In [62]:

```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set(color_codes=True)

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LinearRegression

data=pd.read_csv('Bank_Personal_Loan_Modelling.csv')
```

In [63]:

```
data.head()
```

Out[63]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Se A
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 [64]:

```
data.tail()
```

Out[64]:

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

In [65]:

```
data.size
```

Out[65]:

```
70000
```

In [66]:

```
#check datatypes
data.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 [67]:

```
# Categrical Variables are
# Age, Experience, Family, Education,Personal Loan,Securities Acc,CD Account,Online, Credit Card
# Categorical_Variables=['Age','Experience','Family', 'Education','Personal Loan','Securities Acc','CD Account','Online', 'Credit Card]
```

In [68]:

```
#data shape
print('Number of rows in dataset: ',data.shape[0])
print('Number of columns in dataset:',data.shape[1])
```

```
Number of rows in dataset:  5000
Number of columns in dataset: 14
```

In [69]:

```
# data statistics
data.describe().T
```

Out[69]:

	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	1.937938	1.747659	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

In [70]:

```
# number of unique values in each column
data.nunique()
```

Out[70]:

```
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
```

In [71]:

```
# check for null values
print (data.isnull().sum())
data.isnull().any().any()
```

```
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
```

Out[71]:

```
False
```

In [72]:

```
# number of people with zero mortgage
zero_mortgage=data[data['Mortgage']==0]
print (' The number of customers that have no mortgage is',zero_mortgage ['Mortgage'].c
ount())
```

```
 The number of customers that have no mortgage is 3462
```

In [73]:

```
zero_credit_card=data[data['CreditCard']==0]
print (' The number of customers with zero card spending is',zero_credit_card['CreditCa
rd'].count())
```

```
 The number of customers with zero card spending is 3530
```

In [74]:

```
data.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
```

Since there are no non-null numbers, that means that there are no missing variables in columns, hence we do not need to replace them with means...

In [75]:

```
z=data['Family']
z.value_counts()
```

Out[75]:

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

In [76]:

```
# checking value counts of all categorical (integer) type columns
# is there a way to feed it a list of columns (defined as catagorical columns as i did
 it above, rather than by datatype?

for i in list(data.columns[data.dtypes=='int64']):
    print(data[i].value_counts())
    print()
```

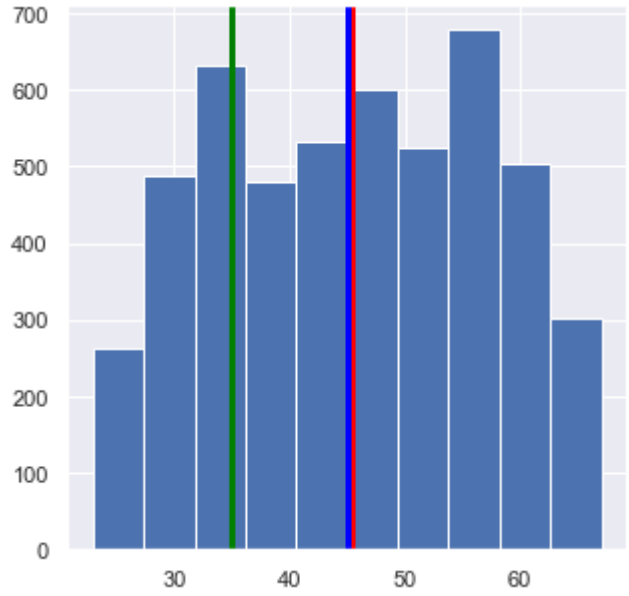
```
94720    169
94305    127
95616    116
90095     71
93106     57
         ...
96145      1
94970      1
94598      1
90068      1
94087      1
Name: ZIP Code, Length: 467, dtype: int64

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

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

0      3462
98       17
103      16
119      16
83       16
        ...
541       1
509       1
505       1
485       1
577       1
Name: Mortgage, Length: 347, dtype: int64

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

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

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

1    2984
0    2016
Name: Online, dtype: int64

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

In [77]:

```
## UNIVARIATE
#Univariate Catagorical Variable Age
plt.figure(figsize=(5,5))
plt.hist(data['Age']);
plt.axvline(data['Age'].mean(),color='red', linewidth=3)
plt.axvline(data['Age'].median(),color='blue', linewidth=3)
plt.axvline(data['Age'].mode()[0],color='green', linewidth=3)
```

Out[77]:

```
<matplotlib.lines.Line2D at 0x206373558c8>
```



In [78]:

```
# Graph above shows that Median and Mean are about the same so the data had normal dist
ribution
# but to check that again using thr Skew function to see if value = 0
# Age =  -0.029341, so its very very small right skewness, so we can accept this as a n
ormal distribution

data.skew()
```

Out[78]:

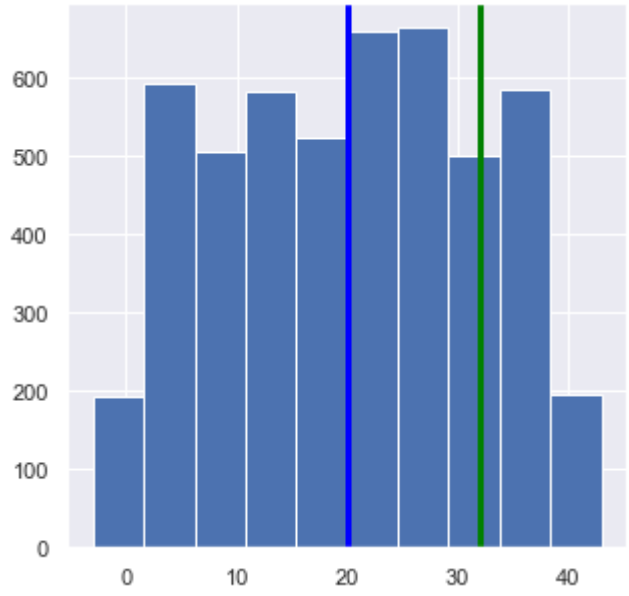
```
ID                    0.000000
Age                  -0.029341
Experience           -0.026325
Income                0.841339
ZIP Code            -12.500221
Family                0.155221
CCAvg                 1.598443
Education             0.227093
Mortgage              2.104002
Personal Loan         2.743607
Securities Account    2.588268
CD Account            3.691714
Online               -0.394785
CreditCard            0.904589
dtype: float64
```

In [79]:

```
#Univariate rest of Catagorical Variables
# why did mean not show ?
plt.figure(figsize=(5,5))
plt.hist(data['Experience']);
plt.axvline(data['Experience'].mean(),color='red', linewidth=3)
plt.axvline(data['Experience'].median(),color='blue', linewidth=3)
plt.axvline(data['Experience'].mode()[0],color='green', linewidth=3)
```

Out[79]:

```
<matplotlib.lines.Line2D at 0x206384d18c8>
```



In [80]:

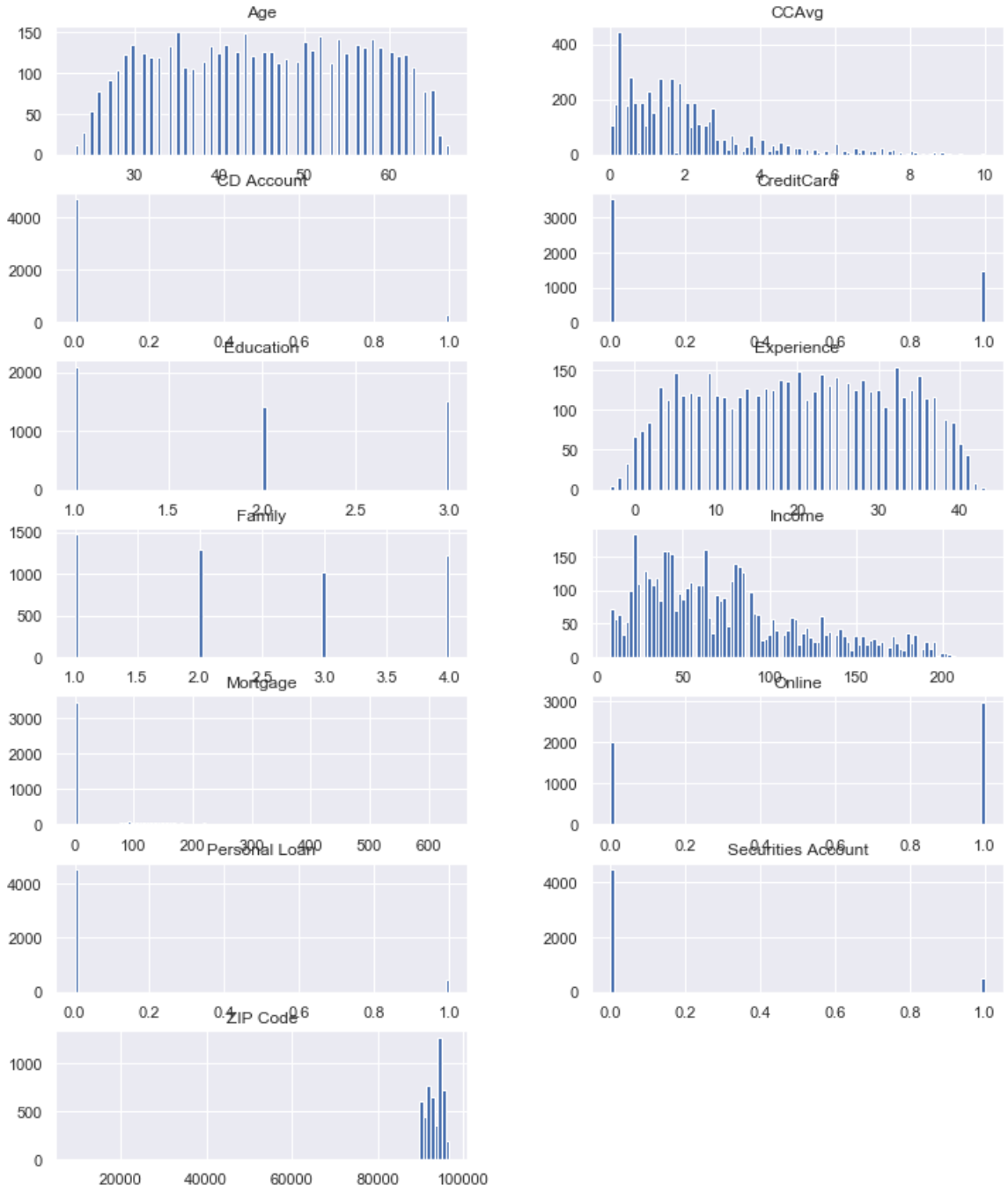
```
# see all column headings
List1=list(data)[0:14]
List1
```

Out[80]:

```
['ID',
 'Age',
 'Experience',
 'Income',
 'ZIP Code',
 'Family',
 'CCAvg',
 'Education',
 'Mortgage',
 'Personal Loan',
 'Securities Account',
 'CD Account',
 'Online',
 'CreditCard']
```

In [81]:

```
# Drop ID variable that adds no value to
# Unvariate
columns=list(data)[1:14] # without ID variable
data [columns].hist (stacked=False, bins=100, figsize=(12,30), layout=(14,2));
```

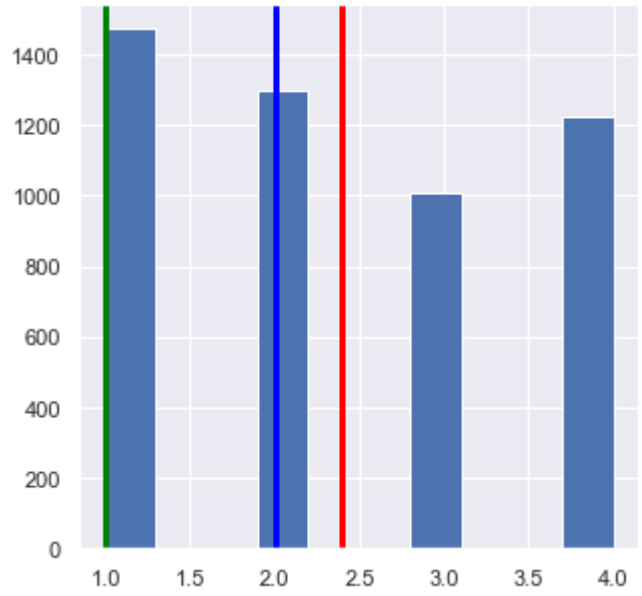


In [82]:

```
# Univariate in more precise way with mode mean and median - for each categorical varla
ble
#Univariate Catagorical Variable Family
plt.figure(figsize=(5,5))
plt.hist(data['Family']);
plt.axvline(data['Family'].mean(),color='red', linewidth=3)
plt.axvline(data['Family'].median(),color='blue', linewidth=3)
plt.axvline(data['Family'].mode()[0],color='green', linewidth=3)
```

Out[82]:

```
<matplotlib.lines.Line2D at 0x2063c482648>
```

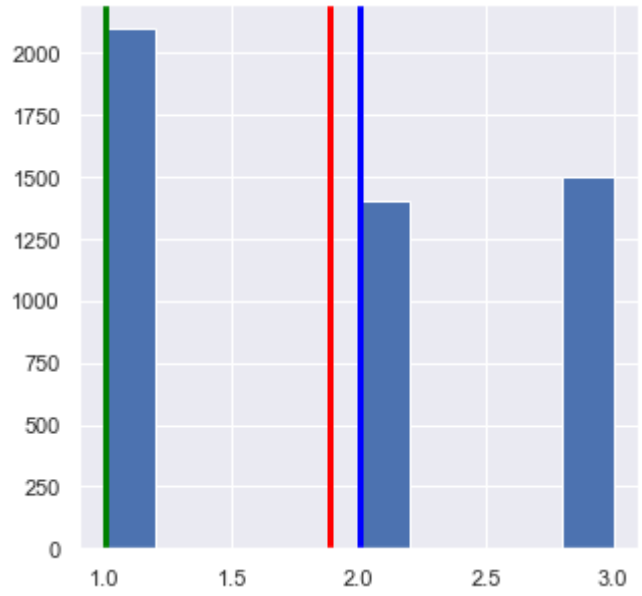


In [83]:

```
#'Age','Experience','Family', 'Education','Personal Loan','Securities Acc','CD Accoun
t','Online', 'Credit Card]

#Univariate Catagorical Variable Education
plt.figure(figsize=(5,5))
plt.hist(data['Education']);
plt.axvline(data['Education'].mean(),color='red', linewidth=3)
plt.axvline(data['Education'].median(),color='blue', linewidth=3)
plt.axvline(data['Education'].mode()[0],color='green', linewidth=3)
```

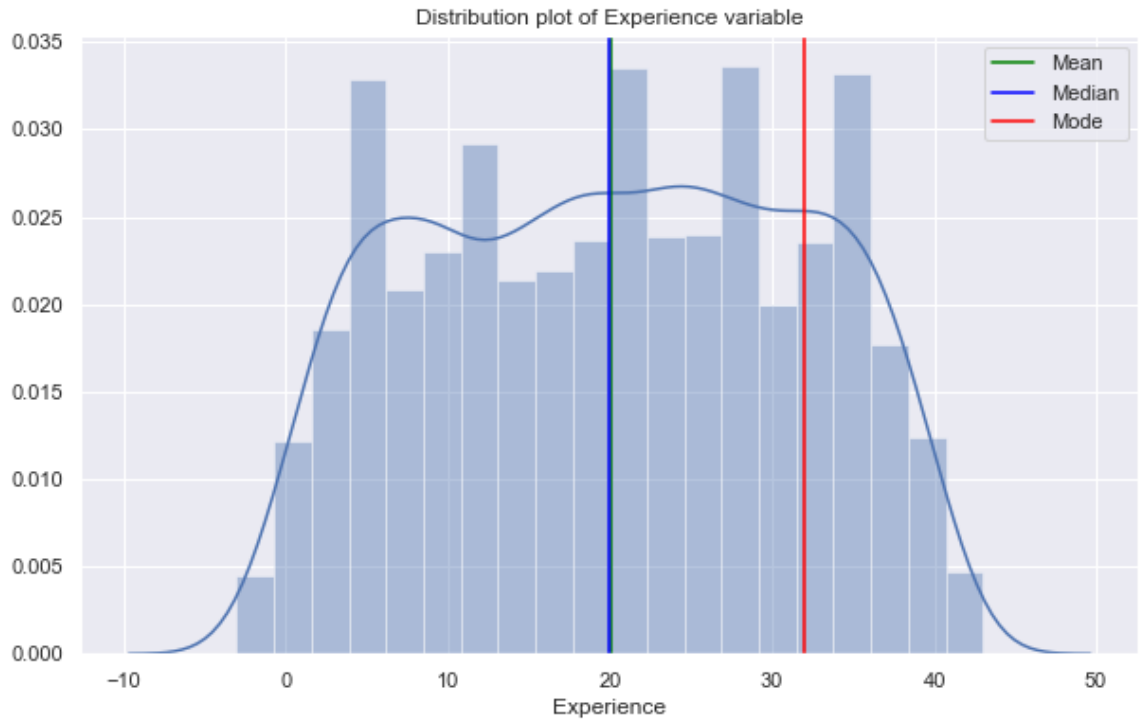
Out[83]:

```
<matplotlib.lines.Line2D at 0x20636f8d248>
```



In [85]:

```
# Univariate Plot - Experience
# Why do we not see mean ?
mean=data['Experience'].mean();
median=data['Experience'].median();
mode=data['Experience'].mode();

fig, ax = plt.subplots(figsize=(10,6)); # Canvas Size


sns.distplot(data['Experience']);

plt.title('Distribution plot of Experience variable');

plt.axvline(mean,color='green',label='Mean');
plt.axvline(median,color='blue',label='Median');
plt.axvline(mode[0],color='red',label='Mode')


plt.legend();
```



In [87]:

```
#BIVARIATE
sns.pairplot(data,);
```

In [88]:

```
# drop the usless ID variable that has no correlation with any other variables
#data=data.drop(['ID'],axis=1)
```

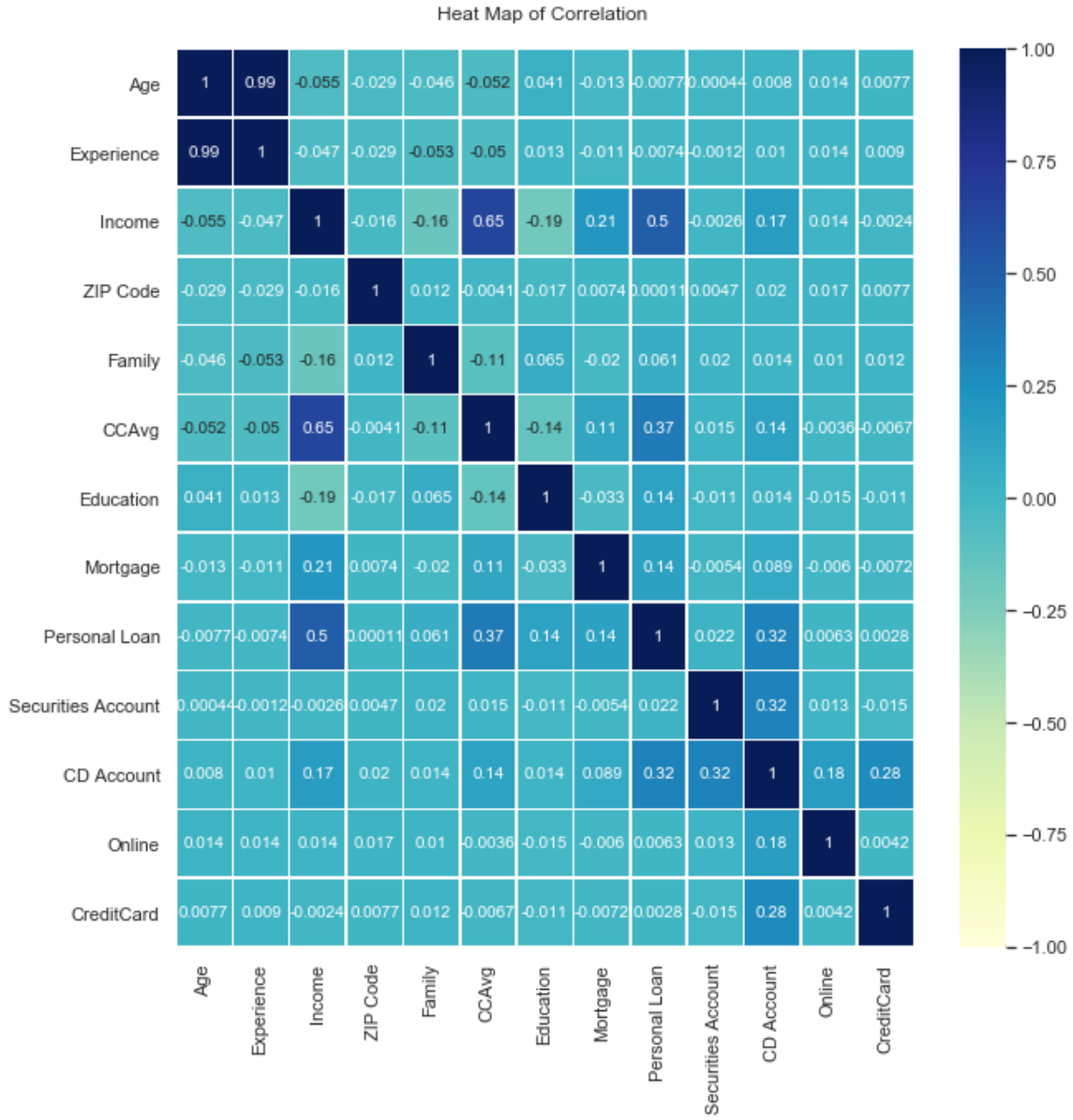
In [89]:

```
# HEAT MAP
# # Pairplot using

corr = data.corr();
print(corr)
# sns.heatmap(corr, annot = True);

fig, ax = plt.subplots()
fig.set_size_inches(10, 10)

plt.title('Heat Map of Correlation \n')

sns.heatmap(corr,annot=True,cmap='YlGnBu',vmin=-1,vmax=1, linewidths=.5, center=0);
```

	Age	Experience	Income	ZIP Code	Family
Age	1.000000	0.994215	-0.055269	-0.029216	-0.046418
Experience	0.994215	1.000000	-0.046574	-0.028626	-0.052563
Income	-0.055269	-0.046574	1.000000	-0.016410	-0.157501
ZIP Code	-0.029216	-0.028626	-0.016410	1.000000	0.011778
Family	-0.046418	-0.052563	-0.157501	0.011778	1.000000
CCAvg	-0.052012	-0.050077	0.645984	-0.004061	-0.109275
Education	0.041334	0.013152	-0.187524	-0.017377	0.064929
Mortgage	-0.012539	-0.010582	0.206806	0.007383	-0.020445
Personal Loan	-0.007726	-0.007413	0.502462	0.000107	0.061367
Securities Account	-0.000436	-0.001232	-0.002616	0.004704	0.019994
CD Account	0.008043	0.010353	0.169738	0.019972	0.014110
Online	0.013702	0.013898	0.014206	0.016990	0.010354
CreditCard	0.007681	0.008967	-0.002385	0.007691	0.011588

	CCAvg	Education	Mortgage	Personal Loan
Age	-0.052012	0.041334	-0.012539	-0.007726
Experience	-0.050077	0.013152	-0.010582	-0.007413
Income	0.645984	-0.187524	0.206806	0.502462
ZIP Code	-0.004061	-0.017377	0.007383	0.000107
Family	-0.109275	0.064929	-0.020445	0.061367
CCAvg	1.000000	-0.136124	0.109905	0.366889
Education	-0.136124	1.000000	-0.033327	0.136722
Mortgage	0.109905	-0.033327	1.000000	0.142095
Personal Loan	0.366889	0.136722	0.142095	1.000000
Securities Account	0.015086	-0.010812	-0.005411	0.021954
CD Account	0.136534	0.013934	0.089311	0.316355
Online	-0.003611	-0.015004	-0.005995	0.006278
CreditCard	-0.006689	-0.011014	-0.007231	0.002802

	Securities Account	CD Account	Online	CreditCard
Age	-0.000436	0.008043	0.013702	0.007681
Experience	-0.001232	0.010353	0.013898	0.008967
Income	-0.002616	0.169738	0.014206	-0.002385
ZIP Code	0.004704	0.019972	0.016990	0.007691
Family	0.019994	0.014110	0.010354	0.011588
CCAvg	0.015086	0.136534	-0.003611	-0.006689
Education	-0.010812	0.013934	-0.015004	-0.011014
Mortgage	-0.005411	0.089311	-0.005995	-0.007231
Personal Loan	0.021954	0.316355	0.006278	0.002802
Securities Account	1.000000	0.317034	0.012627	-0.015028
CD Account	0.317034	1.000000	0.175880	0.278644
Online	0.012627	0.175880	1.000000	0.004210
CreditCard	-0.015028	0.278644	0.004210	1.000000



Interpretation of correlations Correlations that have a value of more than 0.4 or less than -.4 are considered to be correlated For example Age and Experience have almost a perfect correlation of 0.994 ! The other existing significant correlatation is Income & CCAVe and Income and Personal Loan When plotting logistic or linerar regression we want our variables to be independent, because dependent variables give us well in case of age and experience its almost like one variable, and the result will be overweighted bias towards those 2 We may consider dropping one oe of those for a better weightet fit of independent uncorrelated variables

In [92]:

```
## Get Data Model Ready

# There are no Null Values hence there is no need to replace them with mean
# the ID column is usless and does not impact our analysis (adds noting in terms of cus
tomer buying  a personal loan,
# so it was dropped)
# the categorcal variables such as Personal Loan,Securities Account,CD Account, Online,
CreditCard, are either one or zero
# indicating true or false and hence there is no reason to replace them with a mean, as
the zero is a valid entry, not a
# missing entry.

#data=data.drop ('ID', axis=1)
data.head(2)
```

Out[92]:

	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securit Acco
0	25	1	49	91107	4	1.6	1	0	0	
1	45	19	34	90089	3	1.5	1	0	0	

In [ ]:

```

```

In [93]:

```
## Splitting Data into training and test
from sklearn.model_selection import train_test_split

# linear model is
# independant variables
X = data.drop(['Personal Loan'], axis=1)

# the dependent variable
y = data['Personal Loan']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,random_state=1
)

X_train.head()
```

Out[93]:

	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Securities Account	Ac
1334	47	22	35	94304	2	1.3	1	0	0	
4768	38	14	39	93118	1	2.0	2	0	0	
65	59	35	131	91360	1	3.8	1	0	0	
177	29	3	65	94132	4	1.8	2	244	0	
4489	39	13	21	95518	3	0.2	2	0	0	

LOGISTIC REGRESSION

In [94]:

```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
# fit the model first on training data

model = LogisticRegression (solver="liblinear") # not to get a warning sign
model.fit(X_train,y_train)

#predict the model on test  part of data
ypredictor= model.predict(X_test)

coef_df = pd.DataFrame(model.coef_)
coef_df ['intercept'] = model.intercept_
print (coef_df)
```

```
          0         1         2         3         4         5         6  \
0 -0.000528 -0.00201  0.036089 -0.000066  0.015165  0.009454  0.016447

          7         8         9        10        11  intercept
0  0.000833  0.000534  0.004681 -0.000177 -0.000067  -0.000106
```

In [96]:

```
# training score
model_score=model.score(X_train,y_train)
model_score
```

Out[96]:

```
0.9091428571428571
```

COMPARING TEST AND TRAIN DATA PREDICTABILITY SCORE The score shows that the model we built on train data, is just as effective on test data, which means its a good model. It has not been overfit (if train captures too much noise and model is built too perfectly for that particular specific training set rather than any such set of banking data). Nor is it underfit because it closely reflects what it should - a close score to test data). Result is a good model !

In [97]:

```
# testing score
model_score=model.score(X_test,y_test)
model_score
```

Out[97]:

```
0.9073333333333333
```

In [100]:

```
#RMSE
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mean_squared_error(y_test,model.predict(X_test))**0.5
```

Out[100]:

```
0.30441200151548997
```

In [ ]:

```

```

In [ ]:

```
The above is our R2 predictibility score and its 91 % right, only 9% wrong ! Excellent
 !

The training and testing data of R2 is on par so, the model we built on testing data is
still working well on testing data
so its a good model - th R2 for test data has not dropped significantly
RMSE - is
```

CONFUSION METRICS

In [103]:

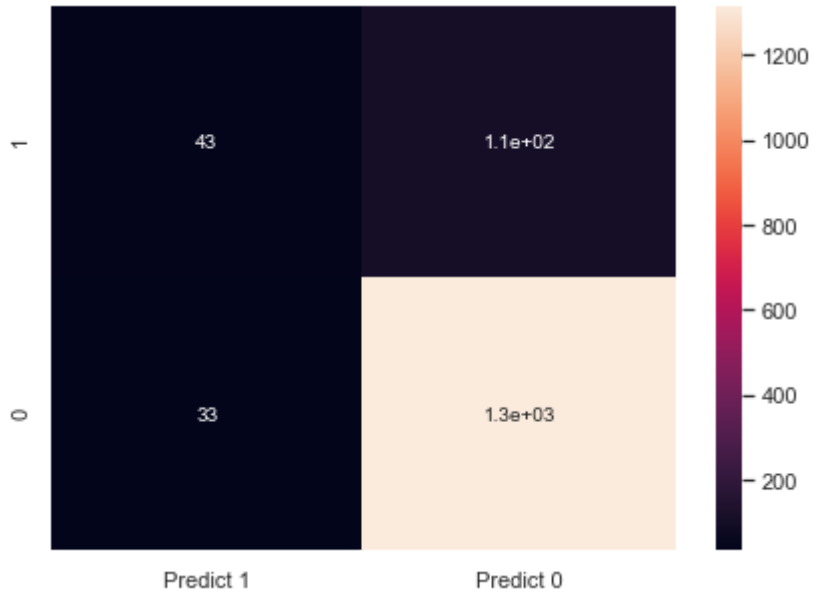
```

```

In [101]:

```
from sklearn import metrics
cm=metrics.confusion_matrix(y_test,ypredictor,labels=[1,0])

CM_DF=pd.DataFrame(cm, index =[i for i in ["1","0"]],
                       columns = [i for i in ["Predict 1","Predict 0"]])
plt.figure(figsize = (7,5))
sns.heatmap(CM_DF, annot=True)
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20640d88688>
```



In [ ]:

```
#F1 SCORE
=2* (PRECISION*RECALL) / (PRECISION +RECALL) = 2*(0.57*0.27)/ (0.57+0.27)= 0.36
```

The confusion matrix

True Positives (TP): Are 43 and are predicting higher than false negatives of 33 (Recall). So of all the qualified Personal Loan customers we are identifying more than half available ones

Precision is =43/43+33= 0.57 not bad, more than random Recall = 43/43+112= 27%, this is not attactive,we are still identyfying many non-qualified or interested customers as targeted Personal Loan borrowers

True Negatives (TN): its the largest number, 133 in the matrix and predicting negatives (no Personal Loan interest) correctly

False Positives (FP):incorrctly predicted 112 as Personal Loan clients

False Negatives (FN): incorrectly predicted 33 clients to be nit interested in a Personal loan but they were Overall the TP + TN are much higher than FP+FN, its a good model

SUMMARY Our models True Negative is high, but True Positive could be higher. Hence the model predicts well who is not interested in a personal loan, but can improve in identyfying who is interested.

Recall (Sensitivity) is =43/43+112= 0.27 not bad, more than random Precision is = 43/43+33= 27%, this is not attactive,we are still identyfying many non-qualified or non-interested customers as targeted Personal Loan borrowers

Specificity (true negative rate) -tn/tn+fp =133/133+33= 0.80, this is high, meaning we are identifying well the clients who are NOT candidates for a Personal Loan

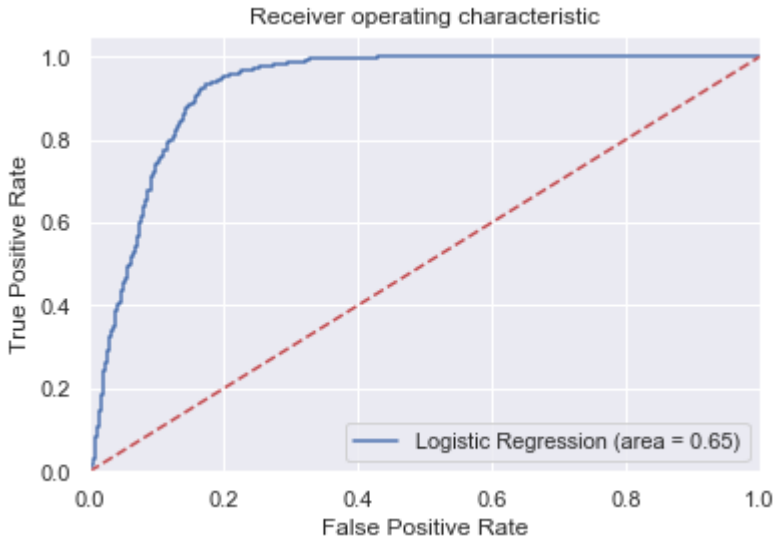
Accuracy- tp+tn/tp+tn+fp+fn = 43+133 / 43+133+112+33 = 176/321= 0.54 , the model is predicting correct values above random 50/50 flip, but has room to improve

F1 SCORE IS 0.36, and the higher it is the better the model so its not satisfying in this respect

AREA UNDER THE CURVE

In [104]:

```
#AUC ROC curve
from sklearn.metrics import confusion_matrix, recall_score, precision_score, f1_score,
roc_auc_score, accuracy_score, log_loss

logreg = LogisticRegression(random_state=42);

logreg.fit(X_train, y_train);
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve


logit_roc_auc = roc_auc_score(y_train, logreg.predict(X_train))

fpr, tpr, thresholds = roc_curve(y_train, logreg.predict_proba(X_train)[:,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 [109]:

```
AOC=logit_roc_auc
Gini_Coef=2*AOC-1

print ( 'The area under the curve is', AOC)
print ( 'The Gini Coeficient  is', Gini_Coef)
```

```
The area under the curve is 0.6463269074750772
The Gini Coeficient  is 0.2926538149501543
```

AOC ANALYSIS The model shows that we are getting values plotted much better than half (diagonal average 50/50) where the model plots along the curve a lot of TPR than FPR, meaning that only past 90% of predicting True Positives (Personal Loan candidates) do we start getting an increase of False Positives(identified as Personal Loan candidates) but are not.

Hence the graph looks very good !

How may my model work better ?

We could add some error penalties to variables to change their "weight" and see if the we an even more improved score ... Potentialy we can ask for more variables to be introduced like - if married, is the spouse working - and if so one can assume this would inflience the clients potential for a personal loan- hence more independent variables to add to equation

Because Age and Experience are highly correlated, we could consider dropping one, to have a pure set of independent variables, as keeping both may give us overweighted results with both of these included...

Give Business understanding of your model !

We are tryign to help identify potential clients for a Personal Loan within a banks general datbase. We have a set of parameters (variables) and have to determine if these parameters influence the outcome - hence give predictability. For example certainly Income is correlated (.5) to personal Loan, hence we can assume just from that factor that clients with high income have a higher chance of taking out a persona loan. By additing the other variables we have and studying the data, we can make predctions and identify who is a candidate for a personal loan for the bank, hence giving the bank more business in precise upsels.

In case of loans where our loss of non-repayment is greater than our opportunity cost of failing to lend and earn interest to a qualified buyer, we may set the threshold, higher and perhaps only lend to those who quality not above 50% but those who qualify above 80% , hence set the parameters of the model to reflect that threshold and identify such customers from the database.