Case study on Bank Loan dataset

===

Bank loan dataset contains data of 5000 customers. The data include customer information (age, income, etc.), the customer's relationship with the bank (mortgage, securities account, etc.), and the customer response to the last personal loan campaign (Personal Loan).

Objective:

To predict the likelihood of a liability customer buying personal loans.

1. Reading the data

In [1]:
```
# Importing the libraries
import pandas as pd                     # for data manipulation
import seaborn as sns                   # for statistical data visualisation
import numpy as np                      # for linear algebra
import matplotlib.pyplot as plt         # for data visualization
from scipy import stats                 # for calculating statistics

# Importing various machine learning algorithm from sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,roc_curve,auc,accuracy_score
from  sklearn.neighbors import KNeighborsClassifier
from scipy.stats import zscore
from sklearn.naive_bayes import GaussianNB
```

In [2]:
```
dataframe= pd.read_csv("Bank_Personal_Loan_Modelling.csv")  # Reading the data
dataframe.head()   # showing first 5 datas
```

Out[2]:

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

In [3]:
```
dataframe.shape
```

Out[3]: (5000, 14)

The data given has 14 columns and consist of 5000 data.

In [4]:
```
dataframe.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
```

The above information shows the following:

a. There are no null or missing values present
b. The attributes are either int or float

In [5]:
```python
dataframe.apply(lambda x: len(x.unique()))
```

Out[5]:
```
ID                    5000
Age                     45
Experience              47
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
```

From the data:

The ID column is associated with customers ID and does not provide any valuable information for the prediction of personal loan. So this variable can be neglected in model predictions.

5 variable have interval data:

Age: Age of the customer

Experience: Years of experience of customer

Income: Annual income of customer in "$"

CCAvg: Average spending in credit card

Mortage: Value of House Mortgage

5 variables have categorical data:

Personal Loan: customer accept the personal loan or not.

Securities Account: Does the customer have a securities account with the bank

CD Account: customer have a certificate of deposit or not

Online: Does the customer use internet banking

Credit card: Does customer use a credit card

2 variables contains Ordinal categorical data:

Family: Family size

Education: Education level of the customer

In [6]:
```
dataframe.iloc[:,1:].describe()
```

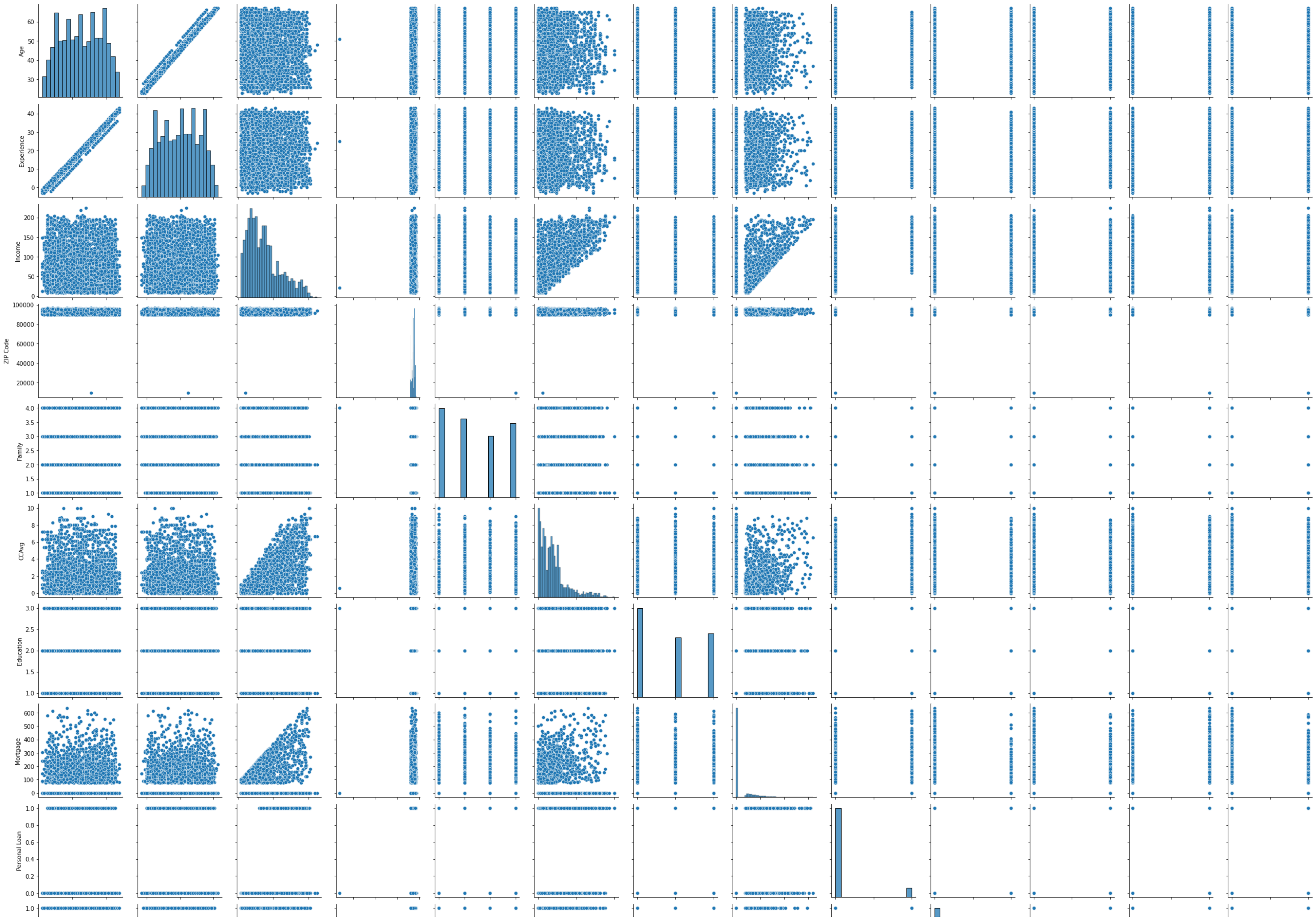
Out[6]:

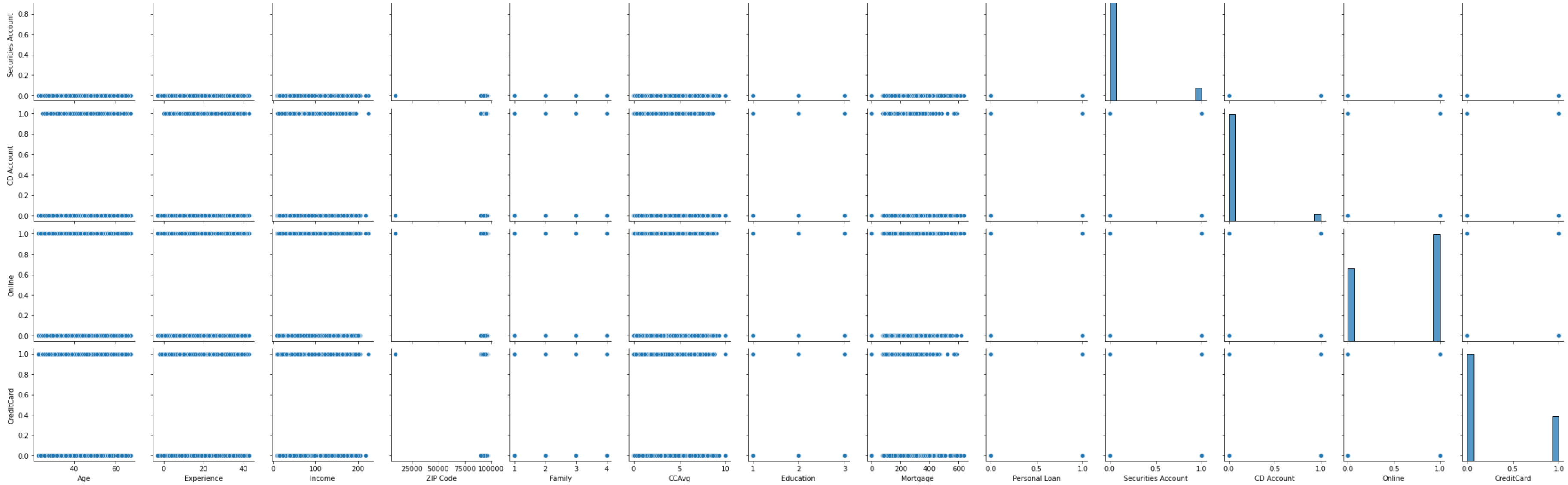
	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	Acc
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.0
mean	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937938	1.881000	56.498800	0.096000	0.104400	0.0
std	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747659	0.839869	101.713802	0.294621	0.305809	0.2
min	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.0
25%	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	0.0
50%	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	0.0
75%	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	0.0
max	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	1.0

The Experience column data should be cleaned as it contain negative values as experience. This can be seen in the value of min of Experience.

```
In [7]: sns.pairplot(dataframe.iloc[:,1:])
```

```
Out[7]: <seaborn.axisgrid.PairGrid at 0x1a4c7dcc9d0>
```

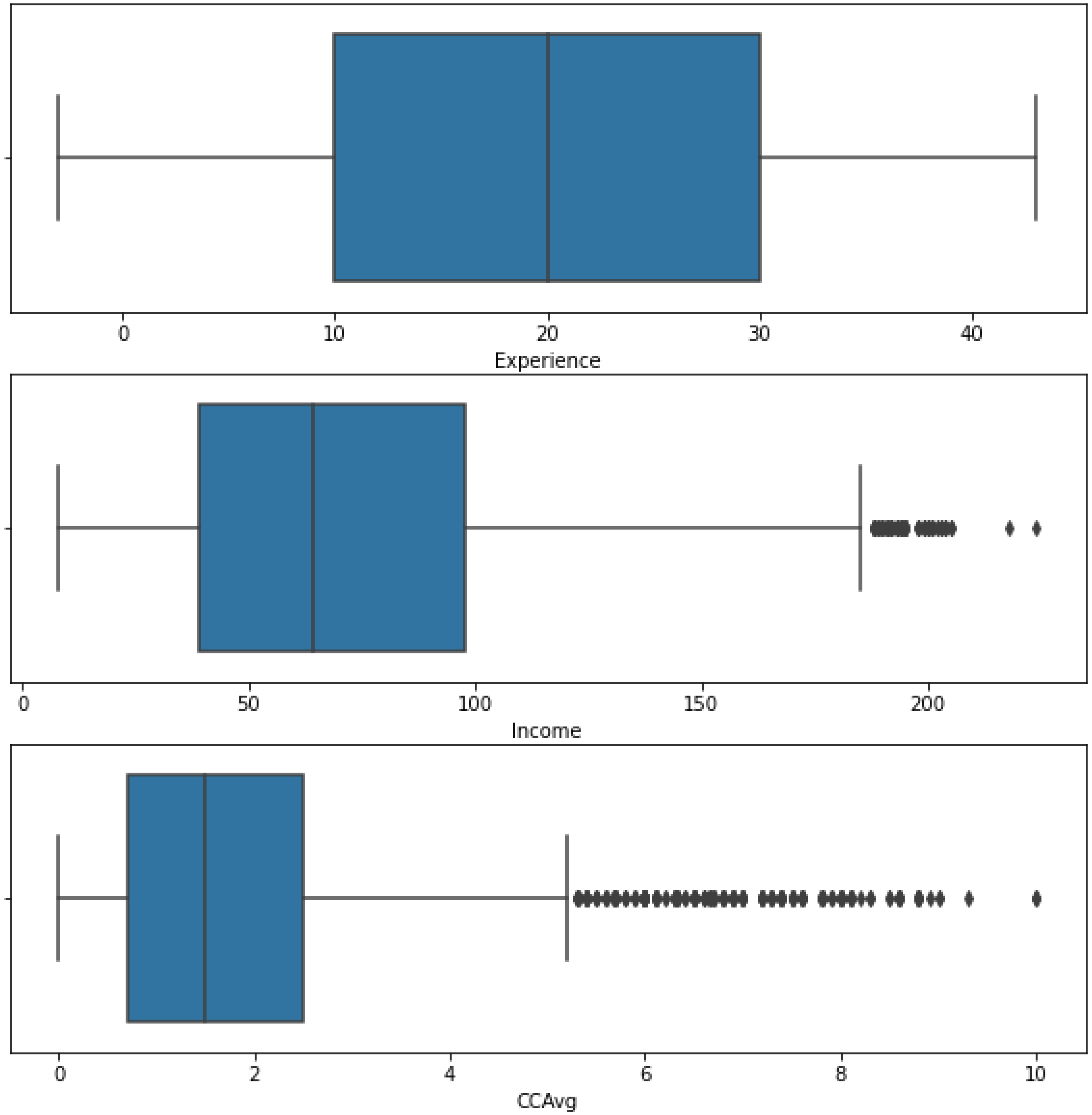



Securities Account
CD Account
Online
CreditCard
Age
Experience
Income
ZIP Code
Family
CCAvg
Education
Mortgage
Personal Loan
Securities Account
CD Account
Online
CreditCard

In [8]:
```
plt.figure(figsize=(10,10))
plt.subplot(3,1,1)
sns.boxplot(dataframe.Experience)
plt.subplot(3,1,2)
sns.boxplot(dataframe.Income)
plt.subplot(3,1,3)
sns.boxplot(dataframe.CCAvg)
```

```
C:\Users\celvi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\celvi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\celvi\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[8]:
```
<AxesSubplot:xlabel='CCAvg'>
```


0
10
20
30
40
Experience
0
50
100
150
200
Income
0
2
4
6
8
10
CCAvg

Obv

Age feature is normally distributed.

Experience is normally distributed. As both Age and Experience the mean is nearly equal to median.

Income and CCAvg is positively skewed as we can see the mean is greater than the median

CCAvg is highly skewed and has lot of outliers.

There are some negative values contained in experience that actually dont make any sense. Its better to clean them by applying the median of experience of the group having same age and education but positive experience.

In [9]:
```
dataframe.iloc[:,1:9].skew()
```

Out[9]:
```
Age           -0.029341
Experience    -0.026325
Income         0.841339
ZIP Code     -12.500221
Family         0.155221
CCAvg          1.598443
Education      0.227093
Mortgage       2.104002
dtype: float64
```

In [10]:
```
dataframe.Experience[dataframe.Experience<0].count()
```

Out[10]:
```
52
```

In [11]:
```
neg_ids=dataframe.loc[dataframe.Experience<0].ID.tolist()
pos_exp_data=dataframe.loc[dataframe.Experience>0]
for i in neg_ids:
    education=dataframe.Education[dataframe.ID==i].tolist()[0]
    age=dataframe.Age[dataframe.ID==i].tolist()[0]
    pos_record=pos_exp_data[(pos_exp_data.Age==age) & (pos_exp_data.Education==education)]
    x=pos_record['Experience'].median()
    dataframe.loc[(dataframe.ID==i),'Experience']=x
```

In [12]:
```
dataframe.Experience[dataframe.Experience<0].count()
```

Out[12]:
```
0
```

In [13]:
```
dataframe.Experience.describe()
```

Out[13]:
```
count    4971.000000
mean       20.243211
std        11.359189
min         0.000000
25%        10.000000
50%        20.000000
75%        30.000000
max        43.000000
Name: Experience, dtype: float64
```

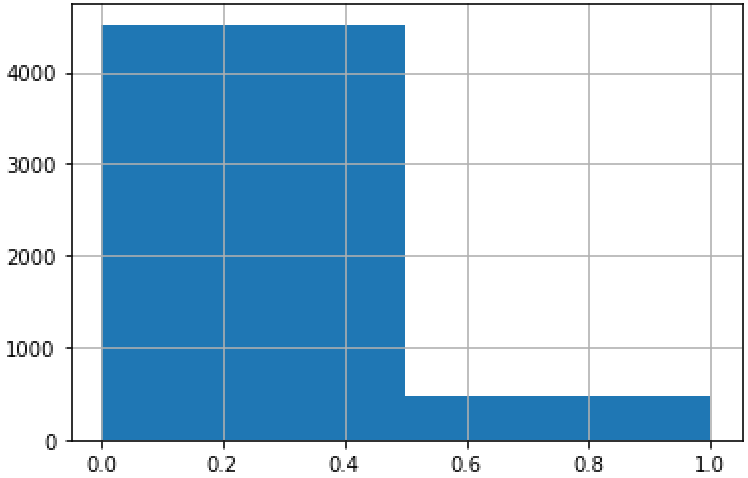
So the negative values are removed by using median of experience from group having same age and education data as of data of negative valued experience.

Choosing the target column

As the objective is to redict the likelihood of a liability customer buying personal loans, the Personal Loan column will be target column. And the distribution is as shown

In [14]:
```
dataframe["Personal Loan"].hist(bins=2)
```

Out[14]: <AxesSubplot:>



In [15]:
```
dataframe["Personal Loan"].value_counts()
```

Out[15]:
```
0    4520
1     480
Name: Personal Loan, dtype: int64
```

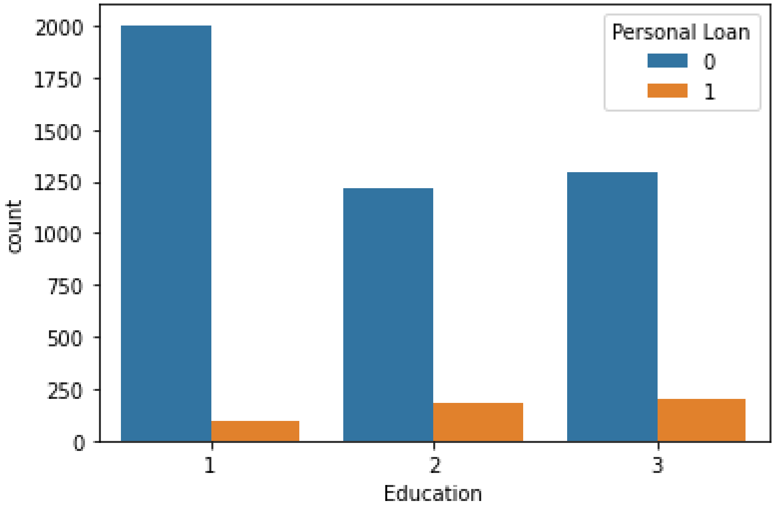
As in the data, the count of customer how takes the personal loan is very less compared to who didn't. Due to which there maybe chances that the model perdiction will be effected due to this.

Checking the influence of various attributes on customer taking personal loan

Influence of Customers Education on taking personal Loan

In [16]:
```
sns.countplot(x='Education',data=dataframe,hue='Personal Loan')
```

Out[16]: <AxesSubplot:xlabel='Education', ylabel='count'>

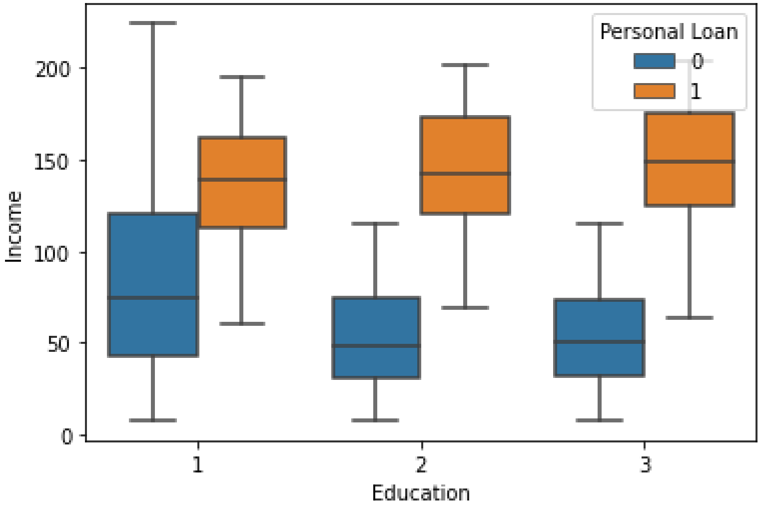


Obv: The graph shows there is no much influence of education on customers to personal loan.

Influence of Customers Income on taking personal Loan

In [17]:
```
sns.boxplot(x='Education',y='Income',hue='Personal Loan',data=dataframe)
```

Out[17]: <AxesSubplot:xlabel='Education', ylabel='Income'>



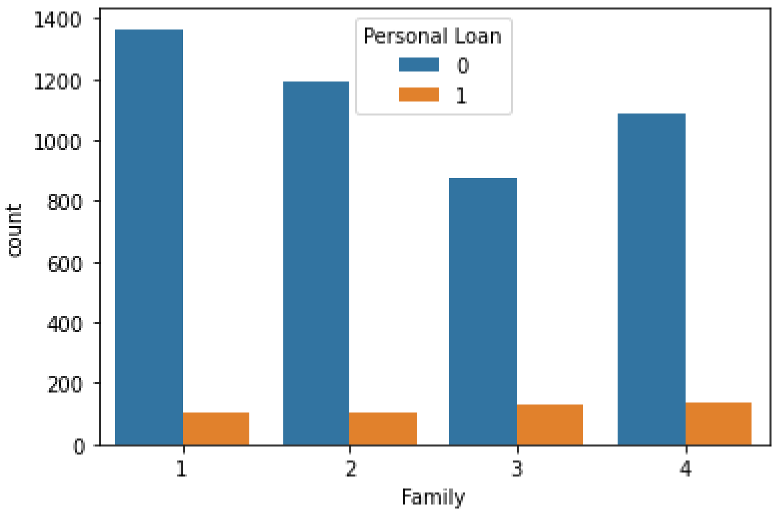
Obv

The graph shows that the customer who has taken the personal loan has same income level regardless education level.

Influence of Customers' Family size and Income on taking personal Loan

In [18]:
```
sns.countplot(x="Family", data=dataframe,hue="Personal Loan")
```

Out[18]: <AxesSubplot:xlabel='Family', ylabel='count'>



In [19]:
```
fs_takenloan = np.mean( dataframe[dataframe['Personal Loan']== 0].Family )
fs_nottaken_loan = np.mean( dataframe[dataframe['Personal Loan'] == 1].Family )
print("Family size of those taken loan  is",fs_takenloan )
print("Family size of those not taken loan  is",fs_nottaken_loan )
```

```
Family size of those taken loan  is 2.3734513274336284
Family size of those not taken loan  is 2.6125
```

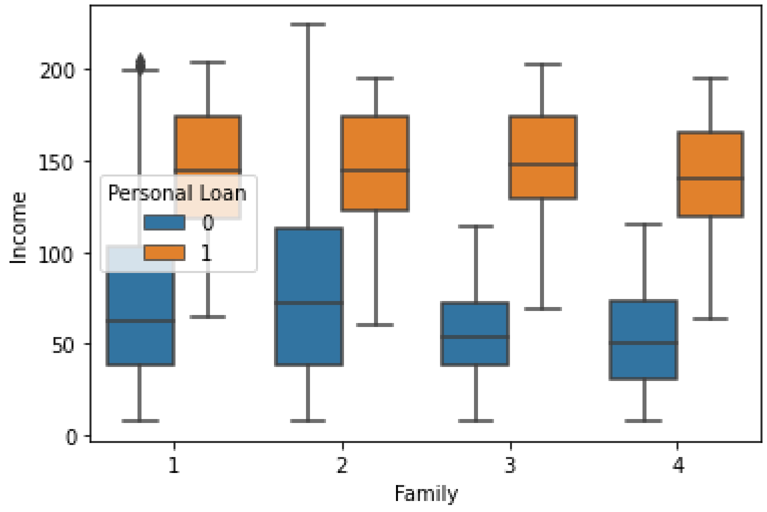
In [20]:
```
stats.ttest_ind(dataframe[dataframe['Personal Loan'] == 1]['Family'], dataframe[dataframe['Personal Loan'] == 1]['Family
```

Out[20]: Ttest_indResult(statistic=0.0, pvalue=1.0)

Family size seems to have no impact on decision to take a loan.

In [21]:
```
sns.boxplot(x='Family',y='Income',data=dataframe,hue='Personal Loan')
```

Out[21]:
```
<AxesSubplot:xlabel='Family', ylabel='Income'>
```



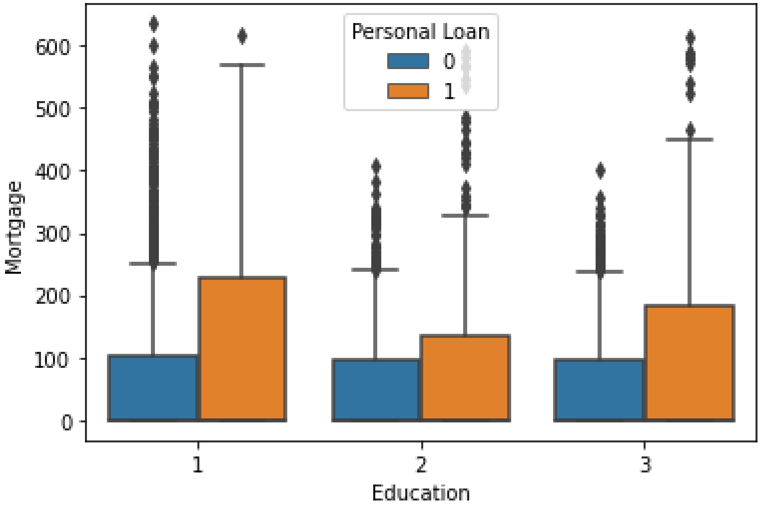
Obv

The graph shows that the customer who has taken the personal loan has same income level regardless home size

Influence of Customers' Mortgage on taking personal Loan

In [22]:
```
sns.boxplot(x='Education',y='Mortgage',data=dataframe,hue='Personal Loan')
```

Out[22]:
```
<AxesSubplot:xlabel='Education', ylabel='Mortgage'>
```



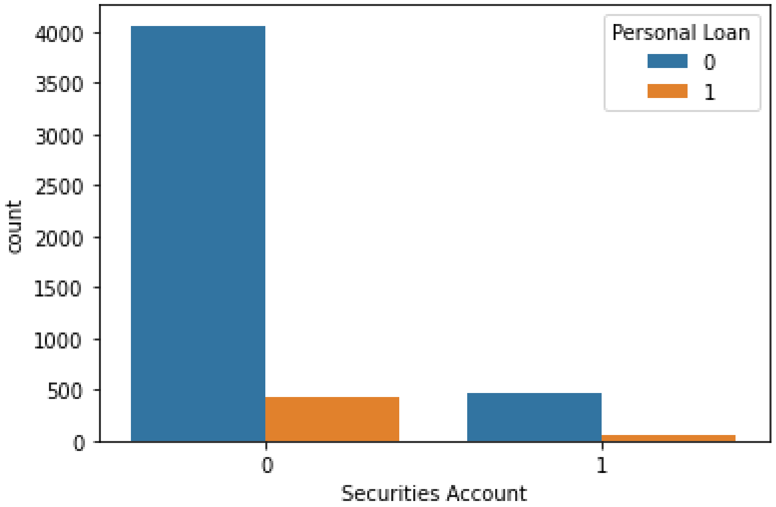
Obv

The customers who have taken the personal loan has hign Mortgage than customer who have not taken the Personal loan.

Influence of whether customer having Securities Account on taking personal Loan

In [23]:
```
sns.countplot(x='Securities Account',data=dataframe,hue='Personal Loan')
```

Out[23]:
```
<AxesSubplot:xlabel='Securities Account', ylabel='count'>
```

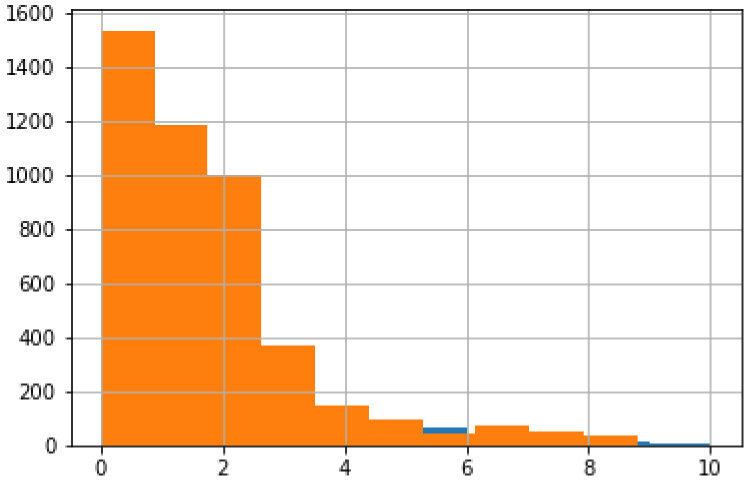


Obv

There are more customer who have Securities Account and not taken personal loan than vice-versa.

In [24]:
```
dataframe[dataframe['Personal Loan']==1].CCAvg.hist()
dataframe[dataframe['Personal Loan']==0].CCAvg.hist()
```

Out[24]: <AxesSubplot:>



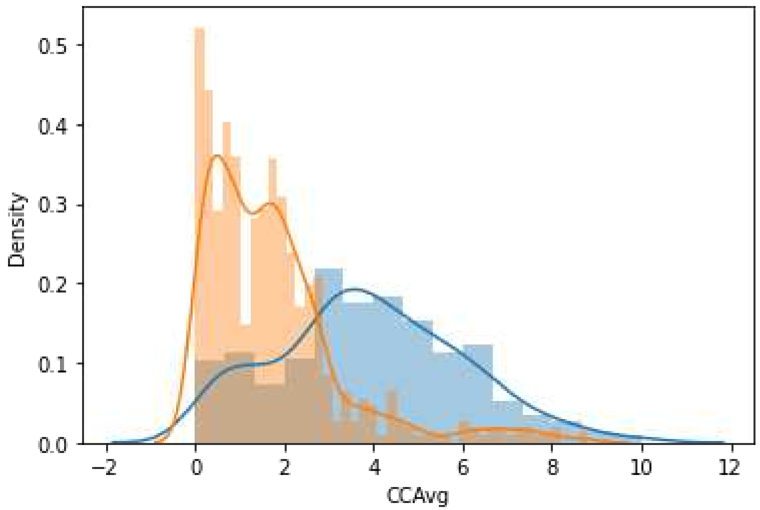
Obv

The graph show persons who have personal loan have a higher credit card average. Credit card spending greater than median of 1400 dollars is likely to take a loan.

In [25]:
```
sns.distplot(dataframe[dataframe['Personal Loan']==1].CCAvg)
sns.distplot(dataframe[dataframe['Personal Loan']==0].CCAvg)
```

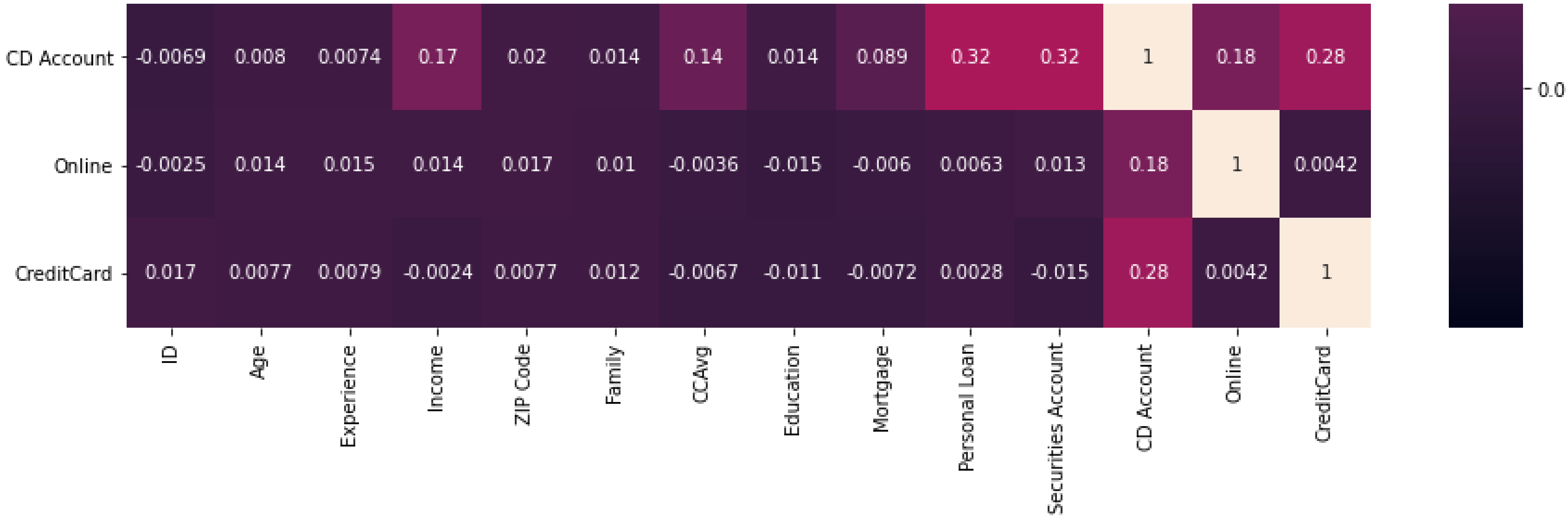
```
C:\Users\celvi\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\celvi\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[25]: <AxesSubplot:xlabel='CCAvg', ylabel='Density'>



In [26]:
```
corelation=dataframe.corr()
```

In [27]:
```python
plt.figure(figsize=(15,15))
a=sns.heatmap(corelation,annot=True)
```



Obv:

We can see that Customer's Income and CCAvg are fairly correlated

Also Age and Experience are highly correlated

Personal Loan and Income can be seen correlated from the heat map shown above

Classification Models

Splitting the Data

In [28]:
```
dataframe.columns
```

Out[28]:
```
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

As for the side note as the range of various attribute vary a lot (like range of age is 23 to 67 where are the income is 8 to 224 having different units),there may come a need to normalize the data.

But for Logestic Regression and Naive Bayes classification Normalization is not required as it does not effect it. For KNN algorithm normalization is required as it depends on distance of data points.

In [29]:
```
features=['Age', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Securities Account',
       'CD Account', 'Online', 'CreditCard']
X=dataframe[features]
Y=dataframe['Personal Loan']
```

Splitting the model in 7:3 ratio

In [30]:
```
train_X,test_X,train_y,test_y=train_test_split(X,Y,test_size=0.3,random_state=1)
train_X.count()
```

Out[30]:
```
Age                   3500
Income                3500
ZIP Code              3500
Family                3500
CCAvg                 3500
Education             3500
Mortgage              3500
Securities Account    3500
CD Account            3500
Online                3500
CreditCard            3500
dtype: int64
```

In [31]:
```
train_X.head()
```

Out[31]:

	Age	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Securities Account	CD Account	Online	CreditCard
1334	47	35	94304	2	1.3	1	0	0	0	1	0
4768	38	39	93118	1	2.0	2	0	0	0	1	0
65	59	131	91360	1	3.8	1	0	0	0	1	1
177	29	65	94132	4	1.8	2	244	0	0	0	0
4489	39	21	95518	3	0.2	2	0	0	0	1	0

In [32]:
```
test_X.count()
```

Out[32]:
```
Age                   1500
Income                1500
ZIP Code              1500
Family                1500
CCAvg                 1500
Education             1500
Mortgage              1500
Securities Account    1500
CD Account            1500
Online                1500
CreditCard            1500
dtype: int64
```

In [33]:
```
test_X.head()
```

Out[33]:

	Age	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Securities Account	CD Account	Online	CreditCard
2764	31	84	91320	1	2.9	3	105	0	0	0	1
4767	35	45	90639	3	0.9	1	101	1	0	0	0
3814	34	35	94304	3	1.3	1	0	0	0	0	0
3499	49	114	94550	1	0.3	1	286	0	0	1	0
2735	36	70	92131	3	2.6	2	165	0	0	1	0

Using Logestic Regression for prediction

Training the model

In [34]:
```
LR_Model=LogisticRegression()
Logestic_Model=LR_Model.fit(train_X,train_y)
Logestic_Model
```

Out[34]:
```
LogisticRegression()
```

Predicting from the trained model and showing the confusion matrix

In [35]:

```python
predict=LR_Model.predict(test_X)
print(predict[0:1000])
metrics=confusion_matrix(test_y,predict)
metrics
```

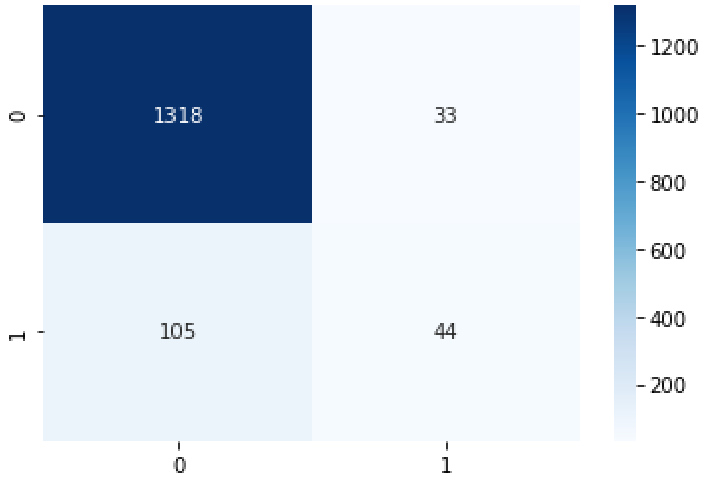
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0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

Out[35]:

```
array([[1318,   33],
       [ 105,   44]], dtype=int64)
```

In [36]:
```
sns.heatmap(metrics,annot=True,fmt='g',cmap='Blues')
```

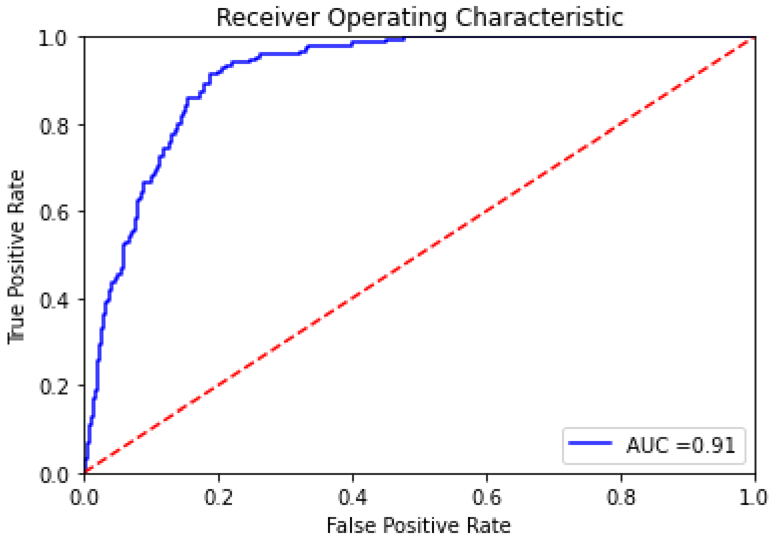
Out[36]: <AxesSubplot:>



In [37]:
```
print(classification_report(test_y,predict))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      1351
           1       0.57      0.30      0.39       149

    accuracy                           0.91      1500
   macro avg       0.75      0.64      0.67      1500
weighted avg       0.89      0.91      0.89      1500
```

In [38]:
```
probability=Logestic_Model.predict_proba(test_X)
pred=probability[:,1]
fpr,tpr,thresh=roc_curve(test_y,pred)
roc_auc=auc(fpr,tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr,tpr,'b',label='AUC =%0.2f'%roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0,1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [39]:
```
LR_accuracy=accuracy_score(test_y,predict)
LR_accuracy
```

Out[39]: 0.908

In [40]:
```
LR_AUC=roc_auc
LR_AUC
```

Out[40]:
```
0.9143264497091392
```

In [41]:
```
LR_Gini = 2*roc_auc - 1
LR_Gini
```

Out[41]:
```
0.8286528994182785
```

Obv:

The heat map shows that the model perdicts customer how dont take personal loan pretty well whereas prediction on wheather customer taking loan is not so good (44 out of 149).

The confusion matrix shows that the model prediction of customer taking loan not that satisfactory. This maybe due to lack of available data of customers who goes for taking personal loan for the model to learn.

As we can see the accuracy is 90.8% along with Area Under the Curve is 91.4% which pretty good. The Gini value is 0.828.

Using KNN Classification Model

Noramalising the data and training the model

In [42]:
```
X=dataframe[features].apply(zscore)
Y=dataframe['Personal Loan']
train_X,test_X,train_y,test_y=train_test_split(X,Y,test_size=0.3,random_state=1)
KNN_Model=KNeighborsClassifier()
KNearestN_Model=KNN_Model.fit(train_X,train_y)
KNearestN_Model
```

Out[42]:
```
KNeighborsClassifier()
```

Predicting from the trained model

In [43]:
```
predict=KNN_Model.predict(test_X)
predict[0:200,]
```

Out[43]:
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

Checking for suitable number for number of nearest neighbor the model should look into for prediction

In [44]:
```python
n=[1,3,5,7,11,13,15,17,19,21,23,25,27,29,31,33,35]
accuracy_scores=[]
for i in n:
    KNN_Model=KNeighborsClassifier(n_neighbors=i)
    KNN_Model.fit(train_X,train_y)
    predict=KNN_Model.predict(test_X)
    accuracy_scores.append(accuracy_score(test_y,predict))
accuracy_scores
```

Out[44]:
```
[0.9486666666666667,
 0.95,
 0.9506666666666667,
 0.9493333333333334,
 0.9466666666666667,
 0.944,
 0.946,
 0.9413333333333334,
 0.9406666666666667,
 0.9393333333333334,
 0.9366666666666666,
 0.936,
 0.934,
 0.9346666666666666,
 0.9353333333333333,
 0.9346666666666666,
 0.934]
```

Looks like for N_neigbors = 5 the accuracy score is highest for this model.

Checking fro whether we should use manhattan_distance (p=1) or euclidean_distance (p=2)

In [45]:
```python
p=[1,2]
accuracy_scores=[]
for i in p:
    KNN_Model=KNeighborsClassifier(n_neighbors=5,p=i)
    KNN_Model.fit(train_X,train_y)
    predict=KNN_Model.predict(test_X)
    accuracy_scores.append(accuracy_score(test_y,predict))
accuracy_scores
```

Out[45]:
```
[0.9506666666666667, 0.9506666666666667]
```

The p value doesnot make a difference in this model.

In [46]:
```python
KNN_Model=KNeighborsClassifier(n_neighbors=5,p=2)
KNN_Model.fit(train_X,train_y)
predict=KNN_Model.predict(test_X)
print(predict[0:200,])
Knn_matrics=confusion_matrix(test_y,predict)
Knn_matrics
```

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

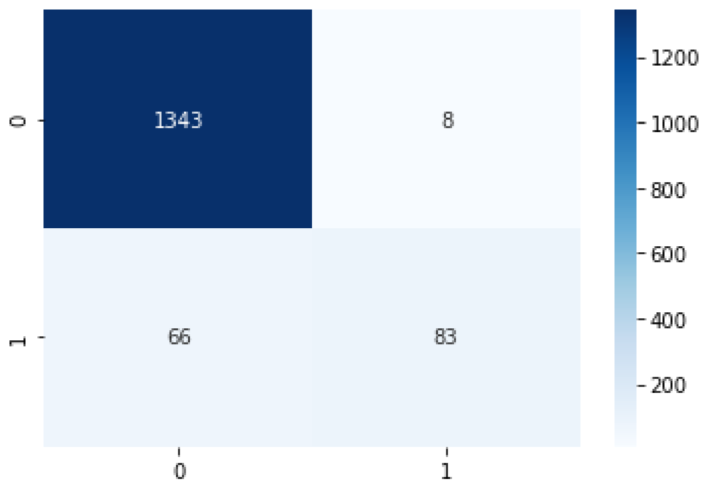
Out[46]:
```
array([[1343,    8],
       [  66,   83]], dtype=int64)
```

In [47]:
```
print(classification_report(test_y,predict))
```

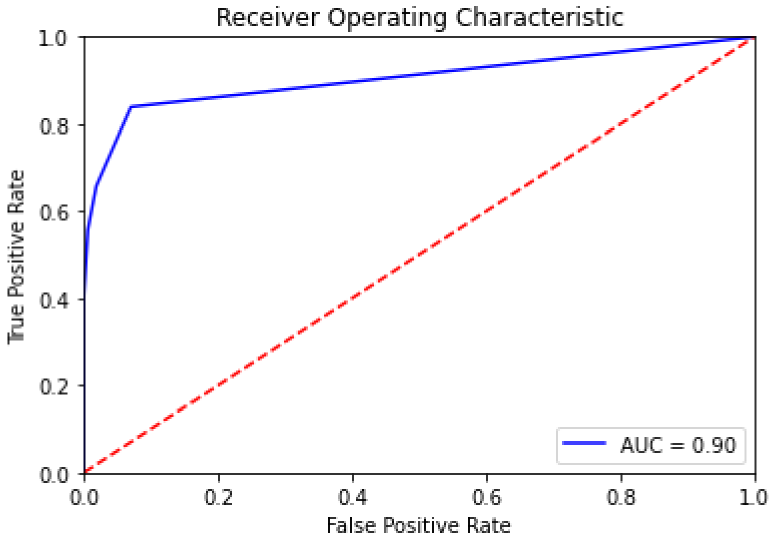
```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      1351
           1       0.91      0.56      0.69       149

    accuracy                           0.95      1500
   macro avg       0.93      0.78      0.83      1500
weighted avg       0.95      0.95      0.95      1500
```

In [48]:
```
sns.heatmap(Knn_matrics,annot=True,cmap='Blues',fmt='g')
```

Out[48]: <AxesSubplot:>



In [49]:
```
probs = KNN_Model.predict_proba(test_X)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(test_y, preds)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0,1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [50]:
```
KNN_accuracy=accuracy_score(test_y,predict)
KNN_accuracy
```
Out[50]:
```
0.9506666666666667
```
In [51]:
```
KNN_Gini=2*roc_auc-1
KNN_Gini
```
Out[51]:
```
0.8081311879343662
```
In [52]:
```
KNN_AUC=roc_auc
KNN_AUC
```
Out[52]:
```
0.9040655939671831
```
Obv:

The heat map shows that the model perdicts customer who dont take personal loan pretty well as well as prediction on wheather customer taking loan is also good compared to Logestic regression (83 out of 149).

The confusion matrix shows that the model prediction of customer taking loan is comparitively satisfactory. As we can see the accuracy has increased to 95.07% along with Area Under the Curve is 90.4% which pretty good. The Gini value is 0.808.

The AUC and Gini value decreased but not by huge difference.

Using Naive Bayes Classification Model

This model do not need the data to be normalised. Splitting the data again and Training the model

In [53]:
```
X=dataframe[features]
Y=dataframe['Personal Loan']
train_X,test_X,train_y,test_y=train_test_split(X,Y,test_size=0.3,random_state=1)
NB_Model=GaussianNB()
naiveB_Model=NB_Model.fit(train_X,train_y)
naiveB_Model
```

Out[53]:
```
GaussianNB()
```

In [54]:
```
predict=NB_Model.predict(test_X)
predict[0:200,]
```

Out[54]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

Predicting with the above trained model.

In [55]:
```
ac_score=accuracy_score(test_y,predict)
ac_score
```

Out[55]:
```
0.8833333333333333
```

In [56]:
```python
print(classification_report(test_y,predict))
```

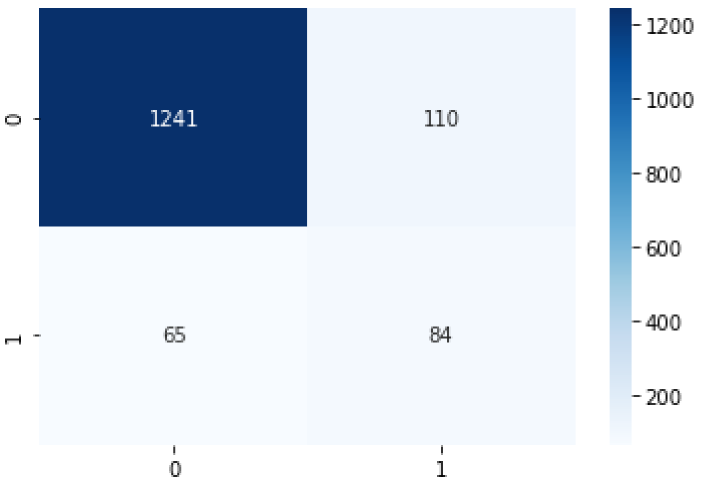
```
              precision    recall  f1-score   support

           0       0.95      0.92      0.93      1351
           1       0.43      0.56      0.49       149

    accuracy                           0.88      1500
   macro avg       0.69      0.74      0.71      1500
weighted avg       0.90      0.88      0.89      1500
```

In [57]:
```python
NB_matrics=confusion_matrix(test_y,predict)
NB_matrics
```

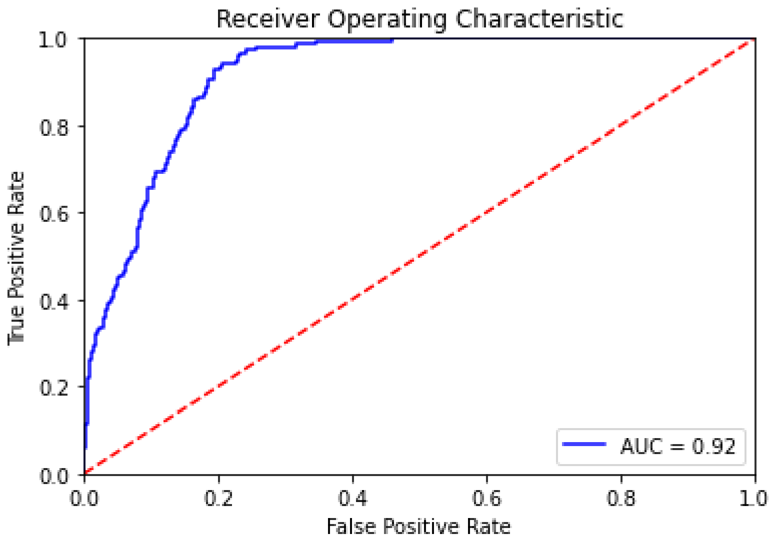
Out[57]:
```
array([[1241,  110],
       [  65,   84]], dtype=int64)
```

In [58]:
```python
sns.heatmap(NB_matrics,annot=True,cmap='Blues',fmt='g')
```

Out[58]: <AxesSubplot:>



In [59]:
```
probs=NB_Model.predict_proba(test_X)

preds = probs[:,1]
fpr, tpr, threshold = roc_curve(test_y, preds)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0,1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [60]:
```
NB_accuracy=accuracy_score(test_y,predict)
NB_accuracy
```
Out[60]:
```
0.8833333333333333
```
In [61]:
```
NB_Gini=2*roc_auc-1
NB_Gini
```
Out[61]:
```
0.8341869557225818
```
In [62]:
```
NB_AUC=roc_auc
NB_AUC
```
Out[62]:
```
0.9170934778612909
```
Obv:

The heat map shows that the model perdicts customer who dont take personal loan pretty well as well as prediction on wheather customer taking loan is also good compared to Logestic regression (84 out of 149).

The confusion matrix shows that the model prediction of customer taking loan is comparitively satisfactory. As we can see the accuracy has decreased to 88.3% compared to along with Area Under the Curve is 91.7% which pretty good. The Gini value is 0.834.

Comparing the models

In [63]:
```
data=[[LR_accuracy,LR_Gini,LR_AUC],[KNN_accuracy,KNN_Gini,KNN_AUC],[NB_accuracy,NB_Gini,NB_AUC]]
```

In [64]:
```
comparison=pd.DataFrame(data,index=['Logestic','KNN','Naive Bayes'],columns=['Accuracy','Gini','AUC'])
comparison
```

Out[64]:

	Accuracy	Gini	AUC
Logestic	0.908000	0.828653	0.914326
KNN	0.950667	0.808131	0.904066
Naive Bayes	0.883333	0.834187	0.917093

As for the above matrix, the accuracy of KNN model is highest among others whereas the Gini value and AUC is of KNN model is lower but nor significantly.

So, in this case KNN model would be the best model to use for predicting the likelihood of a liability customer buying personal loans.

In [ ]: