Project Title: Marketing Campaign for Banking Products

Bank is has a growing customer base. The bank wants to increase borrowers (asset customers) base to bring in more loan business and earn more through the interest on loans. So , bank wants to convert the liability based customers to personal loan customers. (while retaining them as depositors).The department wants to build a model that will help them identify the potential customers who have higher probability of purchasing the loan. This will increase the success ratio while at the same time reduce the cost of the campaign.

The dataset contains data on 5000 customers

The case is The Bank has a customers Data with various characteristics of the customers. The management built a new product - Personal Loan, and ran a small campaign towards selling the New Product to their clients. After some time, 9% of customers have Personal Loan from The Bank.

The GOAL IS!

- To sell more Personal Loan products to Bank customers.
- To devise campaigns to better target marketing to increase the success ratio with a minimal budget.
- To identify the potential customers who have a higher probability of purchasing the loan.

Increase the success ratio of advertisement campaign while at the same time reduce the cost of the campaign.

1. Import the datasets and libraries, check datatype, statistical summary, shape, null values etc

1.1 Importing the required Libraries and the dataset

In [2]:

```
#Importing libraries
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Exploratory Data Analysis

I have use Bank Loan Modeling dataset.

Dataset is an excel file and it has 2 excel sheets, Data and Description.

Pandas **read_excel** function is used to read Data sheet.

In [93]:

```
#Importing the datasets
df = pd.read_excel("Bank_Personal_Loan_Modelling.xlsx", sheet_name='Data')
```

In [4]:

```
#To check highest value and lowest values
df.head(10).style.background_gradient(cmap="PuBuGn")
```

Out[4]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Sec A
0	1	25	1	49	91107	4	1.600000	1	0	0	
1	2	45	19	34	90089	3	1.500000	1	0	0	
2	3	39	15	11	94720	1	1.000000	1	0	0	
3	4	35	9	100	94112	1	2.700000	2	0	0	
4	5	35	8	45	91330	4	1.000000	2	0	0	
5	6	37	13	29	92121	4	0.400000	2	155	0	
6	7	53	27	72	91711	2	1.500000	2	0	0	
7	8	50	24	22	93943	1	0.300000	3	0	0	
8	9	35	10	81	90089	3	0.600000	2	104	0	
9	10	34	9	180	93023	1	8.900000	3	0	1	

There are **12** features.
The aim is to construct a model that can identify potential customers who have a higher probability of purchasing loan. Output column is Personal Loan.

Features are detailed below:

Age Customer's age
Experience Number of years of professional experience
Income Annual income of the customer
ZIPCode Home Address ZIP code
Family Family size of the customer
CCAvg Average spending on credit cards per month
Education Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional
Mortgage Value of house mortgage if any
Securities Account Does the customer have a securities account with the bank?
CD Account Does the customer have a certificate of deposit (CD) account with the bank?
Online Does the customer use internet banking facilities?
CreditCard Does the customer uses a credit card issued by UniversalBank?
Personal Loan Did this customer accept the personal loan offered in the last campaign?

In [5]:

```
#To display top 5 rows
df.head()
```

Out[5]:

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

In [6]:

```
#To display bottom 5 rows
df.tail()
```

Out[6]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan
4995	4996	29	3	40	92697	1	1.9	3	0	0
4996	4997	30	4	15	92037	4	0.4	1	85	0
4997	4998	63	39	24	93023	2	0.3	3	0	0
4998	4999	65	40	49	90034	3	0.5	2	0	0
4999	5000	28	4	83	92612	3	0.8	1	0	0

1.2 Check the types of the Data

In [7]:

```
# To find the dtypes in the DataFrame of each columns
df.dtypes
```

Out[7]:

```
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
```

1.3 Check Statistical Summary

In [8]:

```
# To view some basic statistical details.
df.describe()
```

Out[8]:

	ID	Age	Experience	Income	ZIP Code	Family	C(
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.0(
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	1.93
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	1.74
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.0(
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.7(
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.5(
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.5(
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.0(

In [9]:

```
# Transpose of df.describe()
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750.25	5000.0
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55.00	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30.00	43.0
Income	5000.0	73.774200	46.033729	8.0	39.00	64.0	98.00	224.0
ZIP Code	5000.0	93152.503000	2121.852197	9307.0	91911.00	93437.0	94608.00	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	4.0
CCAvg	5000.0	1.937913	1.747666	0.0	0.70	1.5	2.50	10.0
Education	5000.0	1.881000	0.839869	1.0	1.00	2.0	3.00	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101.00	635.0
Personal Loan	5000.0	0.096000	0.294621	0.0	0.00	0.0	0.00	1.0
Securities Account	5000.0	0.104400	0.305809	0.0	0.00	0.0	0.00	1.0
CD Account	5000.0	0.060400	0.238250	0.0	0.00	0.0	0.00	1.0
Online	5000.0	0.596800	0.490589	0.0	0.00	1.0	1.00	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1.00	1.0

We can observe that Experience has some negative values

1.4 Check the shape of dataframe and null values

In [94]:

```
# To check the Dimensionality of the DataFrame
df.shape
```

Out[94]:

```
(5000, 14)
```

In [11]:

```
# To check the total null values
df.isnull().sum()
```

Out[11]:

```
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

2.1 Dropping Irrelavant column

```
The variable ID does not add any interesting information. There is no association between
a person's customer ID and loan, also it does not provide any general conclusion for
future potential loan customers. We can neglect this information for our model
prediction.
```

In [95]:

```
# To check the counts of negative values in experience column
df[df['Experience'] < 0]['Experience'].count()
```

Out[95]:

```
52
```

In [96]:

```
#To check the ammount of negative values
df[df['Experience'] < 0]['Experience'].value_counts()
```

Out[96]:

```
-1    33
-2    15
-3     4
Name: Experience, dtype: int64
```

Since Experience Column and Age are highly correlated. Drop Experience Column

In [97]:

```
# Dropping the ID and Experience column
df.drop(['ID','Experience'],axis=1,inplace=True)
```

In [98]:

```
#To display top 5 rows
df.head()
```

Out[98]:

	Age	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	(
0	25	49	91107	4	1.6	1	0	0	1	0	
1	45	34	90089	3	1.5	1	0	0	1	0	
2	39	11	94720	1	1.0	1	0	0	0	0	
3	35	100	94112	1	2.7	2	0	0	0	0	
4	35	45	91330	4	1.0	2	0	0	0	0	

In [99]:

```
#To display bottom 5 rows
df.tail()
```

Out[99]:

	Age	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CI Accoun
4995	29	40	92697	1	1.9	3	0	0	0	(
4996	30	15	92037	4	0.4	1	85	0	0	(
4997	63	24	93023	2	0.3	3	0	0	0	(
4998	65	49	90034	3	0.5	2	0	0	0	(
4999	28	83	92612	3	0.8	1	0	0	0	(

In [16]:

```
#To check the names of each column
df.columns
```

Out[16]:

```
Index(['Age', 'Income', 'ZIP Code', 'Family', 'CCAvg', 'Education', 'Mortgag
e',
       'Personal Loan', 'Securities Account', 'CD Account', 'Online',
       'CreditCard'],
      dtype='object')
```

3. EDA

In [17]:

```
#To check number of unique elements in each columns
df.nunique()
```

Out[17]:

```
Age                    45
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
```

Zip Code has 467 distinct value. It is nominal variable. It will not affect the prediction. So we will drop zip code column

In [100]:

```
#Drop the Zip Code Column
df.drop(['ZIP Code'], axis = 1)
```

Out[100]:

	Age	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Onlin
0	25	49	4	1.6	1	0	0	1	0	
1	45	34	3	1.5	1	0	0	1	0	
2	39	11	1	1.0	1	0	0	0	0	
3	35	100	1	2.7	2	0	0	0	0	
4	35	45	4	1.0	2	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
4995	29	40	1	1.9	3	0	0	0	0	
4996	30	15	4	0.4	1	85	0	0	0	
4997	63	24	2	0.3	3	0	0	0	0	
4998	65	49	3	0.5	2	0	0	0	0	
4999	28	83	3	0.8	1	0	0	0	0	

5000 rows × 11 columns

In [101]:

```
# Number of people with zero mortgage
df[df['Mortgage'] == 0]['Mortgage'].value_counts()
```

Out[101]:

```
0    3462
Name: Mortgage, dtype: int64
```

In [20]:

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

Out[20]:

```
0.0    106
Name: CCAvg, dtype: int64
```

In [21]:

```
# Value counts of Family column
df['Family'].value_counts()
```

Out[21]:

```
1    1472
2    1296
4    1222
3    1010
Name: Family, dtype: int64
```

In [22]:

```
# Value counts of Securities Account column
df['Securities Account'].value_counts()
```

Out[22]:

```
0    4478
1     522
Name: Securities Account, dtype: int64
```

In [23]:

```
# Value counts of CD Account column
df['CD Account'].value_counts()
```

Out[23]:

```
0    4698
1     302
Name: CD Account, dtype: int64
```

In [24]:

```
# Value counts of CreditCard column
df['CreditCard'].value_counts()
```

Out[24]:

```
0    3530
1    1470
Name: CreditCard, dtype: int64
```

In [25]:

```
# Value counts of Education column
df['Education'].value_counts()
```

Out[25]:

```
1    2096
3    1501
2    1403
Name: Education, dtype: int64
```

In [26]:

```
# Value counts of Online column
df['Online'].value_counts()
```

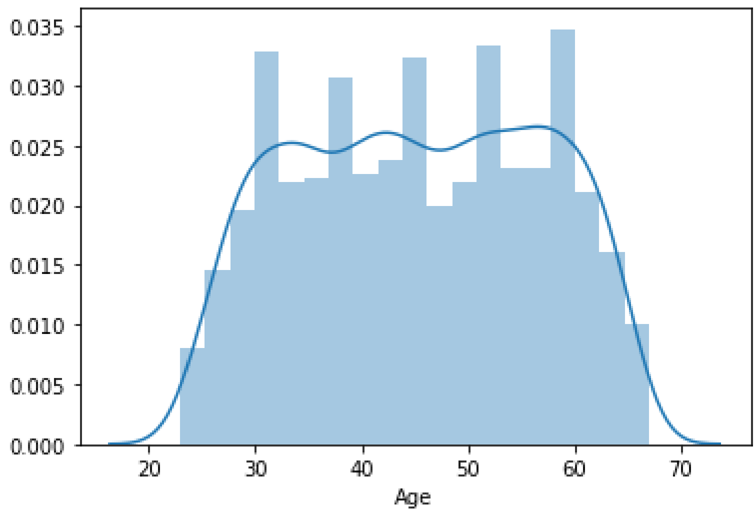
Out[26]:

```
1    2984
0    2016
Name: Online, dtype: int64
```

3.1 Univariate Analysis

In [27]:

```
# Age have normal distributions
sns.distplot(df["Age"])
plt.show()
```

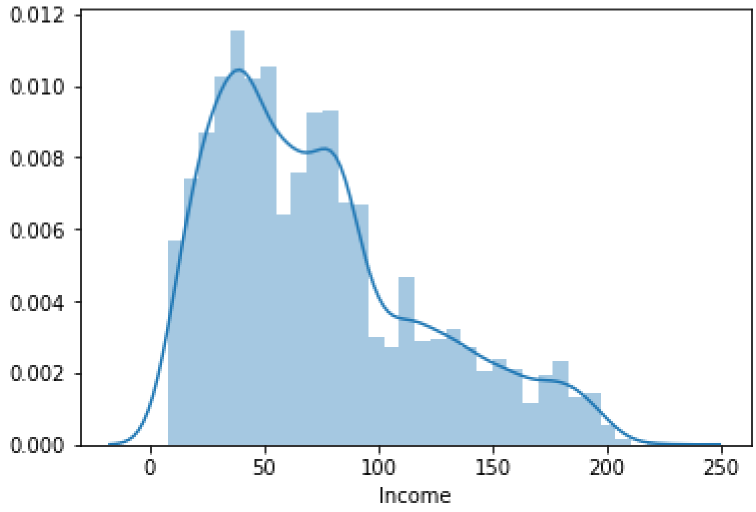


In [28]:

```
# Income is right skewed distributions
sns.distplot(df["Income"])
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286f18ef2c8>
```

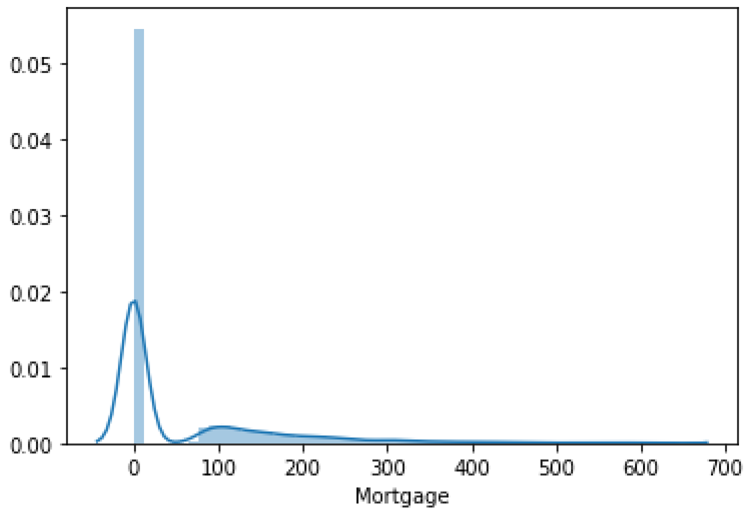


In [29]:

```
# Mortgage seems to be highly skewed
sns.distplot(df["Mortgage"])
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286f1a01a48>
```

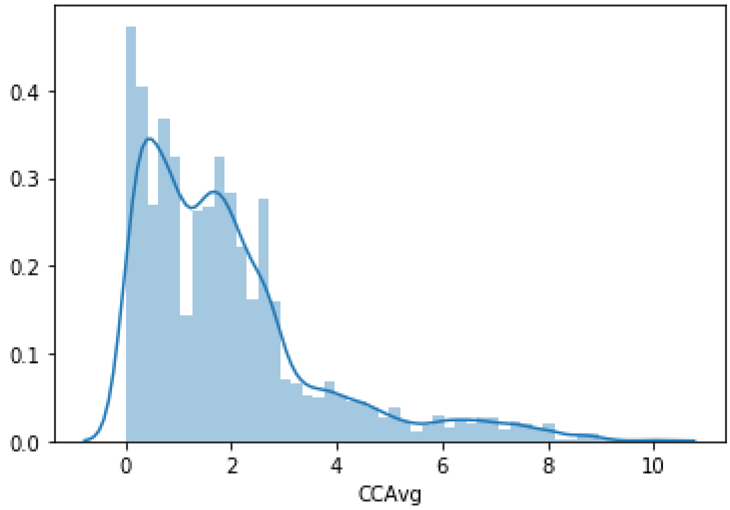


In [30]:

```
# Credit Card Average is right skewed distributions
sns.distplot(df["CCAvg"])
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286f1b0c488>
```



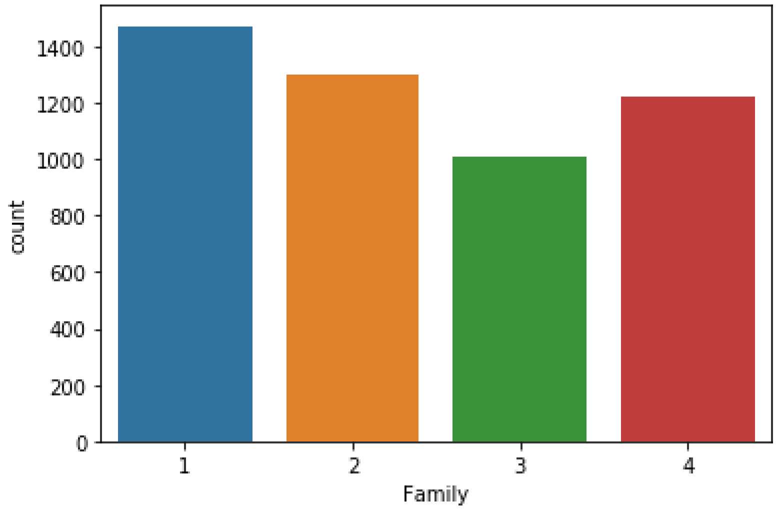
We have to do some feature engineering on Income, CCAVg and Mortgage variables. Because if we use skewed then it will create fault in logistic regression.

In [31]:

```
# Count Plot to show Family Distributions
sns.countplot(x='Family',data=df)
```

Out[31]:

<matplotlib.axes._subplots.AxesSubplot at 0x286f1becc08>

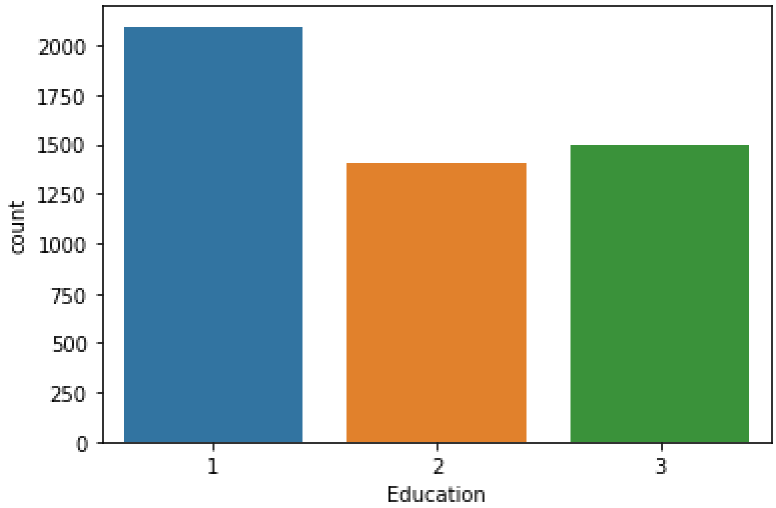


In [102]:

```
# Count Plot to show Education Distributions
sns.countplot(x='Education',data=df)
```

Out[102]:

<matplotlib.axes._subplots.AxesSubplot at 0x286fca3cbc8>

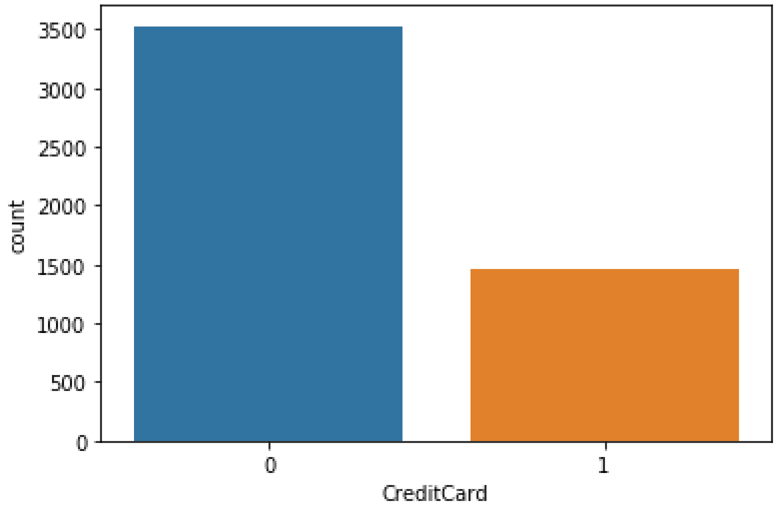


In [103]:

```
# Count Plot to show CreditCard Distributions
sns.countplot(x='CreditCard',data=df)
```

Out[103]:

<matplotlib.axes._subplots.AxesSubplot at 0x286fca3c248>

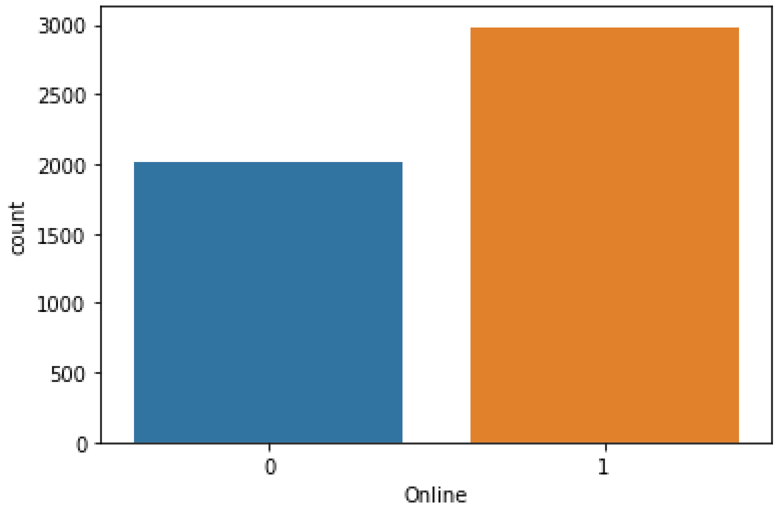


In [104]:

```
#  Count Plot to show Online Distributions
sns.countplot(x='Online',data=df)
```

Out[104]:

<matplotlib.axes._subplots.AxesSubplot at 0x286fc986948>



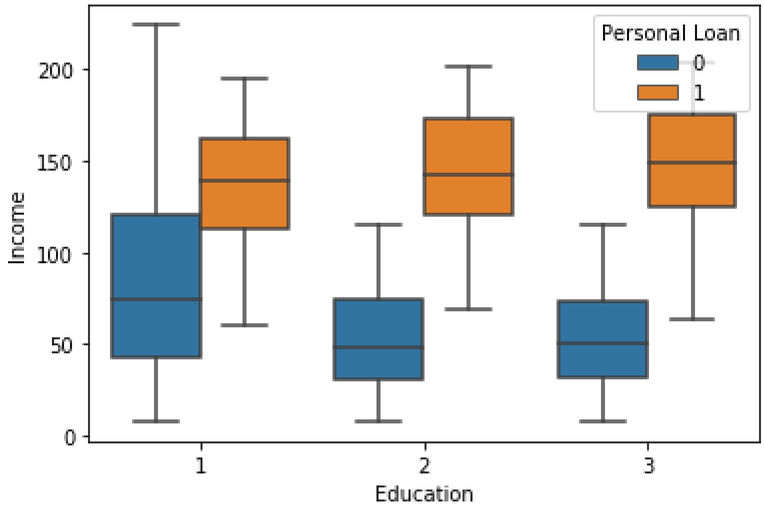
3.2 Multivariate Analysis

In [33]:

```
# Influence of income and education on personal loan
sns.boxplot(x='Education',y='Income',hue='Personal Loan',data=df)
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286f1cb1ac8>
```



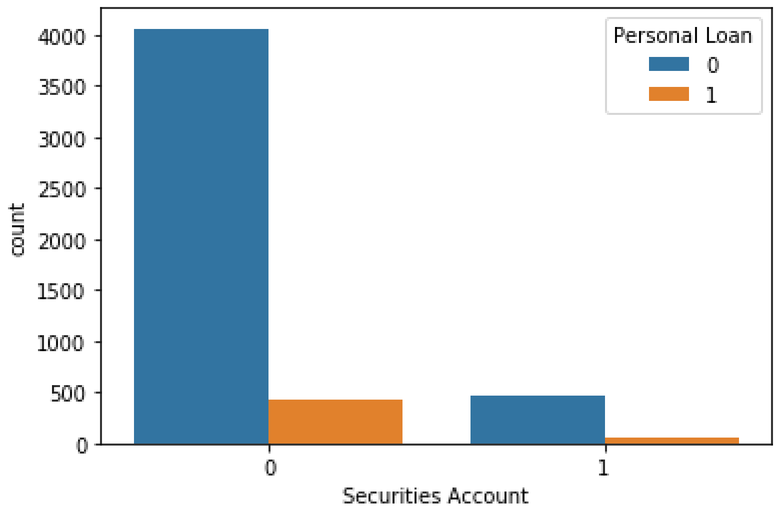
Observation : It seems the customers whose education level is 1 is having more income. However customers who has taken the personal loan have the same income levels

In [105]:

```
sns.countplot(x="Securities Account", data=df,hue="Personal Loan")
```

Out[105]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286fca5eec8>
```



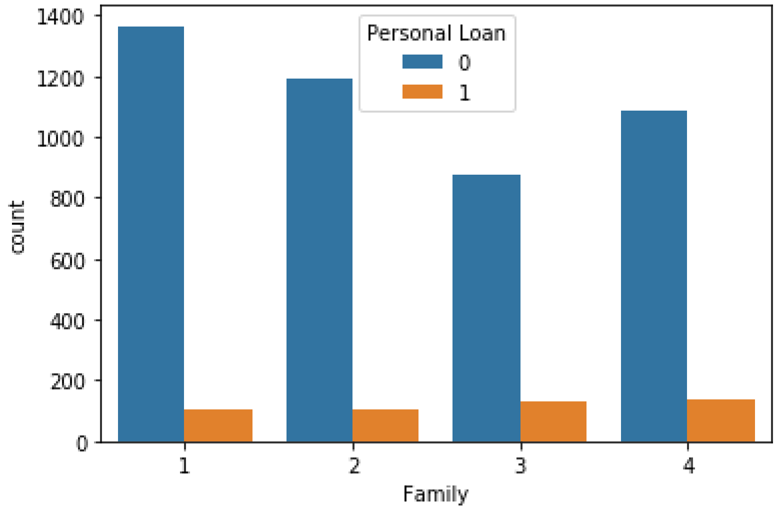
Observation : Majority of customers who does not have loan have securities account

In [108]:

```
sns.countplot(x='Family',data=df,hue='Personal Loan')
```

Out[108]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286fc9fe248>
```



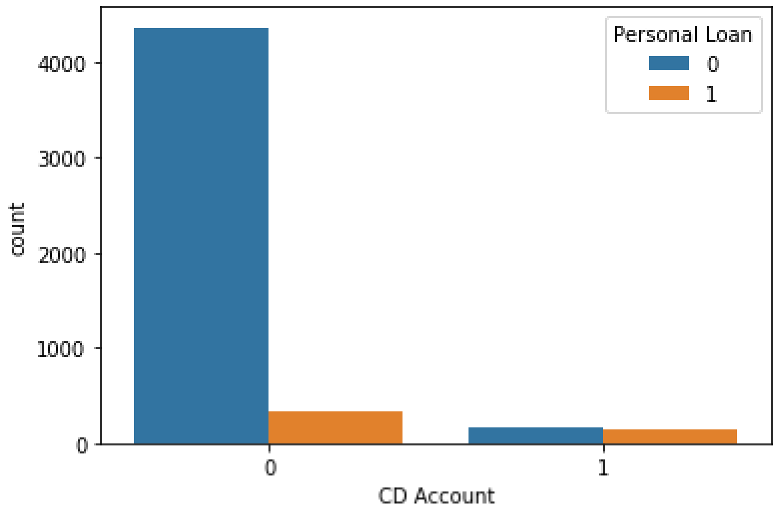
Observation: Family size does not have any impact in personal loan. But it seems families with size of 3 are more likely to take loan. When considering future campaign this might be good association.

In [110]:

```
sns.countplot(x='CD Account',data=df,hue='Personal Loan')
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286fcba9708>
```



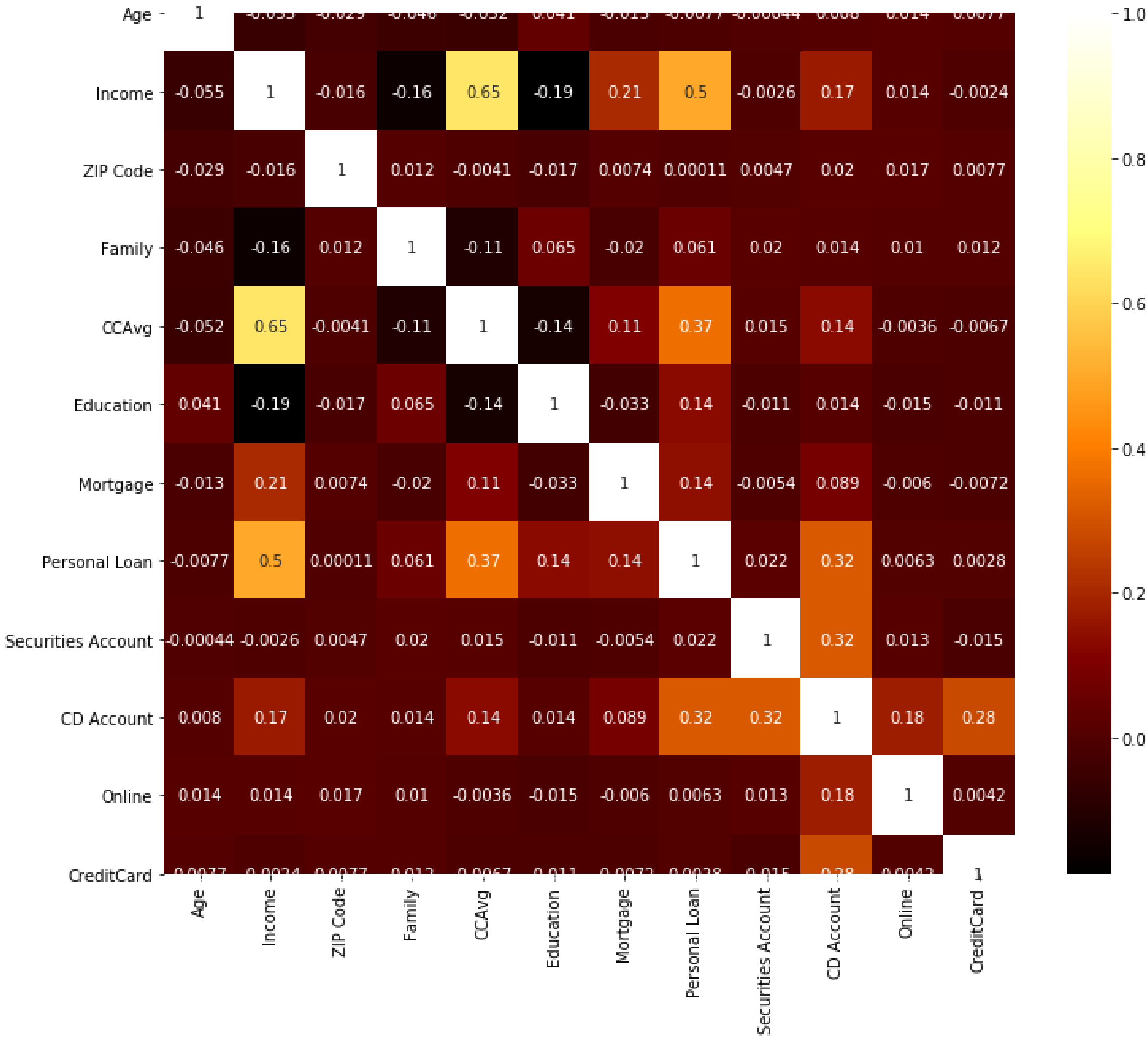
Observation: Customers who does not have CD account , does not have loan as well. This seems to be majority. But almost all customers who has CD account has loan as well

In [35]:

```
# CCAvg Credit average and income are highly correlated
fig, ax = plt.subplots(figsize=(12,10))
sns.heatmap(df.corr(), cmap='afmhot' , annot = True)
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x286f18a1048>
```

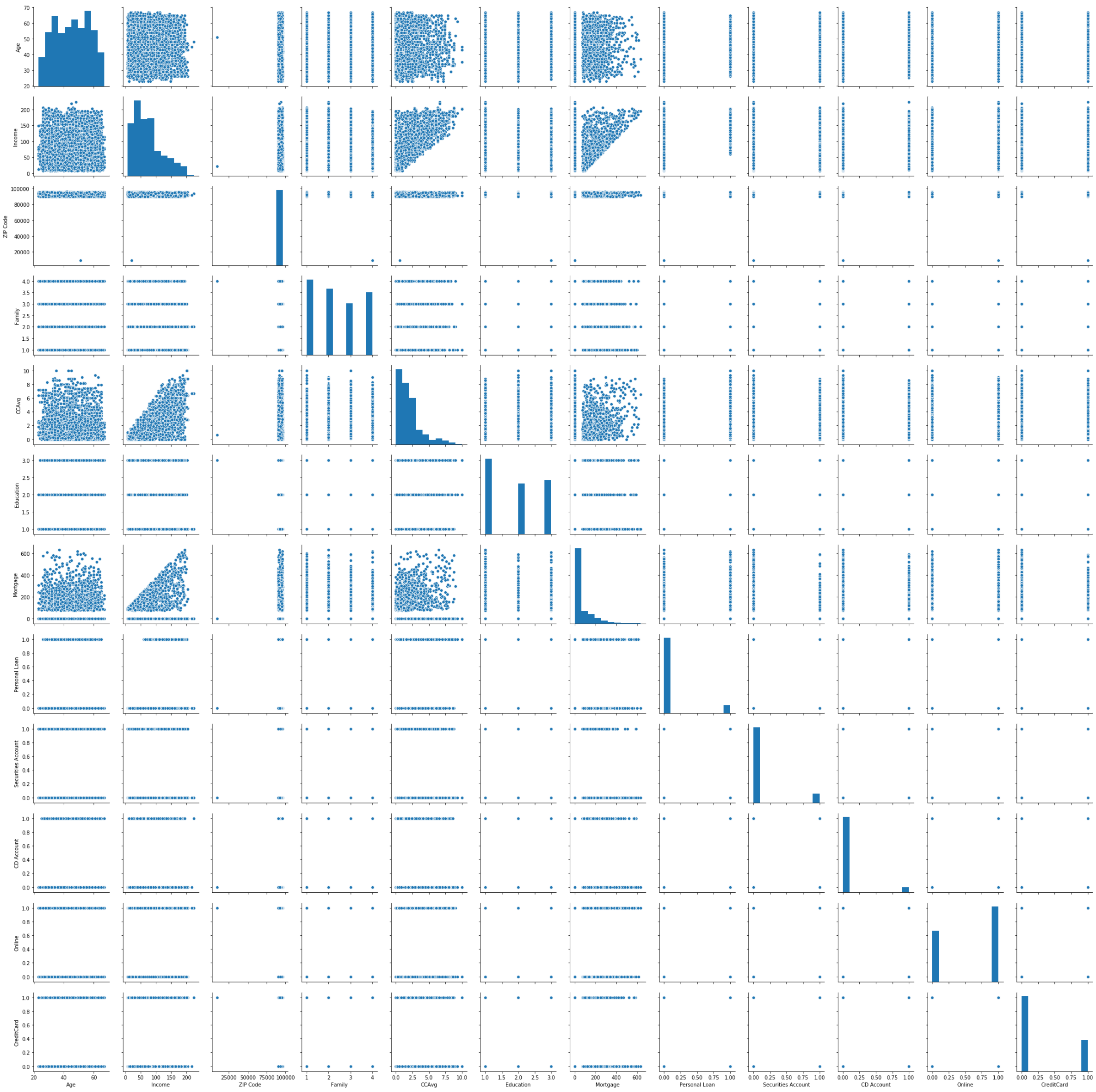


In [36]:

```
sns.pairplot(df)
```

Out[36]:

```
<seaborn.axisgrid.PairGrid at 0x286eff4a208>
```



4. Apply necessary transformations for the feature variables

In [37]:

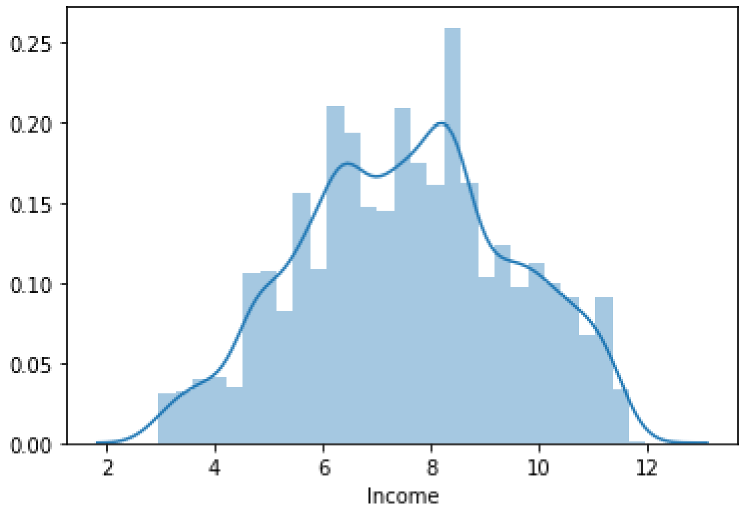
```
data_X = df.loc[:, df.columns  != 'Personal Loan']
data_Y = df[['Personal Loan']]
```

In [38]:

```
# Applying the Yeo Johnson method of Transformation on the Income variable.
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method='yeo-johnson',standardize=False)
pt.fit(data_X['Income'].values.reshape(-1,1))
temp = pt.transform(data_X['Income'].values.reshape(-1,1))
data_X['Income'] = pd.Series(temp.flatten())
```

In [39]:

```
# Distplot to show transformed Income variable
sns.distplot(data_X['Income'])
plt.show()
```

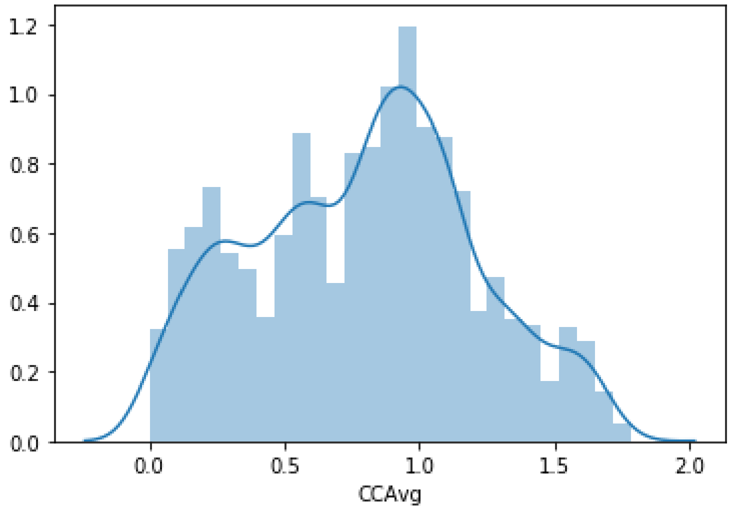


In [40]:

```
# Applying the Yeo Johnson method of Transformation on the CCAvg variable.
pt = PowerTransformer(method='yeo-johnson',standardize=False)
pt.fit(data_X['CCAvg'].values.reshape(-1,1))
temp = pt.transform(data_X['CCAvg'].values.reshape(-1,1))
data_X['CCAvg'] = pd.Series(temp.flatten())
```

In [41]:

```
# Distplot to show transformed CCAvg variable
sns.distplot(data_X['CCAvg'])
plt.show()
```

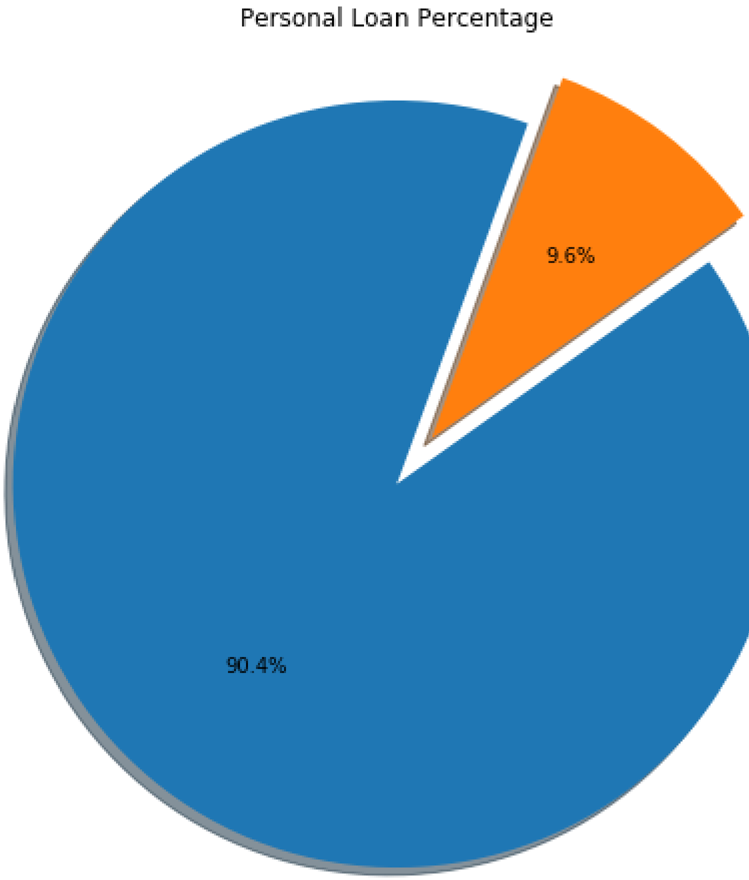


In [42]:

```
# Binning on Mortgage variable.
data_X['Mortgage_Int'] = pd.cut(data_X['Mortgage'],
                                bins=[0,100,200,300,400,500,600,700],
                                labels= [0,1,2,3,4,5,6],
                                include_lowest =True)
data_X.drop('Mortgage', axis = 1, inplace= True)
```

In [90]:

```
## Univariate Analysis
## 9.6% of all the applicants get approved for personal loan
tempDF = pd.DataFrame(df['Personal Loan'].value_counts()).reset_index()
tempDF.columns = ['Labels', 'Personal Loan']
fig1, ax1 = plt.subplots(figsize=(10,8))
explode = (0, 0.15)
ax1.pie(tempDF['Personal Loan'] , explode= explode, autopct= '%1.1f%%',
       shadow=True , startangle = 70)
ax1.axis('equal')
plt.title('Personal Loan Percentage')
plt.show()
```



In [43]:

```
# To display top 5 rows
data_X.head()
```

Out[43]:

	Age	Income	ZIP Code	Family	CCAvg	Education	Securities Account	CD Account	Online	CreditCard
0	25	6.827583	91107	4	0.845150	1	1	0	0	0
1	45	5.876952	90089	3	0.814468	1	1	0	0	0
2	39	3.504287	94720	1	0.633771	1	0	0	0	0
3	35	8.983393	94112	1	1.107409	2	0	0	0	0
4	35	6.597314	91330	4	0.633771	2	0	0	0	1

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

In [44]:

```
# Importing required libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [45]:

```
# Splitting the data into train and test. We use stratify parameter of train_test_split fun
X_train,X_test,Y_train,Y_test = train_test_split(data_X,data_Y,test_size = 0.3, random_stat
```

In [46]:

```
X_train.reset_index(drop= True, inplace= True);
X_test.reset_index(drop= True, inplace= True);
Y_train.reset_index(drop= True, inplace= True);
Y_test.reset_index(drop= True, inplace= True);
```

In [47]:

```
# To display top 5 rows
X_train.head()
```

Out[47]:

	Age	Income	ZIP Code	Family	CCAvg	Education	Securities Account	CD Account	Online	CreditCard
0	51	5.058173	94301	3	0.322048	1	0	0	1	1
1	64	5.948841	90266	1	0.814468	2	1	0	0	0
2	52	5.651776	94923	4	0.902268	1	0	0	1	1
3	32	4.661500	93106	1	0.384643	3	0	0	1	0
4	62	7.097040	91320	1	0.544705	1	1	0	0	1

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

We will apply the StandardScaler to the dataset to standardize the input variables

In [48]:

```
for ind, column in enumerate(X_train.columns):
    scaler = StandardScaler()

    #fit to train data
    scaler.fit(X_train[[column]])

    #transform train data
    np_array = scaler.transform(X_train[[column]])
    X_train.loc[: , column] = pd.Series(np_array.flatten())

    #transform test data
    np_array = scaler.transform(X_test[[column]])
    X_test.loc[: , column] = pd.Series(np_array.flatten())
```

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

Logistic Regression

In [49]:

```
# Importing required libraries
from sklearn.linear_model import LogisticRegression
```

In [50]:

```
model = LogisticRegression(random_state = 0)
```

In [51]:

```
model.fit(X_train, Y_train)
```

Out[51]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [52]:

```
# Importing required libraries
from sklearn.metrics import confusion_matrix, recall_score , precision_score , f1_score , a
```

In [53]:

```
X_test_pred1 = model.predict(X_test)
X_test_pred1
```

Out[53]:

```
array([1, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [54]:

```
# Accuracy of train data
model.score(X_train, Y_train)
```

Out[54]:

```
0.9571428571428572
```

In [55]:

```
# Accuracy of test data
model.score(X_test,Y_test)
```

Out[55]:

```
0.9546666666666667
```

In [56]:

```
# Defining the Confusion Matrix
def Confusion_Matrix(actual, predicted):
    cm = confusion_matrix(actual, predicted)
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_ylim([0,5])
    sns.heatmap(cm, annot=True, fmt= '.2f', xticklabels= [0,1], yticklabels=[0,1])
    plt.ylabel('Observed')
    plt.xlabel('Predicted')
    plt.show()
```

In [57]:

```
Y_test.shape
```

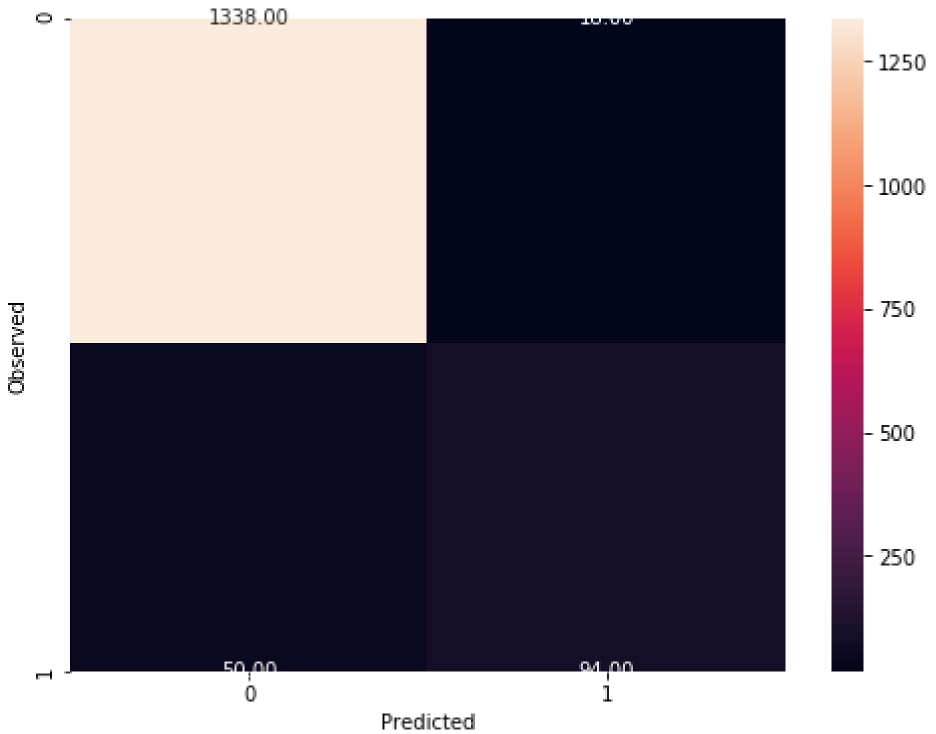
Out[57]:

```
(1500, 1)
```

In [58]:

```
print('Confusion Matrix')
print(Confusion_Matrix(Y_test,X_test_pred1.reshape(-1,1)))
```

```
Confusion Matrix
```




```
None
```

In [59]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,X_test_pred1.reshape(-1,1))
cm
```

Out[59]:

```
array([[1338,   18],
       [  50,   94]], dtype=int64)
```

7. Print all the metrics related for evaluating the model performance

In [60]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,X_test_pred1))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1356
           1       0.84      0.65      0.73       144

    accuracy                           0.95      1500
   macro avg       0.90      0.82      0.85      1500
weighted avg       0.95      0.95      0.95      1500
```

In [61]:

```
print("Roc Auc Score: ", roc_auc_score(Y_test,X_test_pred1))
```

```
Roc Auc Score:  0.819751720747296
```

For Logistic Regression we got 95% accuracy for test data. The F1 score is 0.73. Now lets compare that values with other models.

8. Build various other classification algorithms and compare their performance

Random Forest Classifier

Random forest is an ensemble machine learning algorithm.

It is perhaps the most popular and widely used machine learning algorithm given its good or excellent performance across a wide range of classification and regression predictive modeling problems.

It works in four steps:

1)Select random samples from a given dataset.

2)Construct a decision tree for each sample and get a prediction result from each decision tree.

3)Perform a vote for each predicted result.

4)Select the prediction result with the most votes as the final prediction.

In [63]:

```
# Importing required libraries
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(n_estimators=500, max_depth=8)
model2.fit(X_train, Y_train)
```

Out[63]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=8, max_features='auto', max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [64]:

```
X_test_pred2 = model2.predict(X_test)
X_test_pred2
```

Out[64]:

```
array([1, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [65]:

```
# Accuracy of train data
model2.score(X_train, Y_train)
```

Out[65]:

```
0.9945714285714286
```

In [67]:

```
# Accuracy of test data
model2.score(X_test,Y_test)
```

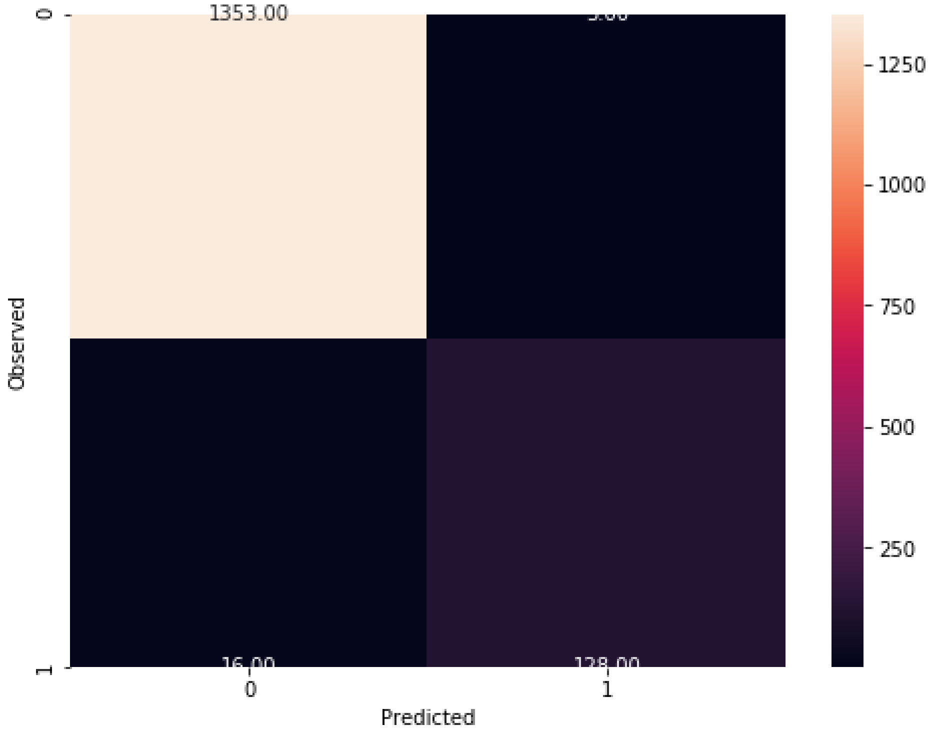
Out[67]:

```
0.9873333333333333
```

In [68]:

```
print('Confusion Matrix')
print(Confusion_Matrix(Y_test,X_test_pred2.reshape(-1,1)))
```

Confusion Matrix



None

In [69]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,X_test_pred2.reshape(-1,1))
cm
```

Out[69]:

```
array([[1353,    3],
       [  16,  128]], dtype=int64)
```

In [70]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,X_test_pred2))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1356
           1       0.98      0.89      0.93       144

    accuracy                           0.99      1500
   macro avg       0.98      0.94      0.96      1500
weighted avg       0.99      0.99      0.99      1500
```

In [71]:

```
print("Roc Auc Score: ", roc_auc_score(Y_test,X_test_pred2))
```

```
Roc Auc Score:  0.943338249754179
```

The ROC AUC score and F1 score are higher then Logistic Regression model.

Decision Tree Classifier

Decision Trees (DTs) are a non-parametric **supervised learning** method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

In [72]:

```
# Importing required libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [73]:

```
model3 = DecisionTreeClassifier(random_state=0, max_depth=8)
```

In [74]:

```
model3.fit(X_train, Y_train)
```

Out[74]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=8,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [75]:

```
X_test_pred3 = model3.predict(X_test)
X_test_pred3
```

Out[75]:

```
array([1, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [76]:

```
# Accuracy of train data
model3.score(X_train, Y_train)
```

Out[76]:

```
0.996
```

In [77]:

```
# Accuracy of test data
model3.score(X_test,Y_test)
```

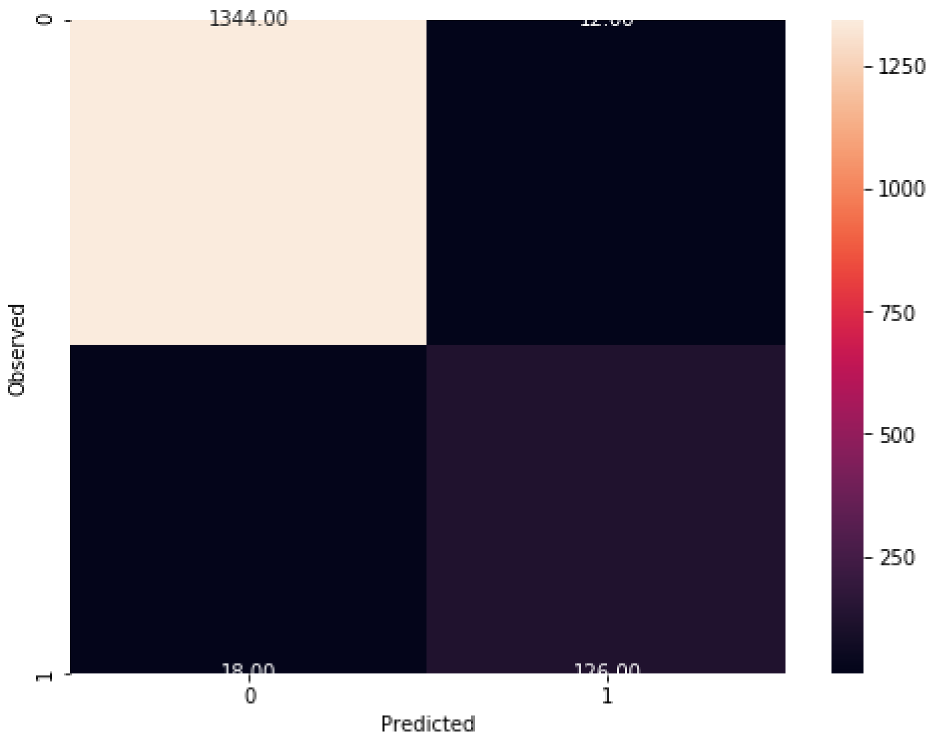
Out[77]:

```
0.98
```

In [78]:

```
print('Confusion Matrix')
print(Confusion_Matrix(Y_test,X_test_pred3.reshape(-1,1)))
```

```
Confusion Matrix
```




```
None
```

In [79]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,X_test_pred3.reshape(-1,1))
cm
```

Out[79]:

```
array([[1344,   12],
       [  18,  126]], dtype=int64)
```

In [80]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,X_test_pred3))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1356
           1       0.91      0.88      0.89       144

    accuracy                           0.98      1500
   macro avg       0.95      0.93      0.94      1500
weighted avg       0.98      0.98      0.98      1500
```

In [81]:

```
print("Roc Auc Score: ", roc_auc_score(Y_test,X_test_pred3))
```

```
Roc Auc Score:  0.933075221238938
```

Naive Bayes

Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

P(class|data) = (P(data|class) * P(class)) / P(data)
Where P(class|data) is the probability of class given the provided data.

In [82]:

```
# Importing required libraries
from sklearn.naive_bayes import GaussianNB
model4 = GaussianNB()
model4.fit(X_train,Y_train)
```

Out[82]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [83]:

```
X_test_pred4 = model4.predict(X_test)
X_test_pred4
```

Out[83]:

```
array([1, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [84]:

```
# Accuracy of train data
model4.score(X_train, Y_train)
```

Out[84]:

```
0.9105714285714286
```

In [85]:

```
# Accuracy of test data
model4.score(X_test,Y_test)
```

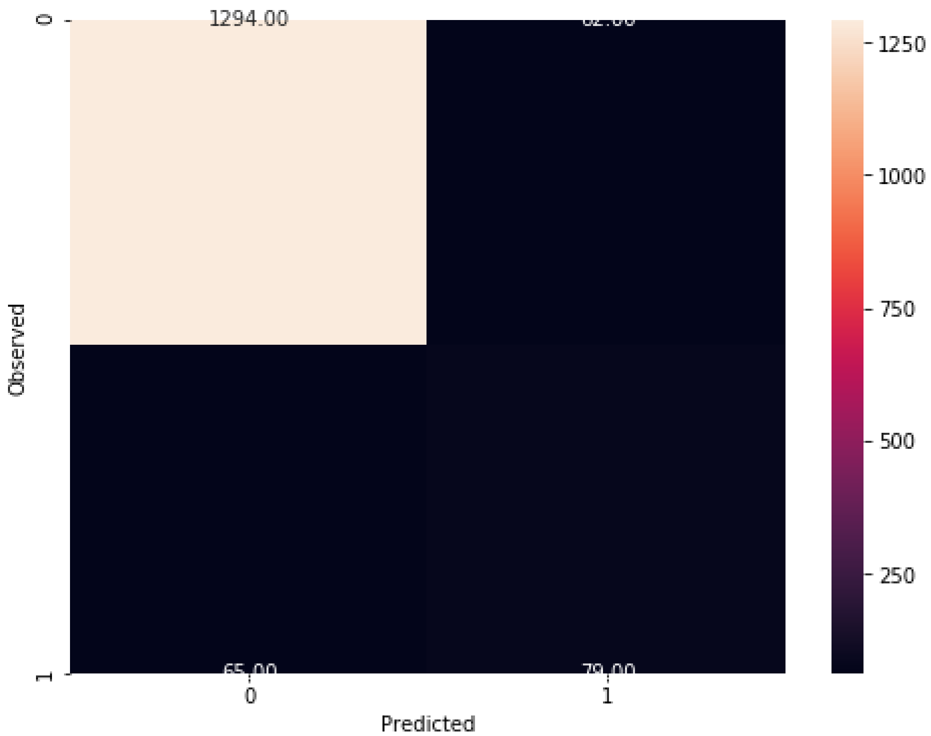
Out[85]:

```
0.9153333333333333
```

In [86]:

```
print('Confusion Matrix')
print(Confusion_Matrix(Y_test,X_test_pred4.reshape(-1,1)))
```

```
Confusion Matrix
```




```
None
```

In [87]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,X_test_pred4.reshape(-1,1))
cm
```

Out[87]:

```
array([[1294,   62],
       [  65,   79]], dtype=int64)
```

In [88]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,X_test_pred4))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      1356
           1       0.56      0.55      0.55       144

    accuracy                           0.92      1500
   macro avg       0.76      0.75      0.75      1500
weighted avg       0.91      0.92      0.91      1500
```

In [89]:

```
print("Roc Auc Score: ", roc_auc_score(Y_test,X_test_pred4))
```

```
Roc Auc Score:  0.7514441986234022
```

9. Give a business understanding of your model

In the first step of this project we imported various libraries and our data. Than we found out various things about our data.

1) We have to make the model to predict whether a person will take personal loan or not.
2) We found that age and experience are highly correlated so we droped the experience column.
3) ID and ZIPcode were not contributing factors for a person to take loan so we dropped them.
4) The Income and CCAvg column were left skewed so we applied Power transformation to them to normalize them.
5) The mortgage column was also skewed but since it was discrete so rather than power transformation, we use binning technique.

After this we used several models to make predictions.

1. Logistic Regression
2. Random Forest Classifier
3. Decision Tree Classifier
4. Naive Bayes

1. Logistic Regression

ACCURACY SCORE: 95.46%

CONFUSION MATRIX: [[1338, 18], [50, 94]]

CLASSIFICATION REPORT: precision recall f1-score support

```
           0       0.96      0.99      0.98      1356
           1       0.84      0.65      0.73       144

    accuracy                           0.95      1500
   macro avg       0.90      0.82      0.85      1500
weighted avg       0.95      0.95      0.95      1500
```

2. Random Forest Classifier

ACCURACY SCORE: 98.73%

CONFUSION MATRIX: [[1353, 3], [16, 128]]

CLASSIFICATION REPORT: precision recall f1-score support

```
           0       0.99      1.00      0.99      1356
           1       0.98      0.89      0.93       144

    accuracy                           0.99      1500
   macro avg       0.98      0.94      0.96      1500
weighted avg       0.99      0.99      0.99      1500
```

3. Decision Tree Classifier

ACCURACY SCORE: 98%

CONFUSION MATRIX: [[1344, 12], [18, 126]]

CLASSIFICATION REPORT: precision recall f1-score support

```
           0       0.99      0.99      0.99      1356
           1       0.91      0.88      0.89       144

    accuracy                           0.98      1500
   macro avg       0.95      0.93      0.94      1500
weighted avg       0.98      0.98      0.98      1500
```

4. Naive Bayes

ACCURACY SCORE: 91.5%

CONFUSION MATRIX: [[1294, 62] [65, 79]]

CLASSIFICATION REPORT: precision recall f1-score support

```
           0       0.95      0.95      0.95      1356
           1       0.56      0.55      0.55       144

    accuracy                           0.92      1500
   macro avg       0.76      0.75      0.75      1500
weighted avg       0.91      0.92      0.91      1500
```

Conclusion

The aim of the universal 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. Four classification algorithms were used in this project. From the implementation, it seems like **Random Forest Classifier** have the highest accuracy and we can choose that as our final model

In [ ]: