```
import random
import time
import pandas as pd
import numpy as np

#ignore warnings
import warnings
warnings.filterwarnings('ignore')
print('-'*25)
```

```
--------------------------
```

```
#Common Model Algorithms
from sklearn import svm, tree, linear_model, neighbors, naive_bayes, ensemble, discriminant_
from xgboost import XGBClassifier

#Common Model Helpers
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics

#Visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import seaborn as sns
#from pandas.tools.plotting import scatter_matrix

#Configure Visualization Defaults
#%matplotlib inline = show plots in Jupyter Notebook browser
%matplotlib inline
mpl.style.use('ggplot')
sns.set_style('white')
pylab.rcParams['figure.figsize'] = 12,8
import io
```

ASSIGNMENT QUESTIONS

1. Read the column description and ensure you understand each attribute well
2. Study the data distribution in each attribute, share your findings. (5 points)
3. Get the target column distribution. Your comments
4. Split the data into training and test set in the ratio of 70:30 respectively
5. Use different classification models (Logistic, K-NN and Naïve Bayes) to predict the likelihood of a liability cust
6. *Print* the confusion matrix for all the above models (5 points)
7. Give your reasoning on which is the best model in this case and why it performs better? (5 points)

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=

Enter your authorization code:
··········
Mounted at /content/drive
```

```
file = '/content/drive/My Drive/PGML/supervisedlearning/Bank_Personal_Loan_Modelling-1.csv'
df = pd.read_csv(file)
df.head()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Perso
0	1	25	1	49	91107	4	1.6	1	0	
1	2	45	19	34	90089	3	1.5	1	0	
2	3	39	15	11	94720	1	1.0	1	0	
3	4	35	9	100	94112	1	2.7	2	0	
4	5	35	8	45	91330	4	1.0	2	0	

```
df.shape
```

```
(5000, 14)
```

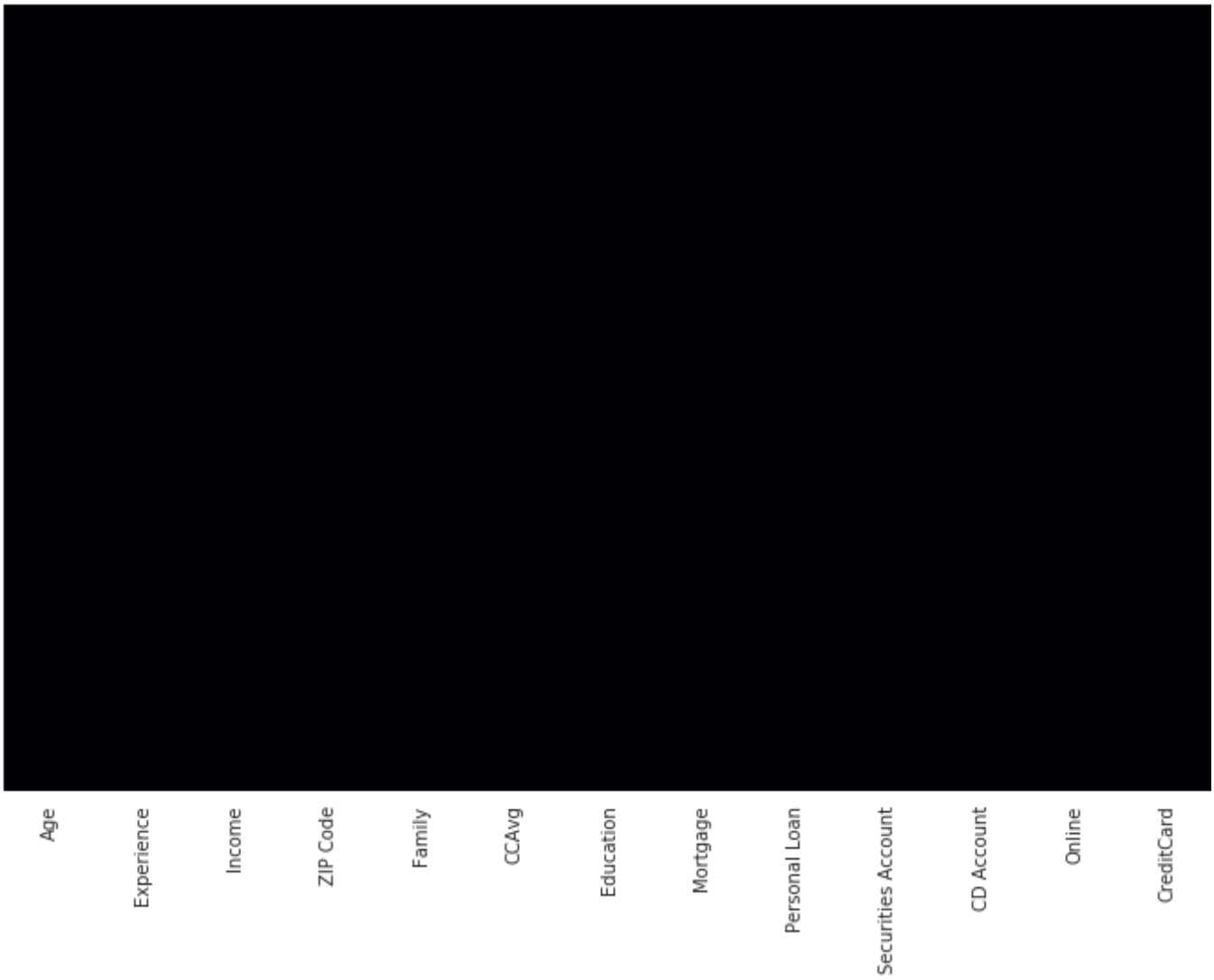
```
## ID column has nothing to do with the prediction so droping this column

df = df.drop('ID',axis=1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
Age                   5000 non-null int64
Experience            5000 non-null int64
Income                5000 non-null int64
ZIP Code              5000 non-null int64
Family                5000 non-null int64
CCAvg                 5000 non-null float64
Education             5000 non-null int64
Mortgage              5000 non-null int64
Personal Loan         5000 non-null int64
Securities Account    5000 non-null int64
CD Account            5000 non-null int64
Online                5000 non-null int64
CreditCard            5000 non-null int64
dtypes: float64(1), int64(12)
memory usage: 507.9 KB
```

```
## finding the null and missing values in the database
sns.heatmap(df.isnull(),xticklabels=True,yticklabels=False,cbar=False,cmap='inferno')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa7f8ba7080>
```




```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
Age	5000.0	45.338400	11.463166	23.0	35.0	45.0	55.0	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.0	20.0	30.0	43.0
Income	5000.0	73.774200	46.033729	8.0	39.0	64.0	98.0	224.0
ZIP Code	5000.0	93152.503000	2121.852197	9307.0	91911.0	93437.0	94608.0	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.0	2.0	3.0	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.7	1.5	2.5	10.0
Education	5000.0	1.881000	0.839869	1.0	1.0	2.0	3.0	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.0	0.0	101.0	635.0
Personal Loan	5000.0	0.096000	0.294621	0.0	0.0	0.0	0.0	1.0
Securities Account	5000.0	0.104400	0.305809	0.0	0.0	0.0	0.0	1.0
CD Account	5000.0	0.060400	0.238250	0.0	0.0	0.0	0.0	1.0
Online	5000.0	0.596800	0.490589	0.0	0.0	1.0	1.0	1.0

1. Age - min and max values lies within IQR , uppper and lower limit so we don't have any outlayers. Major custome majority of them are from working group and retired group(in majortiy).

2. Experience - min value is in negative which could not be possible so either we will make then zero or adjust with with experiences of min 1 to max 45 yrs.

3. Income - shows 75% at 98 while the max values is 224 , it is slightly right skewed.SO most customers we can sa

4. Zip code- its the representation of of the customer location

5. Family size - categorocal value - values are 1,2,3,4

6. CCAvg - almost symmetric distribution of data. Skewed towards right so majority of customers belong to group means their financial status is average and above.

7. Educatuion - categorocal value - 1: Undergrad; 2: Graduate; 3: Advanced/Professional

8. Mortgage - Moajority of customers dont't have mortgage value and only few have mortgage value equal and abc their own house

9. Personal Loan - categorocal value - 1 and 0 - Target column

10. Securities Account -categorocal value - 1 and 0

11. CD Account- categorocal value - 1 and 0

12. Online -categorocal value - 1 and 0

13. CreditCard-categorocal value - 1 and 0

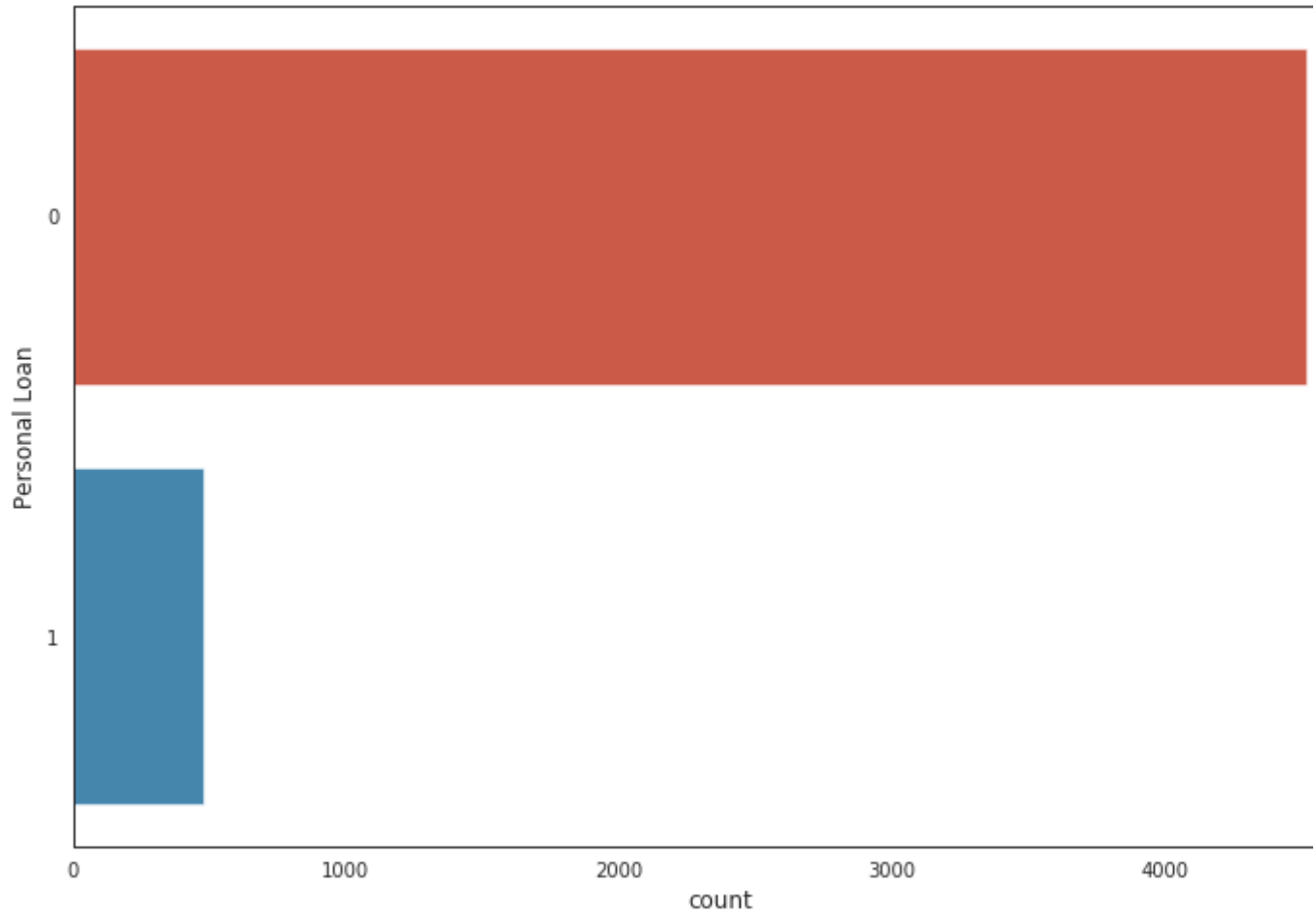
```
## now will see the distribution of data for each column
df['Personal Loan'].value_counts()
```

```
0    4520
1     480
Name: Personal Loan, dtype: int64
```

```
##  our target is an example of Class Imblanace where one class of customers whcih accepted
##  technique like smote or near miss to  remove this imbalance in the classes in our targte
```

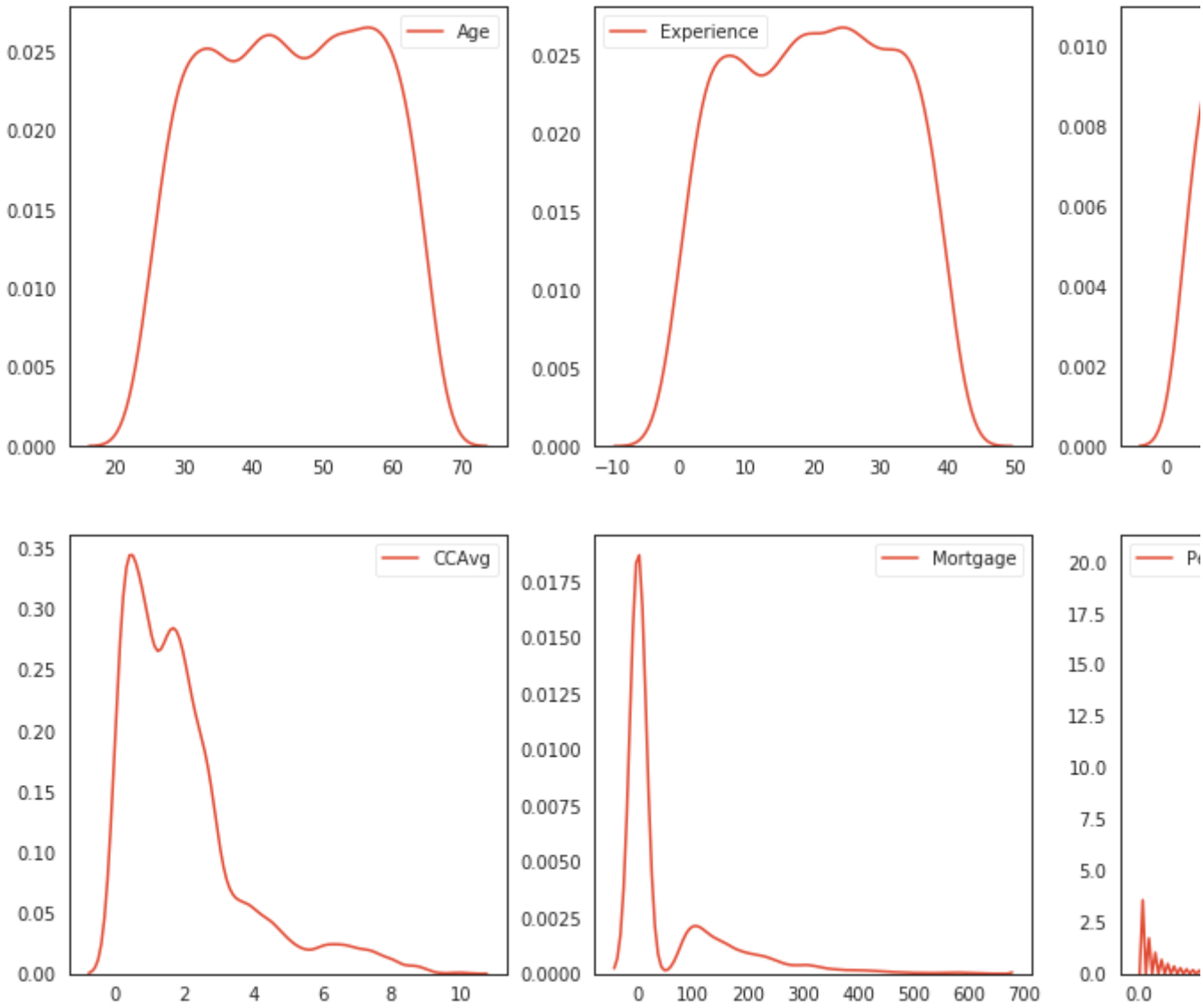
```
sns.countplot(y=df['Personal Loan'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa7f631cf98>
```




```
fig, qaxis = plt.subplots(2,3,figsize=(15,10))

sns.kdeplot(df['Age'], ax = qaxis[0,0])
#axis1.set_title('')

sns.kdeplot(df['Experience'], ax = qaxis[0,1])

sns.kdeplot(df['Income'], ax = qaxis[0,2])


sns.kdeplot(df['CCAvg'], ax = qaxis[1,0])


sns.kdeplot(df['Mortgage'], ax = qaxis[1,1])

sns.kdeplot(df['Personal Loan'], ax = qaxis[1,2])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa7f5ffbcc0>
```



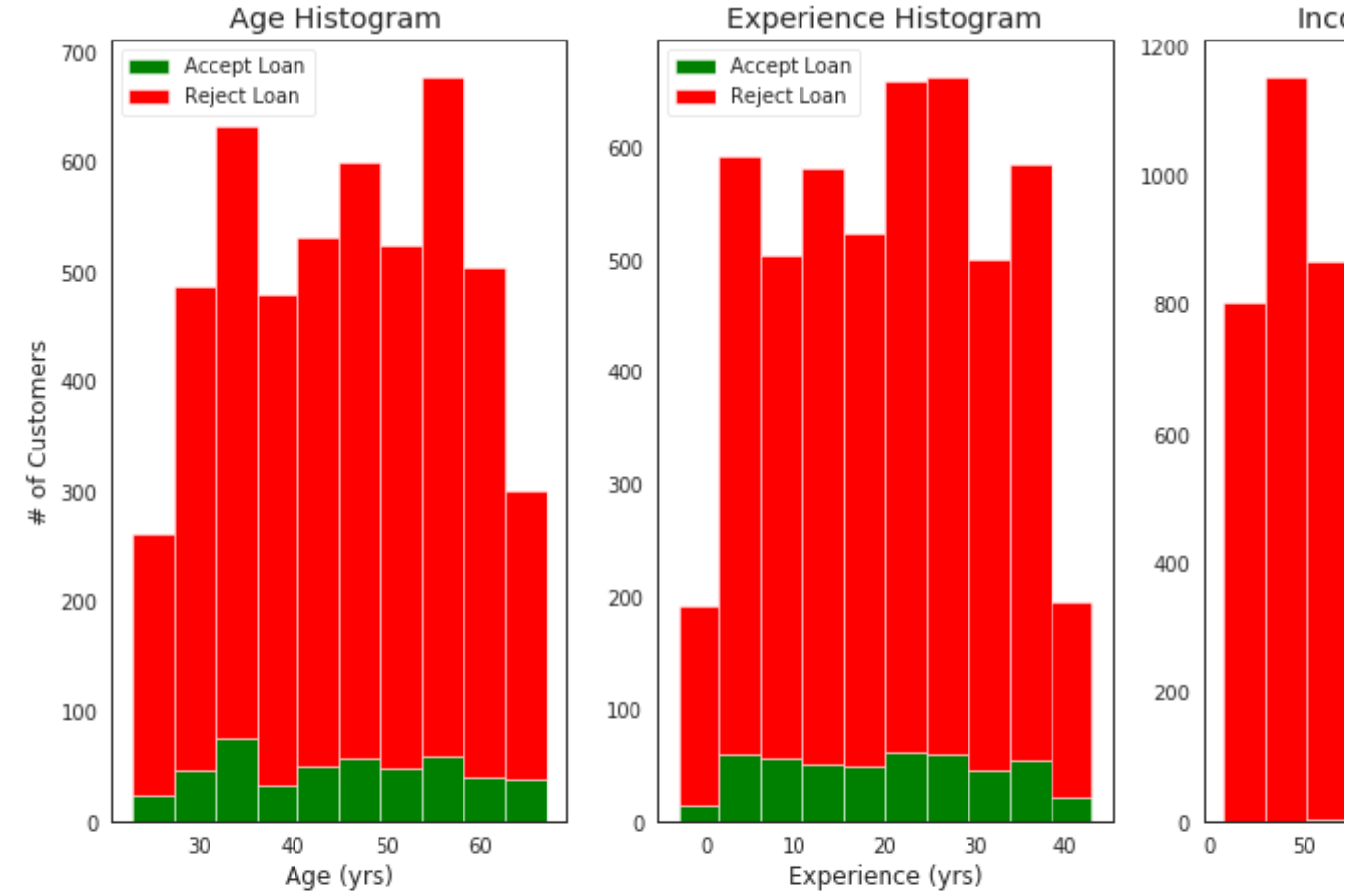
OBSERVATION

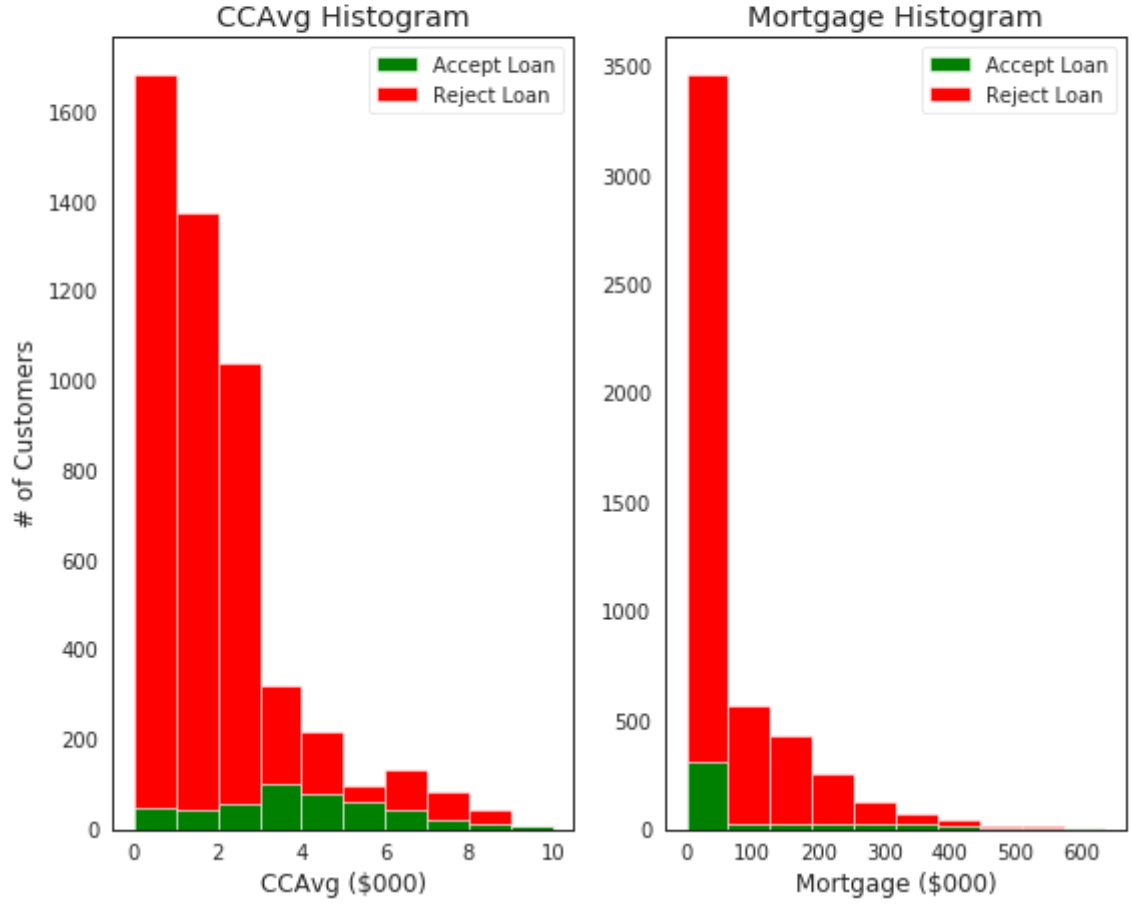
1. Age columns has three sub distribution indcating three age intervals
2. Experience has two sub clusters with second distribution more towrds higher education
3. Income is right skewed showing mid range and low ragnge income classes more
4. Avg. spending on credit cards per month ($000) is more towards lower range
5. *Value* of house mortgage is showing normal distribution with major outliers at higher range.

```
#graph distribution of quantitative data vs Personal Loan
plt.figure(figsize=[14,16])


plt.subplot(231)
plt.hist(x = [df[df['Personal Loan']==1]['Age'], df[df['Personal Loan']==0]['Age']],
         stacked=True, color = ['g','r'],label = ['Accept Loan','Reject Loan'])
plt.title('Age Histogram ')
plt.xlabel('Age (yrs)')
plt.ylabel('# of Customers')
```

```
plt.legend()

plt.subplot(232)
plt.hist(x = [df[df['Personal Loan']==1]['Experience'], df[df['Personal Loan']==0]['Experien
         stacked=True, color = ['g','r'],label = ['Accept Loan','Reject Loan'])
plt.title('Experience Histogram')
plt.xlabel('Experience (yrs)')
#plt.ylabel('# of Customers')
plt.legend()


plt.subplot(233)
plt.hist(x = [df[df['Personal Loan']==1]['Income'], df[df['Personal Loan']==0]['Income']],
         stacked=True, color = ['g','r'],label = ['Accept Loan','Reject Loan'])
plt.title('Income Histogram ')
plt.xlabel('Income ($000)')
#plt.ylabel('# of Customers')
plt.legend()


plt.subplot(234)
plt.hist(x = [df[df['Personal Loan']==1]['CCAvg'], df[df['Personal Loan']==0]['CCAvg']],
         stacked=True, color = ['g','r'],label = ['Accept Loan','Reject Loan'])
plt.title('CCAvg Histogram')
plt.xlabel('CCAvg ($000)')
plt.ylabel('# of Customers')
plt.legend()


plt.subplot(235)
plt.hist(x = [df[df['Personal Loan']==1]['Mortgage'], df[df['Personal Loan']==0]['Mortgage']
         stacked=True, color = ['g','r'],label = ['Accept Loan','Reject Loan'])
plt.title('Mortgage Histogram')
plt.xlabel('Mortgage ($000)')
#plt.ylabel('# of Customers')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa7f6243c88>
```





```
# AGE - there is the equal distrbution of Loan accepting customers in all class intervals ho
# in the 30 - 40 age interval
# Experience - there is the equal distrbution of Loan accepting customers in all class inter
# Income($000)- there is the assymetric distrbution of Loan accepting customers in all class
# Credi Card Average($000)- there is the equal distrbution of Loan accepting customers in al
# Mortgage - if we look fot the distrbution of Loan accepting customers we will find majorit
```

Double-click (or enter) to edit

```
df['AgeBin'] = pd.cut(df['Age'],7)
print(df['AgeBin'].unique())

grp = df.groupby('AgeBin')
print(grp['Age'].count().sum())
print(grp['Age'].count())
```

```
[(22.956, 29.286], (41.857, 48.143], (35.571, 41.857], (29.286, 35.571], (48.143,
Categories (7, interval[float64]): [(22.956, 29.286] < (29.286, 35.571] < (35.571
                                    (41.857, 48.143] < (48.143, 54.429] < (54.429
                                    (60.714, 67.0]]
5000
AgeBin
(22.956, 29.286]    488
(29.286, 35.571]    786
(35.571, 41.857]    722
(41.857, 48.143]    881
(48.143, 54.429]    782
(54.429, 60.714]    794
(60.714, 67.0]      547
Name: Age, dtype: int64
```

```
df['IncomeBin'] = pd.qcut(df['Income'],10)
print(df['IncomeBin'].unique())

grp = df.groupby('IncomeBin')
print(grp['Income'].count().sum())
print(grp['Income'].count())
```

```
[(42.0, 52.0], (33.0, 42.0], (7.999, 22.0], (88.3, 113.0], (22.0, 33.0], (64.0, 7
Categories (10, interval[float64]): [(7.999, 22.0] < (22.0, 33.0] < (33.0, 42.0]
                                     (78.0, 88.3] < (88.3, 113.0] < (113.0, 145.0
                                     (145.0, 224.0]]
5000
IncomeBin
(7.999, 22.0]     507
(22.0, 33.0]      522
(33.0, 42.0]      520
(42.0, 52.0]      453
(52.0, 64.0]      548
(64.0, 78.0]      469
(78.0, 88.3]      481
(88.3, 113.0]     521
(113.0, 145.0]    495
(145.0, 224.0]    484
Name: Income, dtype: int64
```

```
df['ExperienceBin'] = pd.cut(df['Experience'],6)
print(df['ExperienceBin'].unique())

grp = df.groupby('ExperienceBin')
print(grp['Experience'].count().sum())
print(grp['Experience'].count())
```

```
[(-3.046, 4.667], (12.333, 20.0], (4.667, 12.333], (20.0, 27.667], (35.333, 43.0]
Categories (6, interval[float64]): [(-3.046, 4.667] < (4.667, 12.333] < (12.333,
                                    (20.0, 27.667] < (27.667, 35.333] < (35.333,
5000
ExperienceBin
(-3.046, 4.667]      519
(4.667, 12.333]      988
(12.333, 20.0]      1035
(20.0, 27.667]       913
(27.667, 35.333]    1031
(35.333, 43.0]       514
Name: Experience, dtype: int64
```

```
#code categorical data
label = LabelEncoder()
df['AgeBin_Code'] = label.fit_transform(df['AgeBin'])
df['ExperienceBin_Code'] = label.fit_transform(df['ExperienceBin'])
df['IncomeBin_Code'] = label.fit_transform(df['IncomeBin'])
```

```
data_train_dummy = pd.get_dummies(data = df,columns=['Family','Education','Securities Accoun
data_train_dummy.sample(2)
```

	Age	Experience	Income	ZIP Code	CCAvg	Mortgage	Personal Loan	AgeBin	IncomeBin
4506	39	13	89	92037	2.8	153	0	(35.571, 41.857]	(88.3, 113.0]
2380	40	16	50	92606	0.6	0	0	(35.571, 41.857]	(42.0, 52.0]

```
data_train_dummy = data_train_dummy.drop(columns=['Age','Experience','ZIP Code','Income','Ag
data_train_dummy.sample(2)
```

	CCAvg	Mortgage	Personal Loan	AgeBin_Code	ExperienceBin_Code	IncomeBin_Code
1186	1.2	0	0	6	5	3
1939	0.2	0	0	5	4	1

```
data_train_dummy.drop(columns=['Personal Loan'],axis=1,inplace=True)
```

```
X = np.array(data_train_dummy)
Y=np.array(df['Personal Loan'])
```

```
MLA = [

            #Gaussian Processes
            gaussian_process.GaussianProcessClassifier(),

            #GLM
            linear_model.LogisticRegressionCV(),


            #Navies Bayes
            naive_bayes.BernoulliNB(),
            naive_bayes.GaussianNB(),

            #Nearest Neighbor
            neighbors.KNeighborsClassifier()


    ]


MLA_columns = ['MLA_Name']
MLA_compare = pd.DataFrame(columns = MLA_columns)


train1_x, test1_x, train1_y, test1_y = model_selection.train_test_split(X, Y, test_size=0.3,
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
smt = SMOTE()
X_train_sm, y_train_sm = smt.fit_sample(train1_x,train1_y)


MLA_predict = pd.DataFrame({"Actual_Data_Target":test1_y})
MLA_predict.head(2)
```

	Actual_Data_Target
0	1
1	0

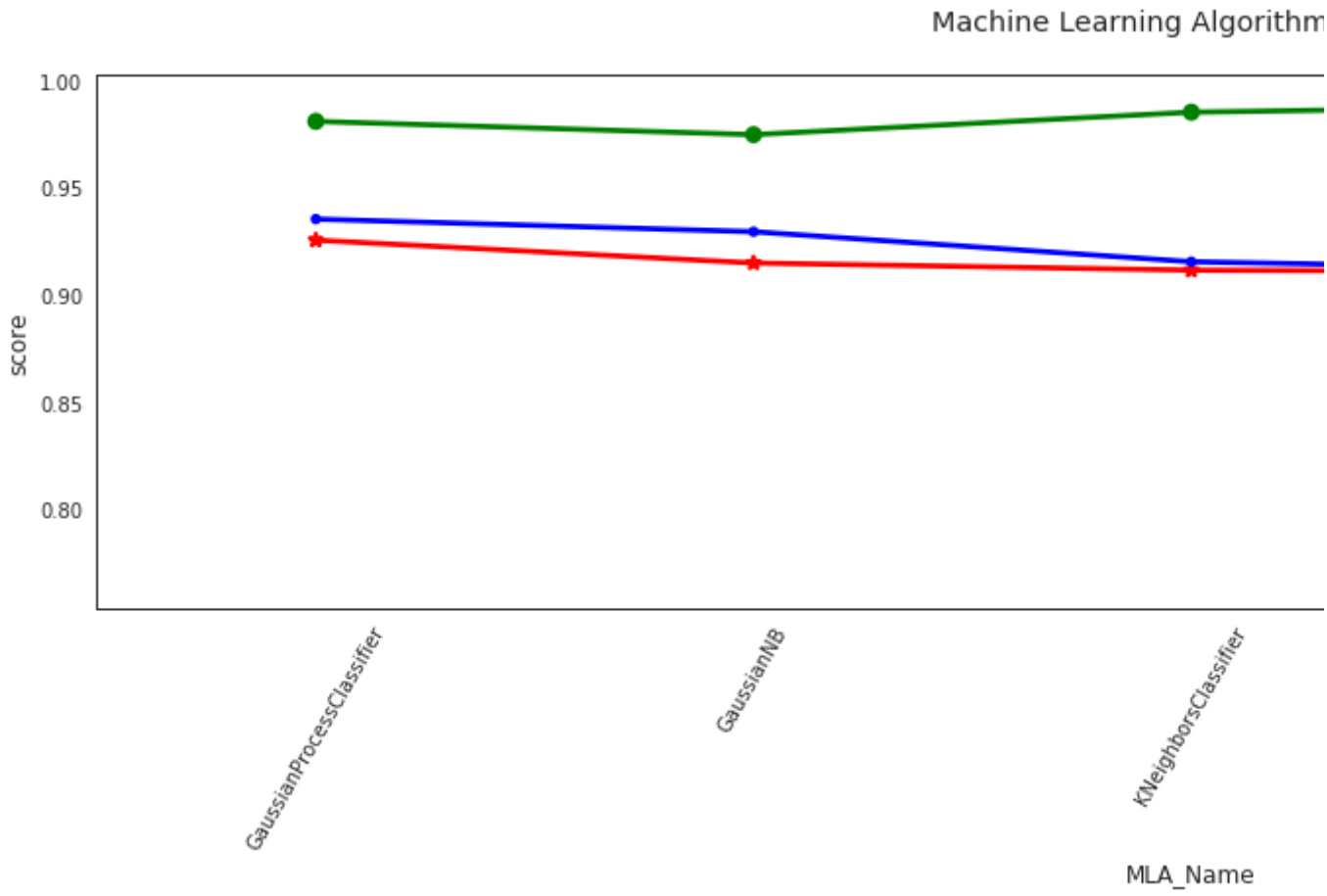
```
row_index = 0
for alg in MLA:


    #set name and parameters
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index, 'MLA_Name'] = MLA_name
    #MLA_compare.loc[row_index, 'MLA_Parameters'] = str(alg.get_params())

    d = alg.fit(X_train_sm, y_train_sm)
    MLA_predict[MLA_name] = alg.predict(test1_x)
    #MLA_compare.loc[row_index, 'MLA_Intercept'] =  alg.intercept_
    MLA_compare.loc[row_index, 'Actual_Y_Shape'] =test1_y.shape
    MLA_compare.loc[row_index, 'confusion_matrix'] = str(metrics.confusion_matrix(test1_y,al
    MLA_compare.loc[row_index, 'accuracy_score'] = metrics.accuracy_score(test1_y,alg.predic
    confusion_matrix = metrics.confusion_matrix(test1_y,alg.predict(test1_x))
    MLA_compare.loc[row_index, 'True_Positive_Rate\n[Sensitivity/Recall]'] = confusion_matri
    MLA_compare.loc[row_index, 'Precision'] = confusion_matrix[0][0]/(confusion_matrix[0][0]
    MLA_compare.loc[row_index, 'True_Negatve_Rate\n[Specificity]'] = confusion_matrix[1][1]/
    MLA_compare.loc[row_index, 'mean_absolute_error'] = metrics.mean_absolute_error(test1_y,
    MLA_compare.loc[row_index, 'mean_squared_error'] = metrics.mean_squared_error(test1_y, a
    MLA_compare.loc[row_index, 'root_mean_squared_error'] = np.sqrt(metrics.mean_squared_err
```

```
    #MLA_compare.loc[row_index, 'explained_variance_score'] = explained_variance_score(test1


    row_index+=1

MLA_compare.sort_values(by = ['accuracy_score','Precision'], ascending = False, inplace = Tr
MLA_compare
```

	MLA_Name	Actual_Y_Shape	confusion_matrix	accuracy_score	True_ [Sensi
0	GaussianProcessClassifier	1500.0	[[1269 87]\n [24 120]]	0.926000	
3	GaussianNB	1500.0	[[1261 95]\n [32 112]]	0.915333	
4	KNeighborsClassifier	1500.0	[[1242 114]\n [18 126]]	0.912000	
1	LogisticRegressionCV	1500.0	[[1237 119]\n [14 130]]	0.911333	
2	BernoulliNB	1500.0	[[1088 268]\n [81 63]]	0.767333	

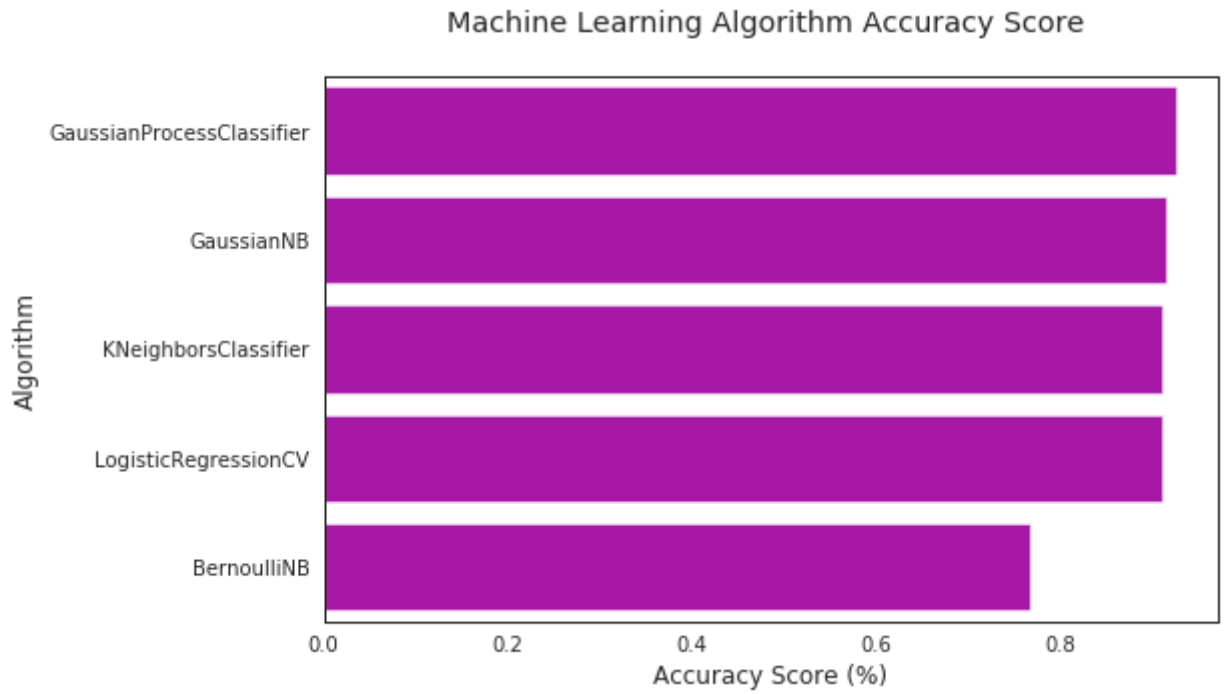
```
plt.figure(figsize=(20,5))
sns.pointplot(y='accuracy_score', x ='MLA_Name', data = MLA_compare , color = 'r',markers=["
sns.pointplot(y='Precision', x ='MLA_Name', data = MLA_compare , color = 'g',markers=["o"],l
sns.pointplot(y='True_Positive_Rate\n[Sensitivity/Recall]', x ='MLA_Name', data = MLA_compar
plt.title('Machine Learning Algorithm Results \n')
plt.xlabel('MLA_Name')
plt.ylabel('score')
plt.xticks(rotation=60)
plt.legend(loc='bottom left', frameon=False)
```

```
No handles with labels found to put in legend.
<matplotlib.legend.Legend at 0x7fa7f2dcdcf8>
```




```
plt.figure(figsize=(8,5))

sns.barplot(x='accuracy_score', y ='MLA_Name', data = MLA_compare , color = 'm')
#prettify using pyplot: https://matplotlib.org/api/pyplot_api.html
plt.title('Machine Learning Algorithm Accuracy Score \n')
plt.xlabel('Accuracy Score (%)')
plt.ylabel('Algorithm')
```

```
Text(0, 0.5, 'Algorithm')
```



Conclusion

the model will be good which have high accuracy score, precision score and recall value and minir

so GaussianProcessClassifier is the best classification algorithm