrics import recall_score,precision_score,f1_score,roc_auc_score
print("Recall : ",recall_score(y_test , pred_1_test_x))
print("Precision : ", precision_score(y_test , pred_1_test_x))
print("F1 Score : ", f1_score(y_test , pred_1_test_x))
print("Roc Auc Score : ", roc_auc_score(y_test , pred_1_test_x))

#Normal Logistic Regression give the accuracy 94% with 58% Recall and 78% Roc.
```

```
Recall :  0.5833333333333334
Precision :  0.7567567567567568
F1 Score :  0.6588235294117648
Roc Auc Score :  0.7816518298714146
```

(8) Build various other classification algorithms and compare their performance

In [130]:
```
#AFTER APPLYING STANDARDIZATION IN LOGISTIC REGRESSION

from sklearn import preprocessing

col_names = df.columns
```

```
scaler=preprocessing.StandardScaler()
scaled_x_train=scaler.fit_transform(x_train)
scaled_x_test=scaler.fit_transform(x_test)
L_R = LogisticRegression()
L_R.fit(scaled_x_train,y_train)

from sklearn.metrics import recall_score,precision_score,f1_score,roc_auc_score,accu
from sklearn.metrics import roc_curve,auc

y_pred1 = L_R.predict(scaled_x_test)
print(classification_report(y_test,y_pred1))
print(accuracy_score(y_test,y_pred1))
print(confusion_matrix(y_test,y_pred1))

LR_prob=L_R.predict_proba(scaled_x_test)
fpr1,tpr1,thresholds1=roc_curve(y_test,LR_prob[:,1])
roc_auc1=auc(fpr1,tpr1)
print("Area under the Roc curve : %f " % roc_auc1)

pred_1_test_x = L_R.predict(scaled_x_test)
pred_1_train_x = L_R.predict(scaled_x_train)
print("Recall : ",recall_score(y_test , pred_1_test_x))
print("Precision : ", precision_score(y_test , pred_1_test_x))
print("F1 Score : ", f1_score(y_test , pred_1_test_x))
print("Roc Auc Score : ", roc_auc_score(y_test , pred_1_test_x))

# Standardization Logistic Regression give the accuracy 95% with 63% Recall and 81% 
# which is better then Logistic Regression .
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1348
           1       0.81      0.64      0.72       144

    accuracy                           0.95      1492
   macro avg       0.89      0.81      0.84      1492
weighted avg       0.95      0.95      0.95      1492

0.9510723860589813
[[1327   21]
 [  52   92]]
Area under the Roc curve : 0.950137
Recall :  0.6388888888888888
Precision :  0.8141592920353983
F1 Score :  0.7159533073929961
Roc Auc Score :  0.811655126937026
```

In [131]:
```
#K-Nearest Neighbor
```

```python
from sklearn.metrics import roc_curve,auc
from sklearn.neighbors import KNeighborsClassifier
knn_Model=KNeighborsClassifier(n_neighbors=3)
knn_Model.fit(scaled_x_train,y_train)

y_pred=knn_Model.predict(scaled_x_test)

print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))


kNN_prob=knn_Model.predict_proba(scaled_x_test)
fpr2,tpr2,thresholds2=roc_curve(y_test,kNN_prob[:,1])
roc_auc2=auc(fpr2,tpr2)
print("Area under the Roc curve : %f " % roc_auc2)

pred_1_test_x = knn_Model.predict(scaled_x_test)
pred_1_train_x = knn_Model.predict(scaled_x_train)
print("Recall : ",recall_score(y_test , pred_1_test_x))
print("Precision : ", precision_score(y_test , pred_1_test_x))
print("F1 Score : ", f1_score(y_test , pred_1_test_x))
print("Roc Auc Score : ", roc_auc_score(y_test , pred_1_test_x))

# K-Nearest Neighbor give the accuracy 94% with 53% Recall and 76% Roc.
# which is worse then Normal Logistic Regression.
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      1348
           1       0.88      0.53      0.66       144

    accuracy                           0.95      1492
   macro avg       0.91      0.76      0.82      1492
weighted avg       0.94      0.95      0.94      1492

0.9477211796246648
[[1337   11]
 [  67   77]]
Area under the Roc curve : 0.884525
Recall :  0.5347222222222222
Precision :  0.875
F1 Score :  0.6637931034482758
Roc Auc Score :  0.7632809924167491
```

In [132]:
```python
#NAIVE BAYES
from sklearn.naive_bayes import GaussianNB
```

```
naive_model = GaussianNB()
naive_model.fit(scaled_x_train,y_train)
y_pred=naive_model.predict(scaled_x_test)

print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))

nbm_prob=naive_model.predict_proba(scaled_x_test)
fpr3,tpr3,thresholds3=roc_curve(y_test,nbm_prob[:,1])
roc_auc3=auc(fpr3,tpr3)
print("Area under the Roc curve : %f " % roc_auc3)


pred_1_test_x = naive_model.predict(scaled_x_test)
pred_1_train_x = naive_model.predict(scaled_x_train)
print("Recall : ",recall_score(y_test , pred_1_test_x))
print("Precision : ", precision_score(y_test , pred_1_test_x))
print("F1 Score : ", f1_score(y_test , pred_1_test_x))
print("Roc Auc Score : ", roc_auc_score(y_test , pred_1_test_x))

# Naive Bayes give the accuracy 88% with 53% Recall and 72% Roc.
# which is  even worse then K-Nearest Neighbor.
```

```
              precision    recall  f1-score   support

           0       0.95      0.92      0.93      1348
           1       0.41      0.53      0.46       144

    accuracy                           0.88      1492
   macro avg       0.68      0.73      0.70      1492
weighted avg       0.90      0.88      0.89      1492

0.8806970509383378
[[1237  111]
 [  67   77]]
Area under the Roc curve : 0.920989
Recall :  0.5347222222222222
Precision :  0.4095744680851064
F1 Score :  0.463855421686747
Roc Auc Score :  0.7261890042861853
```

```
In [133]: #SUPPORT VECTOR MACHINE
          from sklearn import svm
          clfr=svm.SVC(C=3,kernel='rbf',probability=True)
```

```
clfr.fit(scaled_x_train,y_train)
y_pred= clfr.predict(scaled_x_test)

print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))

svm_prob=clfr.predict_proba(scaled_x_test)
fpr4,tpr4,thresholds4=roc_curve(y_test,svm_prob[:,1])
roc_auc4=auc(fpr4,tpr4)
print("Area under the Roc curve : %f " % roc_auc4)


pred_1_test_x = clfr.predict(scaled_x_test)
pred_1_train_x = clfr.predict(scaled_x_train)
print("Recall : ",recall_score(y_test , pred_1_test_x))
print("Precision : ", precision_score(y_test , pred_1_test_x))
print("F1 Score : ", f1_score(y_test , pred_1_test_x))
print("Roc Auc Score : ", roc_auc_score(y_test , pred_1_test_x))

# Support Vector Machine give the accuracy 97% with 80% Recall and 90% Roc.
# which is better then Standardization Logistic Regression.
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1348
           1       0.96      0.81      0.88       144

    accuracy                           0.98      1492
   macro avg       0.97      0.90      0.93      1492
weighted avg       0.98      0.98      0.98      1492

0.9778820375335121
[[1343    5]
 [  28  116]]
Area under the Roc curve : 0.981155
Recall :  0.8055555555555556
Precision :  0.9586776859504132
F1 Score :  0.8754716981132076
Roc Auc Score :  0.9009231783712496
```

In [134]:
```
#DECISION TREE
from sklearn.tree import DecisionTreeClassifier
d_t=DecisionTreeClassifier(criterion='entropy' , random_state=1)
d_t.fit(scaled_x_train,y_train)
y_pred=d_t.predict(scaled_x_test)
```

```
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))

dt_prob=d_t.predict_proba(scaled_x_test)
fpr5,tpr5,thresholds5=roc_curve(y_test,dt_prob[:,1])
roc_auc5=auc(fpr5,tpr5)
print("Area under the Roc curve : %f " % roc_auc5)


pred_1_test_x = d_t.predict(scaled_x_test)
pred_1_train_x = d_t.predict(scaled_x_train)
print("Recall : ",recall_score(y_test , pred_1_test_x))
print("Precision : ", precision_score(y_test , pred_1_test_x))
print("F1 Score : ", f1_score(y_test , pred_1_test_x))
print("Roc Auc Score : ", roc_auc_score(y_test , pred_1_test_x))

# Decison Tree give the accuracy 98.3% with 87% Recall and 93.4% Roc.
# which is better then SVM.
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1348
           1       0.95      0.88      0.91       144

    accuracy                           0.98      1492
   macro avg       0.97      0.93      0.95      1492
weighted avg       0.98      0.98      0.98      1492

0.9832439678284183
[[1341    7]
 [  18  126]]
Area under the Roc curve : 0.934904
Recall :  0.875
Precision :  0.9473684210526315
F1 Score :  0.9097472924187725
Roc Auc Score :  0.9349035608308606
```

In [135]:
```
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
r_f=RandomForestClassifier(criterion='entropy',n_estimators=150,max_features=6,randon
r_f.fit(scaled_x_train,y_train)
y_pred=r_f.predict(scaled_x_test)

print(classification_report(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
rf_prob=r_f.predict_proba(scaled_x_test)
fpr6,tpr6,thresholds6=roc_curve(y_test,rf_prob[:,1])
roc_auc6=auc(fpr6,tpr6)
print("Area under the Roc curve : %f " % roc_auc6)


pred_1_test_x = r_f.predict(scaled_x_test)
pred_1_train_x = r_f.predict(scaled_x_train)
print("Recall : ",recall_score(y_test , pred_1_test_x))
print("Precision : ", precision_score(y_test , pred_1_test_x))
print("F1 Score : ", f1_score(y_test , pred_1_test_x))
print("Roc Auc Score : ", roc_auc_score(y_test , pred_1_test_x))

# Random Forest give the accuracy 98.7% with 86% Recall and 93% Roc.
# which is better then Decision Tree.
```

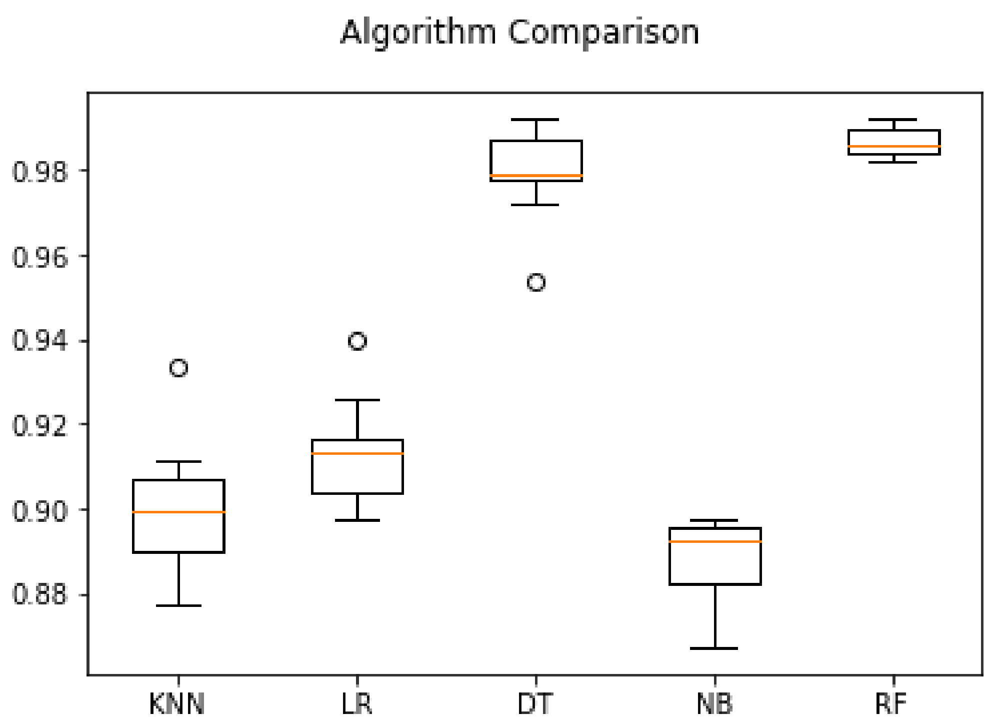
```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1348
           1       1.00      0.87      0.93       144

    accuracy                           0.99      1492
   macro avg       0.99      0.93      0.96      1492
weighted avg       0.99      0.99      0.99      1492

0.9872654155495979
[[1348    0]
 [  19  125]]
Area under the Roc curve : 0.997306
Recall :  0.8680555555555556
Precision :  1.0
F1 Score :  0.929368029739777
Roc Auc Score :  0.9340277777777778
```

(9) Business understanding of the model

In [137]:
```
#It show the comparison boxplot graph of various machinelearning algorithms model

from sklearn import model_selection

models = []
models.append(('KNN', KNeighborsClassifier()))
models.append(('LR', LogisticRegression(max_iter=1000)))
models.append(('DT', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
```

```python
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10)
        cv_results = model_selection.cross_val_score(model, df_x, df_y, cv=kfold, sco
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
KNN: 0.899819 (0.015683)
LR: 0.913297 (0.011714)
DT: 0.979285 (0.010395)
NB: 0.887546 (0.009978)
RF: 0.986522 (0.003123)
```



Conclusion

The aim of the bank is to convert there liability customers into loan customers. They want to set up a new marketing campaign; hence, they need information about the connection between the variables given in the data. So from six classification algorithms were used in this study. From the above graph , it seems like **Random Forest algorithm** have the highest accuracy and we can choose that as our final model.