Background and Context

AllLife Bank is a US bank that has a growing customer base. The majority of these customers are liability customers (depositors) with varying sizes of deposits. The number of customers who are also borrowers (asset customers) is quite small.

Objective :

the bank is interested in expanding this base rapidly to bring in more loan business and in the process, earn more through the interest on loans. In particular, the management wants to explore ways of converting its liability customers to personal loan customers (while retaining them as depositors).

1- build LogisticRegression Model To predict whether a liability customer will buy a personal loan or not.

2-Which variables are most significant.

3-Which segment of customers should be targeted more.

Data Dictionary

- ID: Customer ID
- Age: Customer’s age in completed years
- Experience: #years of professional experience
- Income: Annual income of the customer (in thousand dollars)
- ZIP Code: Home Address ZIP code.
- Family: the Family size of the customer
- CCAvg: Average spending on credit cards per month (in thousand dollars)
- Education: Education Level. 1: Undergrad; 2: Graduate;3: Advanced/Professional
- Mortgage: Value of house mortgage if any. (in thousand dollars)
- Personal_Loan: Did this customer accept the personal loan offered in the last campaign?
- Securities_Account: Does the customer have securities account with the bank?
- CD_Account: Does the customer have a certificate of deposit (CD) account with the bank?
- Online: Do customers use internet banking facilities?
- CreditCard: Does the customer use a credit card issued by any other Bank (excluding All life Bank)?

In [188…
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import zipcodes as zcode # to get zipcodes

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
import warnings

import statsmodels.api as sm
#--Sklearn library--
# Sklearn package's randomized data splitting function
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn import metrics
#AUC ROC curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve

from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay #to plot confusion
from sklearn.metrics import plot_confusion_matrix

from sklearn.linear_model import LogisticRegression #to build the model
from sklearn.tree import DecisionTreeClassifier#to build the model

pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.max_rows', 300)
pd.set_option('display.max_colwidth',400)
pd.set_option('display.float_format', lambda x: '%.5f' % x)
# To supress numerical display in scientific notations
warnings.filterwarnings('ignore') # To supress warnings
 # set the background for the graphs
plt.style.use('ggplot')
```

In [2]:
```
#import csv dataset file
data= pd.read_csv(r'C:\Users\lostsemsem\Desktop\all life bank\Loan_Modelling.csv')
```

In [3]:
```
# let us make another copy of data
df = data.copy()
```

In [4]:
```
np.random.seed(1)  # To get the same random results every time
df.sample(n=10)
```

Out[4]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan
2764	2765	31	5	84	91320	1	2.90000	3	105	0
4767	4768	35	9	45	90639	3	0.90000	1	101	0
3814	3815	34	9	35	94304	3	1.30000	1	0	0
3499	3500	49	23	114	94550	1	0.30000	1	286	0
2735	2736	36	12	70	92131	3	2.60000	2	165	0
3922	3923	31	4	20	95616	4	1.50000	2	0	0
2701	2702	50	26	55	94305	1	1.60000	2	0	0
1179	1180	36	11	98	90291	3	1.20000	3	0	0
932	933	51	27	112	94720	3	1.80000	2	0	0
792	793	41	16	98	93117	1	4.00000	3	0	0

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIPCode             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 [7]:
```
# check number of rows and columns
df.shape
```

Out[7]:
```
(5000, 14)
```

In [8]:
```
# cheking the number of unique values in each column
df.nunique()
```

Out[8]:
```
ID                    5000
Age                     45
Experience              47
Income                 162
ZIPCode                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 [9]:
```
#check for duplicated values
df.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
#Check Nulls
df.isnull().sum()
```

Out[10]:
```
ID                    0
Age                   0
Experience            0
Income                0
```

```
ZIPCode               0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal_Loan         0
Securities_Account    0
CD_Account            0
Online                0
CreditCard            0
dtype: int64
```

In [11]:
```
df.describe()
```

Out[11]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education
count	5000.00000	5000.00000	5000.00000	5000.00000	5000.00000	5000.00000	5000.00000	5000.00000
mean	2500.50000	45.33840	20.10460	73.77420	93169.25700	2.39640	1.93794	1.88100
std	1443.52000	11.46317	11.46795	46.03373	1759.45509	1.14766	1.74766	0.83987
min	1.00000	23.00000	-3.00000	8.00000	90005.00000	1.00000	0.00000	1.00000
25%	1250.75000	35.00000	10.00000	39.00000	91911.00000	1.00000	0.70000	1.00000
50%	2500.50000	45.00000	20.00000	64.00000	93437.00000	2.00000	1.50000	2.00000
75%	3750.25000	55.00000	30.00000	98.00000	94608.00000	3.00000	2.50000	3.00000
max	5000.00000	67.00000	43.00000	224.00000	96651.00000	4.00000	10.00000	3.00000

In [12]:
```
# Checking Experience Negative values
df.sort_values(by=["Age"], ascending=True).head(5)
```

Out[12]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan
3157	3158	23	-1	13	94720	4	1.00000	1	84	0
4285	4286	23	-3	149	93555	2	7.20000	1	0	0
2618	2619	23	-3	55	92704	3	2.40000	2	145	0
4411	4412	23	-2	75	90291	2	1.80000	2	0	0
2430	2431	23	-1	73	92120	4	2.60000	1	0	0

In [13]:
```
# Checking Experience Negative values
df.sort_values(by=["Age"], ascending=False).head(5)
```

Out[13]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan
4468	4469	67	42	51	94117	3	2.20000	1	0	0
1480	1481	67	42	32	93943	1	1.10000	3	0	0
4451	4452	67	41	18	92130	2	0.40000	1	0	0
14	15	67	41	112	91741	1	2.00000	1	0	0
1859	1860	67	41	20	91741	2	0.40000	1	80	0

In [14]:
```
# Checking the negative values
df[df['Experience'] < 0]['Experience'].value_counts()
```

Out[14]:
```
-1    33
-2    15
-3     4
Name: Experience, dtype: int64
```

In [15]:
```
# Total records of negative experience
df[df['Experience'] < 0]['Experience'].count()
```

Out[15]:
```
52
```

Observations on the variables:

- There are no Missing or duplicated values .
- 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.
- The Experience Variable contains a Negative values which may happened because of Typo error so we will apply abslute value function to revert the negative values back to it's abslute value.
- ZipCode column contains ZipCodes for each customer (467 unique value) which could be converted to counties with python zipcode library to reduce the dimensionality in our dataset and indentifying the regions where the customers have a higher probability to take a personal loan.
- Personal_Loan, Securities_Account, CD_Account, 'Online', 'CreditCard' ,Education are of int/object type, we can change them to category type to reduce the dataspace required.
- The variables can be divided accordingly :

1- The binary category have five variables as below:

- Personal Loan - Did this customer accept the personal loan offered in the last campaign? This is our target variable.
- 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?
- Credit_Card - Does the customer use a credit card issued by UniversalBank?

2- Numerical Continous variables are as below:

- Age - Age of the customer.

- Experience - Years of experience.
- Income - Annual income in dollars.
- CCAvg - Average credit card spending.
- Mortage - Value of House Mortgage.

3-Ordinal Categorical Variables are:

- Family - Family size of the customer.
- Education - education level of the customer.

4- The nominal variable is :

- ID
- Zip Code

Data Preprocessing :

In [16]:
```
# Processing Experience Column
df.loc[df['Experience']<0,'Experience']=np.abs(df['Experience'])
```

In [17]:
```
# Checking Experience Negative values
df.sort_values(by=["Age"], ascending=True).head(5)
```

Out[17]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan
3157	3158	23	1	13	94720	4	1.00000	1	84	0
4285	4286	23	3	149	93555	2	7.20000	1	0	0
2618	2619	23	3	55	92704	3	2.40000	2	145	0
4411	4412	23	2	75	90291	2	1.80000	2	0	0
2430	2431	23	1	73	92120	4	2.60000	1	0	0

Processing ZIPCode column

In [18]:
```
# get unique zipcodes
list_zipcode=df.ZIPCode.unique()
```

In [19]:
```
# Ceating a dictionary of counties using library zipcode and matching method.
dict_zip={}
for zipcode in list_zipcode:
    my_city_county = zcode.matching(zipcode.astype('str'))
    if len(my_city_county)==1: # if  zipcode is present then get county else, assign zi
        county=my_city_county[0].get('county')
    else:
        county=zipcode
```

```
    dict_zip.update({zipcode:county})
dict_zip
```

Out[19]:

```
{91107: 'Los Angeles County',
 90089: 'Los Angeles County',
 94720: 'Alameda County',
 94112: 'San Francisco County',
 91330: 'Los Angeles County',
 92121: 'San Diego County',
 91711: 'Los Angeles County',
 93943: 'Monterey County',
 93023: 'Ventura County',
 94710: 'Alameda County',
 90277: 'Los Angeles County',
 93106: 'Santa Barbara County',
 94920: 'Marin County',
 91741: 'Los Angeles County',
 95054: 'Santa Clara County',
 95010: 'Santa Cruz County',
 94305: 'Santa Clara County',
 91604: 'Los Angeles County',
 94015: 'San Mateo County',
 90095: 'Los Angeles County',
 91320: 'Ventura County',
 95521: 'Humboldt County',
 95064: 'Santa Cruz County',
 90064: 'Los Angeles County',
 94539: 'Alameda County',
 94104: 'San Francisco County',
 94117: 'San Francisco County',
 94801: 'Contra Costa County',
 94035: 'Santa Clara County',
 92647: 'Orange County',
 95814: 'Sacramento County',
 94114: 'San Francisco County',
 94115: 'San Francisco County',
 92672: 'Orange County',
 94122: 'San Francisco County',
 90019: 'Los Angeles County',
 95616: 'Yolo County',
 94065: 'San Mateo County',
 95014: 'Santa Clara County',
 91380: 'Los Angeles County',
 95747: 'Placer County',
 92373: 'San Bernardino County',
 92093: 'San Diego County',
 94005: 'San Mateo County',
 90245: 'Los Angeles County',
 95819: 'Sacramento County',
 94022: 'Santa Clara County',
 90404: 'Los Angeles County',
 93407: 'San Luis Obispo County',
 94523: 'Contra Costa County',
 90024: 'Los Angeles County',
 91360: 'Ventura County',
 95670: 'Sacramento County',
 95123: 'Santa Clara County',
 90045: 'Los Angeles County',
 91335: 'Los Angeles County',
 93907: 'Monterey County',
 92007: 'San Diego County',
 94606: 'Alameda County',
 94611: 'Alameda County',
 94901: 'Marin County',
 92220: 'Riverside County',
```

```
93305: 'Kern County',
95134: 'Santa Clara County',
94612: 'Alameda County',
92507: 'Riverside County',
91730: 'San Bernardino County',
94501: 'Alameda County',
94303: 'San Mateo County',
94105: 'San Francisco County',
94550: 'Alameda County',
92612: 'Orange County',
95617: 'Yolo County',
92374: 'San Bernardino County',
94080: 'San Mateo County',
94608: 'Alameda County',
93555: 'Kern County',
93311: 'Kern County',
94704: 'Alameda County',
92717: 92717,
92037: 'San Diego County',
95136: 'Santa Clara County',
94542: 'Alameda County',
94143: 'San Francisco County',
91775: 'Los Angeles County',
92703: 'Orange County',
92354: 'San Bernardino County',
92024: 'San Diego County',
92831: 'Orange County',
92833: 'Orange County',
94304: 'Santa Clara County',
90057: 'Los Angeles County',
92130: 'San Diego County',
91301: 'Los Angeles County',
92096: 'San Diego County',
92646: 'Orange County',
92182: 'San Diego County',
92131: 'San Diego County',
93720: 'Fresno County',
90840: 'Los Angeles County',
95035: 'Santa Clara County',
93010: 'Ventura County',
94928: 'Sonoma County',
95831: 'Sacramento County',
91770: 'Los Angeles County',
90007: 'Los Angeles County',
94102: 'San Francisco County',
91423: 'Los Angeles County',
93955: 'Monterey County',
94107: 'San Francisco County',
92834: 'Orange County',
93117: 'Santa Barbara County',
94551: 'Alameda County',
94596: 'Contra Costa County',
94025: 'San Mateo County',
94545: 'Alameda County',
95053: 'Santa Clara County',
90036: 'Los Angeles County',
91125: 'Los Angeles County',
95120: 'Santa Clara County',
94706: 'Alameda County',
95827: 'Sacramento County',
90503: 'Los Angeles County',
90250: 'Los Angeles County',
95817: 'Sacramento County',
95503: 'Humboldt County',
93111: 'Santa Barbara County',
```

```
               94132: 'San Francisco County',
               95818: 'Sacramento County',
               91942: 'San Diego County',
               90401: 'Los Angeles County',
               93524: 'Kern County',
               95133: 'Santa Clara County',
               92173: 'San Diego County',
               94043: 'Santa Clara County',
               92521: 'Riverside County',
               92122: 'San Diego County',
               93118: 'Santa Barbara County',
               92697: 'Orange County',
               94577: 'Alameda County',
               91345: 'Los Angeles County',
               94123: 'San Francisco County',
               92152: 'San Diego County',
               91355: 'Los Angeles County',
               94609: 'Alameda County',
               94306: 'Santa Clara County',
               96150: 'El Dorado County',
               94110: 'San Francisco County',
               94707: 'Alameda County',
               91326: 'Los Angeles County',
               90291: 'Los Angeles County',
               92807: 'Orange County',
               95051: 'Santa Clara County',
               94085: 'Santa Clara County',
               92677: 'Orange County',
               92614: 'Orange County',
               92626: 'Orange County',
               94583: 'Contra Costa County',
               92103: 'San Diego County',
               92691: 'Orange County',
               92407: 'San Bernardino County',
               90504: 'Los Angeles County',
               94002: 'San Mateo County',
               95039: 'Monterey County',
               94063: 'San Mateo County',
               94923: 'Sonoma County',
               95023: 'San Benito County',
               90058: 'Los Angeles County',
               92126: 'San Diego County',
               94118: 'San Francisco County',
               90029: 'Los Angeles County',
               92806: 'Orange County',
               94806: 'Contra Costa County',
               92110: 'San Diego County',
               94536: 'Alameda County',
               90623: 'Orange County',
               92069: 'San Diego County',
               92843: 'Orange County',
               92120: 'San Diego County',
               95605: 'Yolo County',
               90740: 'Orange County',
               91207: 'Los Angeles County',
               95929: 'Butte County',
               93437: 'Santa Barbara County',
               90630: 'Orange County',
               90034: 'Los Angeles County',
               90266: 'Los Angeles County',
               95630: 'Sacramento County',
               93657: 'Fresno County',
               92038: 'San Diego County',
               91304: 'Los Angeles County',
               92606: 'Orange County',
```

```
92192: 'San Diego County',
90745: 'Los Angeles County',
95060: 'Santa Cruz County',
94301: 'Santa Clara County',
92692: 'Orange County',
92101: 'San Diego County',
94610: 'Alameda County',
90254: 'Los Angeles County',
94590: 'Solano County',
92028: 'San Diego County',
92054: 'San Diego County',
92029: 'San Diego County',
93105: 'Santa Barbara County',
91941: 'San Diego County',
92346: 'San Bernardino County',
94402: 'San Mateo County',
94618: 'Alameda County',
94904: 'Marin County',
93077: 93077,
95482: 'Mendocino County',
91709: 'San Bernardino County',
91311: 'Los Angeles County',
94509: 'Contra Costa County',
92866: 'Orange County',
91745: 'Los Angeles County',
94111: 'San Francisco County',
94309: 'Santa Clara County',
90073: 'Los Angeles County',
92333: 'San Bernardino County',
90505: 'Los Angeles County',
94998: 'Marin County',
94086: 'Santa Clara County',
94709: 'Alameda County',
95825: 'Sacramento County',
90509: 'Los Angeles County',
93108: 'Santa Barbara County',
94588: 'Alameda County',
91706: 'Los Angeles County',
92109: 'San Diego County',
92068: 'San Diego County',
95841: 'Sacramento County',
92123: 'San Diego County',
91342: 'Los Angeles County',
90232: 'Los Angeles County',
92634: 92634,
91006: 'Los Angeles County',
91768: 'Los Angeles County',
90028: 'Los Angeles County',
92008: 'San Diego County',
95112: 'Santa Clara County',
92154: 'San Diego County',
92115: 'San Diego County',
92177: 'San Diego County',
90640: 'Los Angeles County',
94607: 'Alameda County',
92780: 'Orange County',
90009: 'Los Angeles County',
92518: 'Riverside County',
91007: 'Los Angeles County',
93014: 'Santa Barbara County',
94024: 'Santa Clara County',
90027: 'Los Angeles County',
95207: 'San Joaquin County',
90717: 'Los Angeles County',
94534: 'Solano County',
```

```
94010: 'San Mateo County',
91614: 'Los Angeles County',
94234: 'Sacramento County',
90210: 'Los Angeles County',
95020: 'Santa Clara County',
92870: 'Orange County',
92124: 'San Diego County',
90049: 'Los Angeles County',
94521: 'Contra Costa County',
95678: 'Placer County',
95045: 'San Benito County',
92653: 'Orange County',
92821: 'Orange County',
90025: 'Los Angeles County',
92835: 'Orange County',
91910: 'San Diego County',
94701: 'Alameda County',
91129: 'Los Angeles County',
90071: 'Los Angeles County',
96651: 96651,
94960: 'Marin County',
91902: 'San Diego County',
90033: 'Los Angeles County',
95621: 'Sacramento County',
90037: 'Los Angeles County',
90005: 'Los Angeles County',
93940: 'Monterey County',
91109: 'Los Angeles County',
93009: 'Ventura County',
93561: 'Kern County',
95126: 'Santa Clara County',
94109: 'San Francisco County',
93107: 'Santa Barbara County',
94591: 'Solano County',
92251: 'Imperial County',
92648: 'Orange County',
92709: 'Orange County',
91754: 'Los Angeles County',
92009: 'San Diego County',
96064: 'Siskiyou County',
91103: 'Los Angeles County',
91030: 'Los Angeles County',
90066: 'Los Angeles County',
95403: 'Sonoma County',
91016: 'Los Angeles County',
95348: 'Merced County',
91950: 'San Diego County',
95822: 'Sacramento County',
94538: 'Alameda County',
92056: 'San Diego County',
93063: 'Ventura County',
91040: 'Los Angeles County',
92661: 'Orange County',
94061: 'San Mateo County',
95758: 'Sacramento County',
96091: 'Trinity County',
94066: 'San Mateo County',
94939: 'Marin County',
95138: 'Santa Clara County',
95762: 'El Dorado County',
92064: 'San Diego County',
94708: 'Alameda County',
92106: 'San Diego County',
92116: 'San Diego County',
91302: 'Los Angeles County',
```

```
                 90048: 'Los Angeles County',
                 90405: 'Los Angeles County',
                 92325: 'San Bernardino County',
                 91116: 'Los Angeles County',
                 92868: 'Orange County',
                 90638: 'Los Angeles County',
                 90747: 'Los Angeles County',
                 93611: 'Fresno County',
                 95833: 'Sacramento County',
                 91605: 'Los Angeles County',
                 92675: 'Orange County',
                 90650: 'Los Angeles County',
                 95820: 'Sacramento County',
                 90018: 'Los Angeles County',
                 93711: 'Fresno County',
                 95973: 'Butte County',
                 92886: 'Orange County',
                 95812: 'Sacramento County',
                 91203: 'Los Angeles County',
                 91105: 'Los Angeles County',
                 95008: 'Santa Clara County',
                 90016: 'Los Angeles County',
                 90035: 'Los Angeles County',
                 92129: 'San Diego County',
                 90720: 'Orange County',
                 94949: 'Marin County',
                 90041: 'Los Angeles County',
                 95003: 'Santa Cruz County',
                 95192: 'Santa Clara County',
                 91101: 'Los Angeles County',
                 94126: 'San Francisco County',
                 90230: 'Los Angeles County',
                 93101: 'Santa Barbara County',
                 91365: 'Los Angeles County',
                 91367: 'Los Angeles County',
                 91763: 'San Bernardino County',
                 92660: 'Orange County',
                 92104: 'San Diego County',
                 91361: 'Ventura County',
                 90011: 'Los Angeles County',
                 90032: 'Los Angeles County',
                 95354: 'Stanislaus County',
                 94546: 'Alameda County',
                 92673: 'Orange County',
                 95741: 'Sacramento County',
                 95351: 'Stanislaus County',
                 92399: 'San Bernardino County',
                 90274: 'Los Angeles County',
                 94087: 'Santa Clara County',
                 90044: 'Los Angeles County',
                 94131: 'San Francisco County',
                 94124: 'San Francisco County',
                 95032: 'Santa Clara County',
                 90212: 'Los Angeles County',
                 93109: 'Santa Barbara County',
                 94019: 'San Mateo County',
                 95828: 'Sacramento County',
                 90086: 'Los Angeles County',
                 94555: 'Alameda County',
                 93033: 'Ventura County',
                 93022: 'Ventura County',
                 91343: 'Los Angeles County',
                 91911: 'San Diego County',
                 94803: 'Contra Costa County',
                 94553: 'Contra Costa County',
```

```
95211: 'San Joaquin County',
90304: 'Los Angeles County',
92084: 'San Diego County',
90601: 'Los Angeles County',
92704: 'Orange County',
92350: 'San Bernardino County',
94705: 'Alameda County',
93401: 'San Luis Obispo County',
90502: 'Los Angeles County',
94571: 'Solano County',
95070: 'Santa Clara County',
92735: 'Orange County',
95037: 'Santa Clara County',
95135: 'Santa Clara County',
94028: 'San Mateo County',
96003: 'Shasta County',
91024: 'Los Angeles County',
90065: 'Los Angeles County',
95405: 'Sonoma County',
95370: 'Tuolumne County',
93727: 'Fresno County',
92867: 'Orange County',
95821: 'Sacramento County',
94566: 'Alameda County',
95125: 'Santa Clara County',
94526: 'Contra Costa County',
94604: 'Alameda County',
96008: 'Shasta County',
93065: 'Ventura County',
96001: 'Shasta County',
95006: 'Santa Cruz County',
90639: 'Los Angeles County',
92630: 'Orange County',
95307: 'Stanislaus County',
91801: 'Los Angeles County',
94302: 'Santa Clara County',
91710: 'San Bernardino County',
93950: 'Monterey County',
90059: 'Los Angeles County',
94108: 'San Francisco County',
94558: 'Napa County',
93933: 'Monterey County',
92161: 'San Diego County',
94507: 'Contra Costa County',
94575: 'Contra Costa County',
95449: 'Mendocino County',
93403: 'San Luis Obispo County',
93460: 'Santa Barbara County',
95005: 'Santa Cruz County',
93302: 'Kern County',
94040: 'Santa Clara County',
91401: 'Los Angeles County',
95816: 'Sacramento County',
92624: 'Orange County',
95131: 'Santa Clara County',
94965: 'Marin County',
91784: 'San Bernardino County',
91765: 'Los Angeles County',
90280: 'Los Angeles County',
95422: 'Lake County',
95518: 'Humboldt County',
95193: 'Santa Clara County',
92694: 'Orange County',
90275: 'Los Angeles County',
90272: 'Los Angeles County',
```

```
91791: 'Los Angeles County',
92705: 'Orange County',
91773: 'Los Angeles County',
93003: 'Ventura County',
90755: 'Los Angeles County',
96145: 'Placer County',
94703: 'Alameda County',
96094: 'Siskiyou County',
95842: 'Sacramento County',
94116: 'San Francisco County',
90068: 'Los Angeles County',
94970: 'Marin County',
90813: 'Los Angeles County',
94404: 'San Mateo County',
94598: 'Contra Costa County'}
```

All counties code already satisfiesd expect for 96651,92634,93077,92717.

We can google them to find these missing counties.

In [20]:
```
dict_zip.update({92717:'Orange County'})
dict_zip.update({92634:'Orange County'})
df['County']=df['ZIPCode'].map(dict_zip)
df.County.nunique()
```

Out[20]:
```
40
```

Observation :

- 467 ZIPCodes had been assigned to 40 different counties.

In [23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 15 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   ZIPCode             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  
 14  County              5000 non-null   object 
dtypes: float64(1), int64(13), object(1)
memory usage: 586.1+ KB
```

In [25]:
```
# converting object type to category
category_columns = ['Personal_Loan', 'Securities_Account','Family', 'CD_Account', 'Onli
df[category_columns] = df[category_columns].astype('category')
```

In [26]:
```
# Dropping ID, and ZIPCode Columns
df.drop(['ID','ZIPCode'],axis=1,inplace=True)
```

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   Age                 5000 non-null   int64   
 1   Experience          5000 non-null   int64   
 2   Income              5000 non-null   int64   
 3   Family              5000 non-null   category
 4   CCAvg               5000 non-null   float64 
 5   Education           5000 non-null   category
 6   Mortgage            5000 non-null   int64   
 7   Personal_Loan       5000 non-null   category
 8   Securities_Account  5000 non-null   category
 9   CD_Account          5000 non-null   category
 10  Online              5000 non-null   category
 11  CreditCard          5000 non-null   category
 12  County              5000 non-null   category
dtypes: category(8), float64(1), int64(4)
memory usage: 236.8 KB
```

Expolratory Data Analysis

In [28]:
```
df.describe().T
```

Out[28]:

	count	mean	std	min	25%	50%	75%	max
Age	5000.00000	45.33840	11.46317	23.00000	35.00000	45.00000	55.00000	67.00000
Experience	5000.00000	20.13460	11.41519	0.00000	10.00000	20.00000	30.00000	43.00000
Income	5000.00000	73.77420	46.03373	8.00000	39.00000	64.00000	98.00000	224.00000
CCAvg	5000.00000	1.93794	1.74766	0.00000	0.70000	1.50000	2.50000	10.00000
Mortgage	5000.00000	56.49880	101.71380	0.00000	0.00000	0.00000	101.00000	635.00000

observations:

- The Age of customers in our dataset ranges between 23 to 67 with Average 45 years old.
- The Experience ranges from non experienced up to 43 years of experience.
- The Income ranges between 8k per year up to 224k with average Income 73.77k.
- The Average usage on credit cards is 1.93k monthly through our dataset with max of 10k per month.

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

Out[30]:

	Age	Experience	Income	CCAvg	Mortgage
Age	1.00000	0.99399	-0.05527	-0.05201	-0.01254
Experience	0.99399	1.00000	-0.04688	-0.04974	-0.01110
Income	-0.05527	-0.04688	1.00000	0.64598	0.20681
CCAvg	-0.05201	-0.04974	0.64598	1.00000	0.10990
Mortgage	-0.01254	-0.01110	0.20681	0.10990	1.00000

Observations:

- Age and Experience are extremly correlated on 0.99, Hence we can drop one of them to avoid the multicollinearity between the variables.
- There is Significant high Correlation between Income and CCAvg on 0.645.
- there is low correlation between Mortgage and Income on 0.2.

Let's Calculclate the Coversion rate for Personal Loan(our dependent variable).

In [33]:
```
df['Personal_Loan'] = df['Personal_Loan'].astype('int')
```

In [34]:
```
# calculating number of trues and falses in our dependent variable
n_true = len(df.loc[df['Personal_Loan'] == True])
n_false = len(df.loc[df['Personal_Loan'] == False])
print("Number of customers who took Personal Loan: {0} ({1:2.2f}%)".format(n_true, (n_t
print("Number of customers who didn't take Personal Loan: {0} ({1:2.2f}%)".format(n_fal
```

```
Number of customers who took Personal Loan: 480 (9.60%)
Number of customers who didn't take Personal Loan: 4520 (90.40%)
```

In [35]:
```
 print(df['County'].value_counts())
```

```
Los Angeles County        1095
San Diego County           568
Santa Clara County         563
Alameda County             500
Orange County              366
San Francisco County       257
San Mateo County           204
Sacramento County          184
Santa Barbara County       154
Yolo County                130
Monterey County            128
Ventura County             114
San Bernardino County      101
Contra Costa County         85
Santa Cruz County           68
Riverside County            56
Kern County                 54
Marin County                54
Solano County               33
San Luis Obispo County      33
Humboldt County             32
Sonoma County               28
Fresno County               26
Placer County               24
Butte County                19
Shasta County               18
El Dorado County            17
Stanislaus County           15
San Benito County           14
San Joaquin County          13
Mendocino County             8
Siskiyou County              7
Tuolumne County              7
96651                        6
Lake County                  4
Merced County                4
Trinity County               4
Imperial County              3
Napa County                  3
```

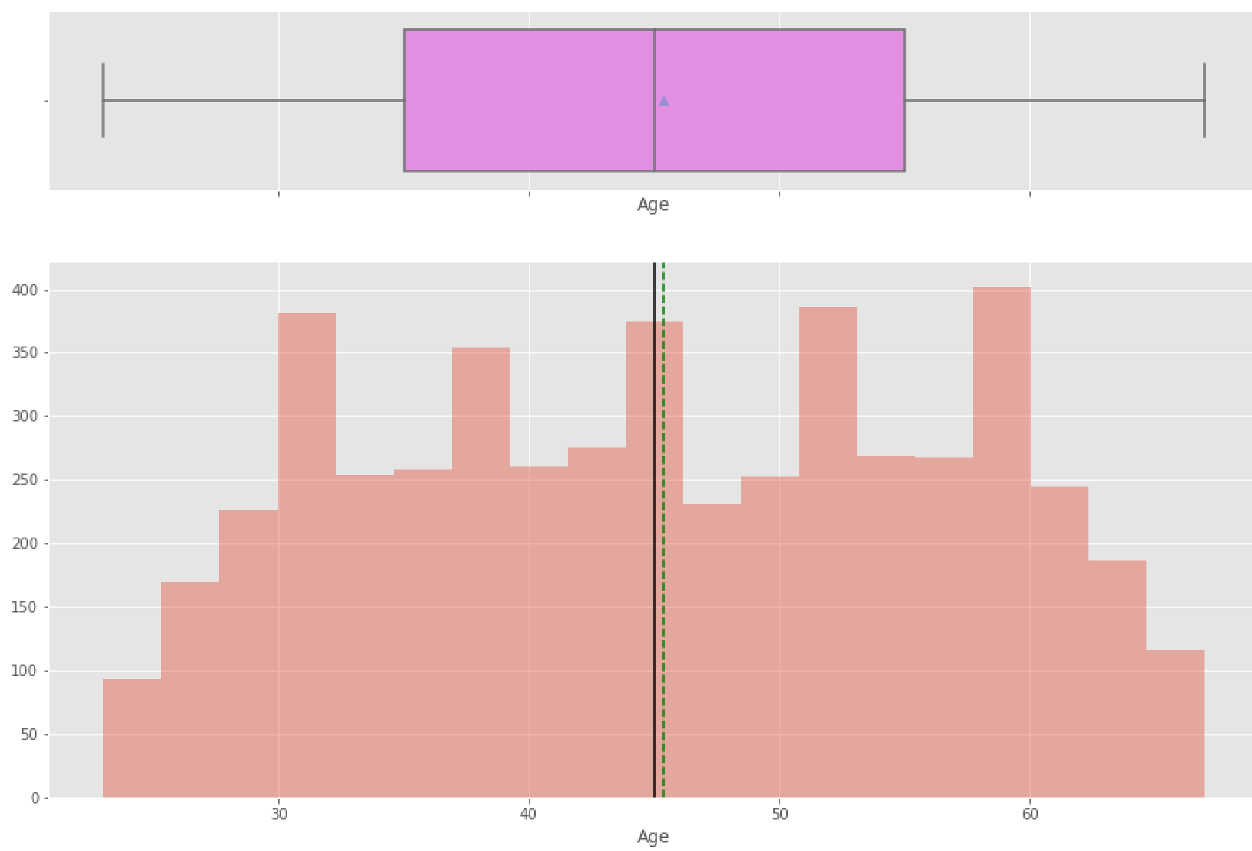
```
93077                  1
Name: County, dtype: int64
```

Univariate Analysis

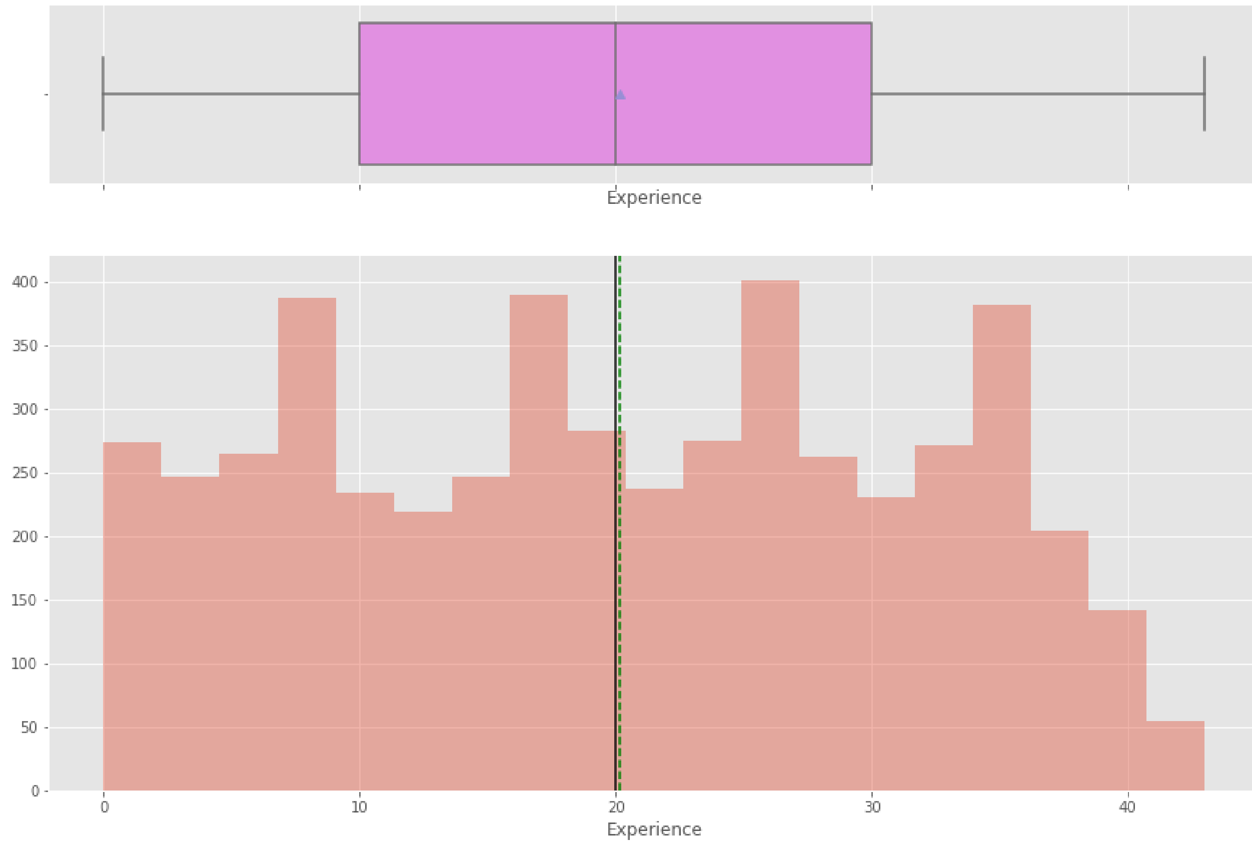
Let's Check the spread for each variable.

In [37]:
```
# While doing uni-variate analysis of numerical variables we want to study their centra
# and dispersion.
# Let us write a function that will help us create boxplot and histogram for any input
# variable.
# This function takes the numerical column as the input and returns the boxplots
# and histograms for the variable.
# Let us see if this help us write faster and cleaner code.
def histogram_boxplot(feature, figsize=(15,10), bins = None):
    """ Boxplot and histogram combined
    feature: 1-d feature array
    figsize: size of fig (default (9,8))
    bins: number of bins (default None / auto)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(nrows = 2, # Number of rows of the subplot g
                                           sharex = True, # x-axis will be shared among
                                           gridspec_kw = {"height_ratios": (.25, .75)},
                                           figsize = figsize
                                           ) # creating the 2 subplots
    sns.boxplot(feature, ax=ax_box2, showmeans=True, color='violet') # boxplot will be
    sns.distplot(feature, kde=F, ax=ax_hist2, bins=bins,palette="winter") if bins else
    ax_hist2.axvline(np.mean(feature), color='green', linestyle='--') # Add mean to the
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-') # Add median to
```

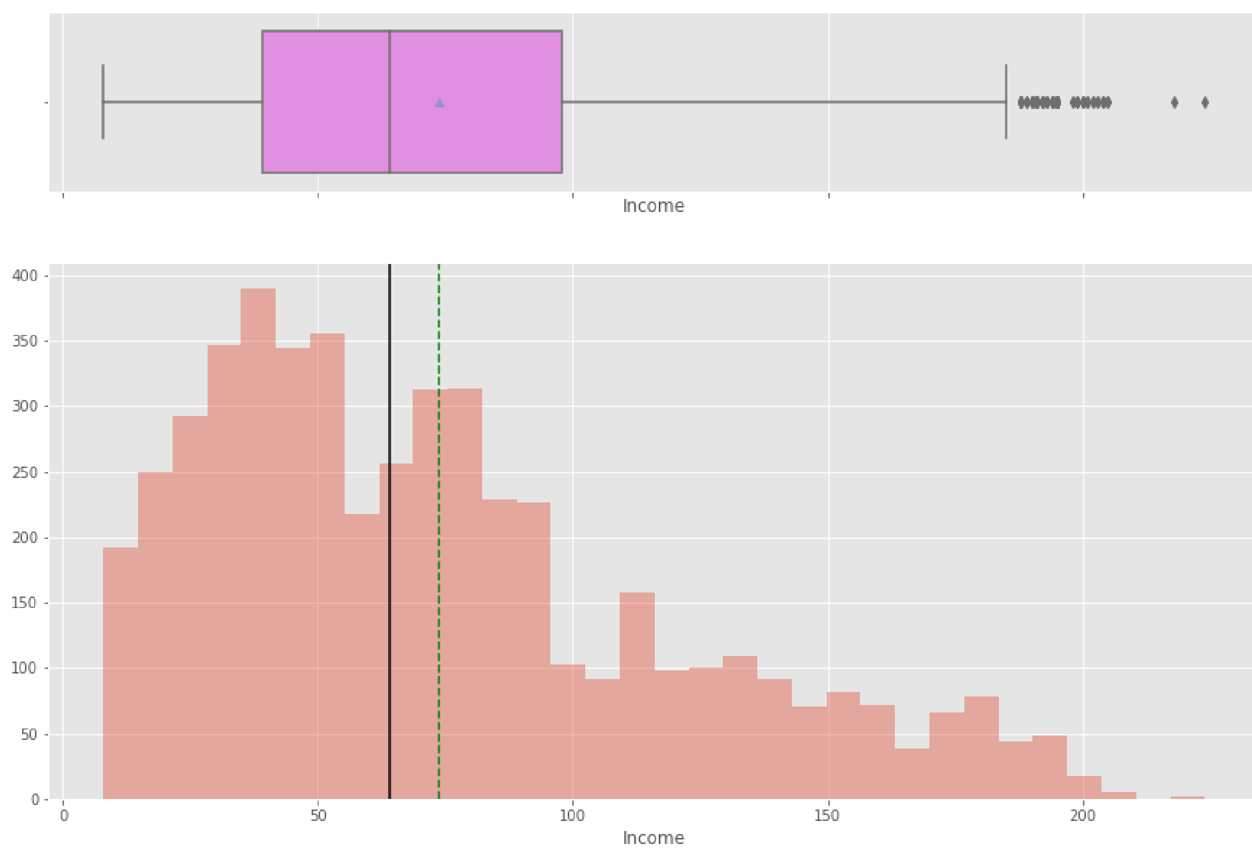
In [38]:
```
histogram_boxplot(df["Age"])
```



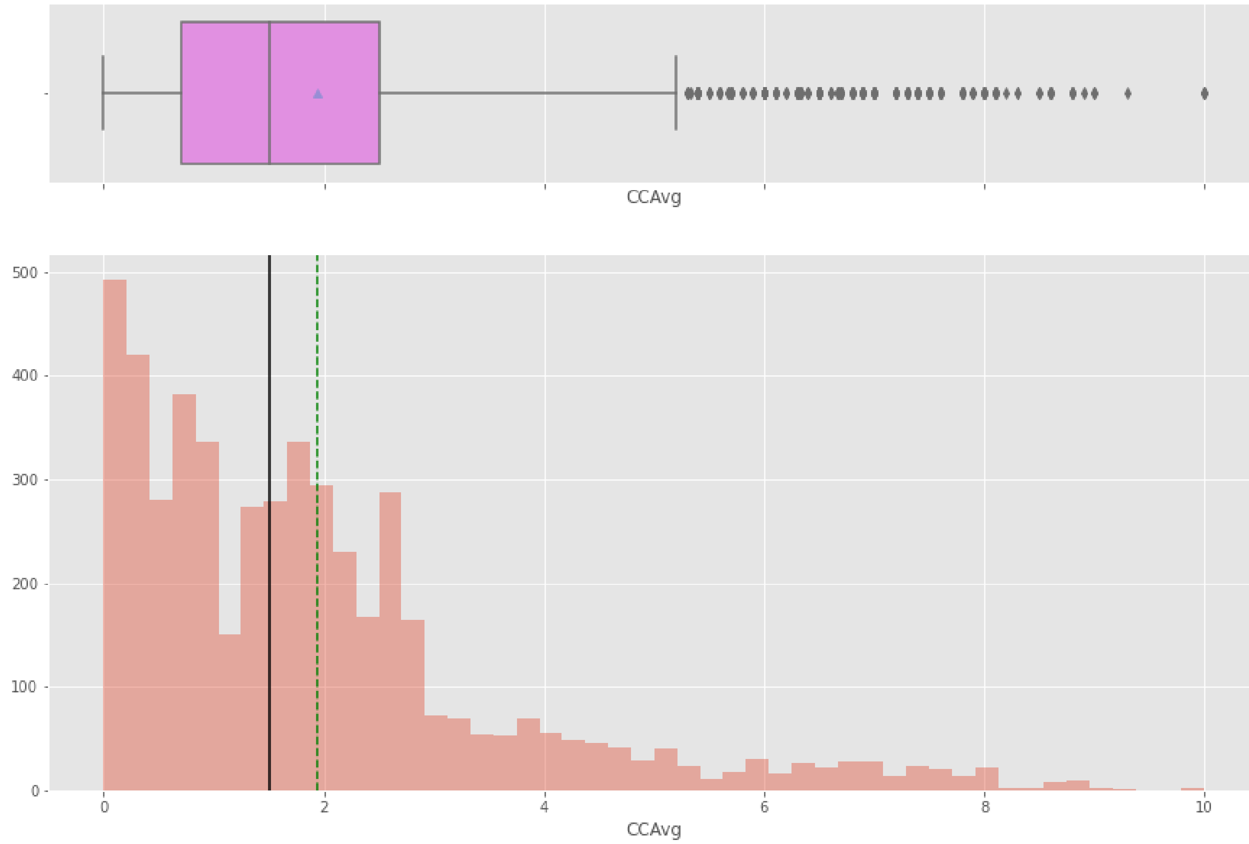
In [39]:
```
histogram_boxplot(df["Experience"])
```



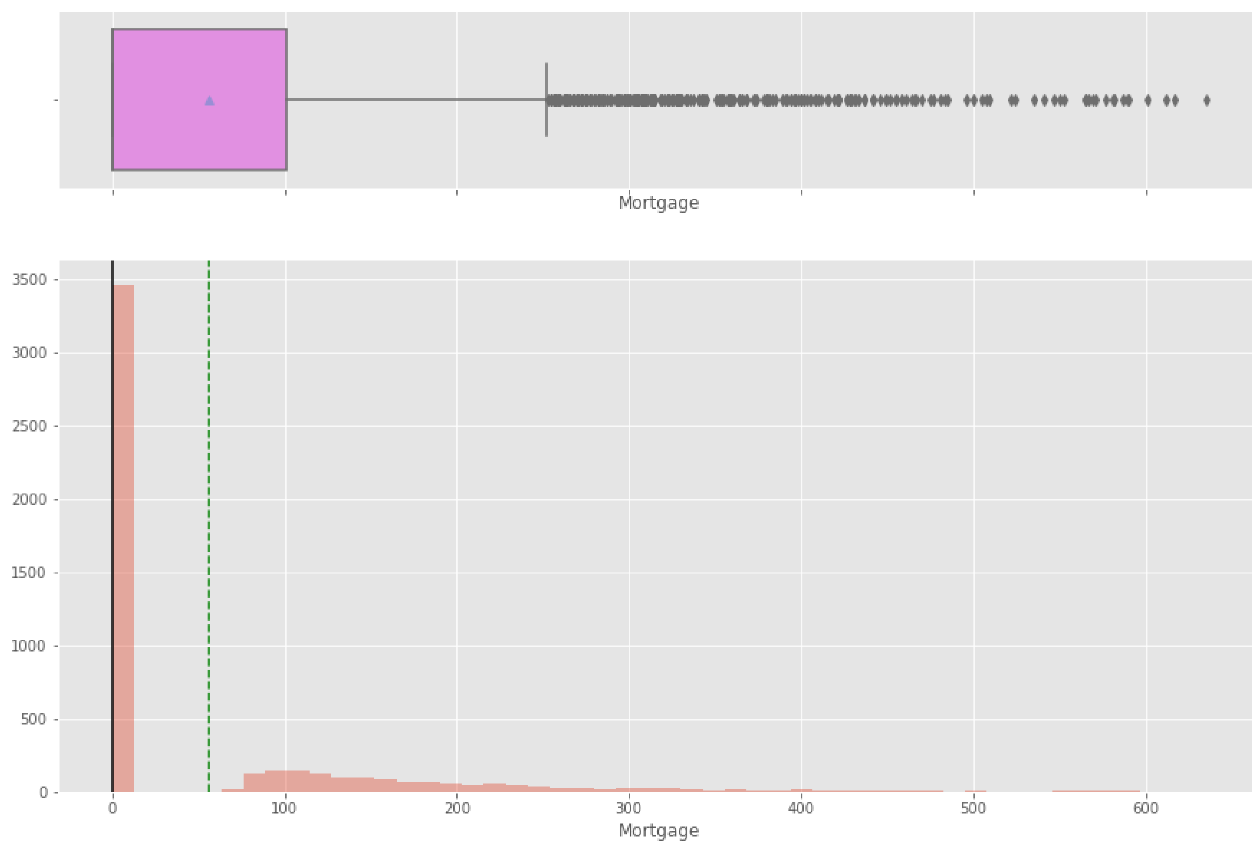
In [40]:
```
histogram_boxplot(df["Income"])
```



In [41]:
```
histogram_boxplot(df["CCAvg"])
```



In [42]:
```
histogram_boxplot(df["Mortgage"])
```



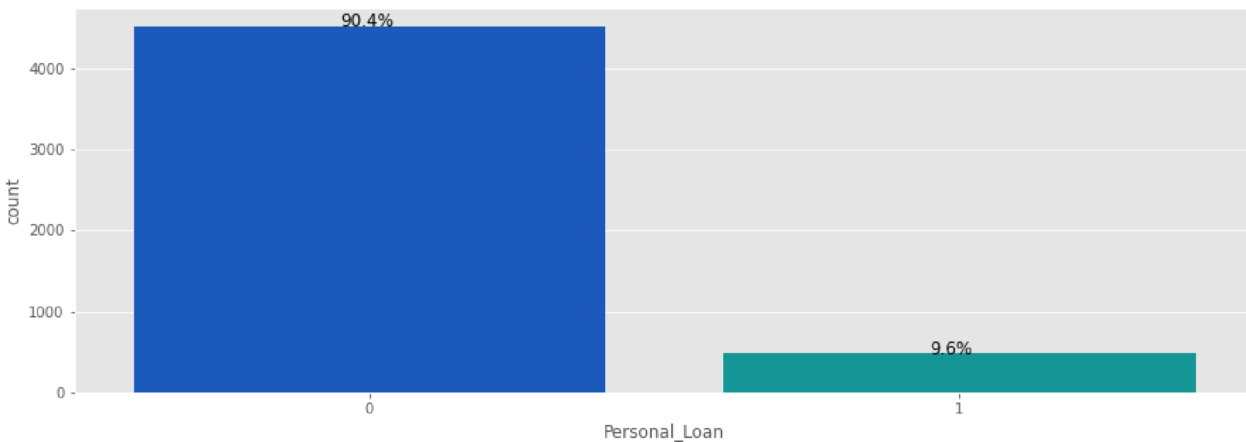
Observation:

- Age and Experience are both normally distributed and has almost the same distribution.

- Income , Mortgage and average usage on credits cards are all Right skewed with lots of outliers on the higher side.
- most of the customers in our dataset doesn't have a Mortgage with us

In [43]:
```
# Function to create barplots that indicate percentage for each category.

def perc_on_bar(plot, feature):
    '''
    plot
    feature: categorical feature
    the function won't work if a column is passed in hue parameter
    '''
    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total) # percentage of each
        x = p.get_x() + p.get_width() / 2 - 0.05 # width of the plot
        y = p.get_y() + p.get_height()           # hieght of the plot
        ax.annotate(percentage, (x, y), size = 12) # annotate the percantage
    plt.show() # show the plot
```

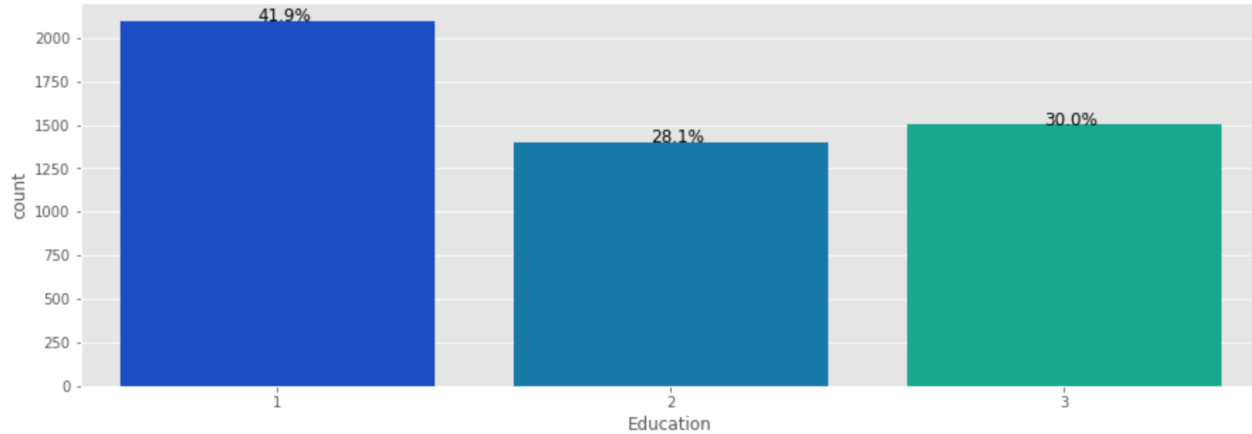
In [44]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(data["Personal_Loan"],palette='winter')
perc_on_bar(ax,data["Personal_Loan"])
```



Number of customers who took Personal Loan: 480 (9.60%)

Number of customers who didn't take Personal Loan: 4520 (90.40%

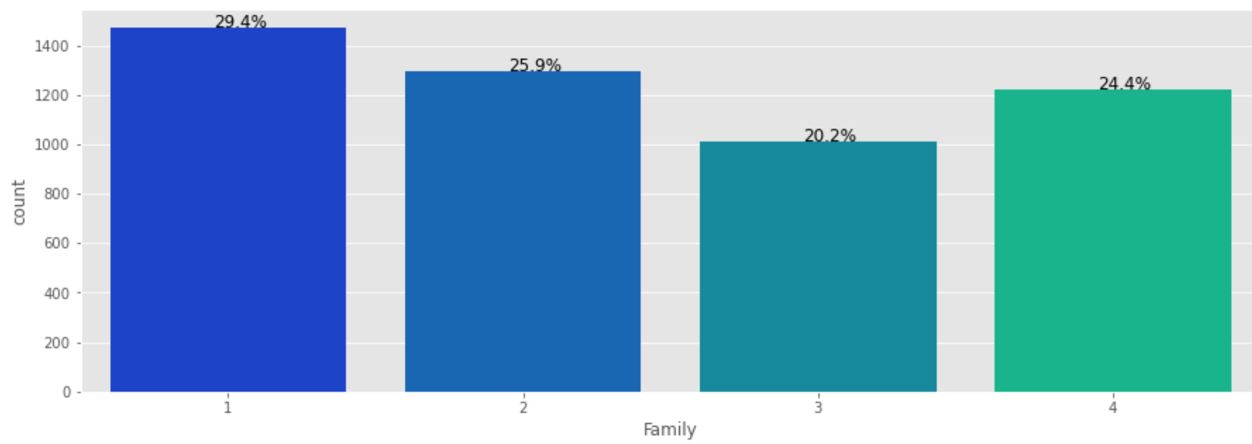
In [45]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(data["Education"],palette='winter')
perc_on_bar(ax,data["Education"])
```



Observation:

- 41.9 % of our customers are in an undergraduate stage while 28.1% are graduate and 30% has Advanced/Professional Degree.

In [46]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(data["Family"],palette='winter')
perc_on_bar(ax,data["Family"])
```

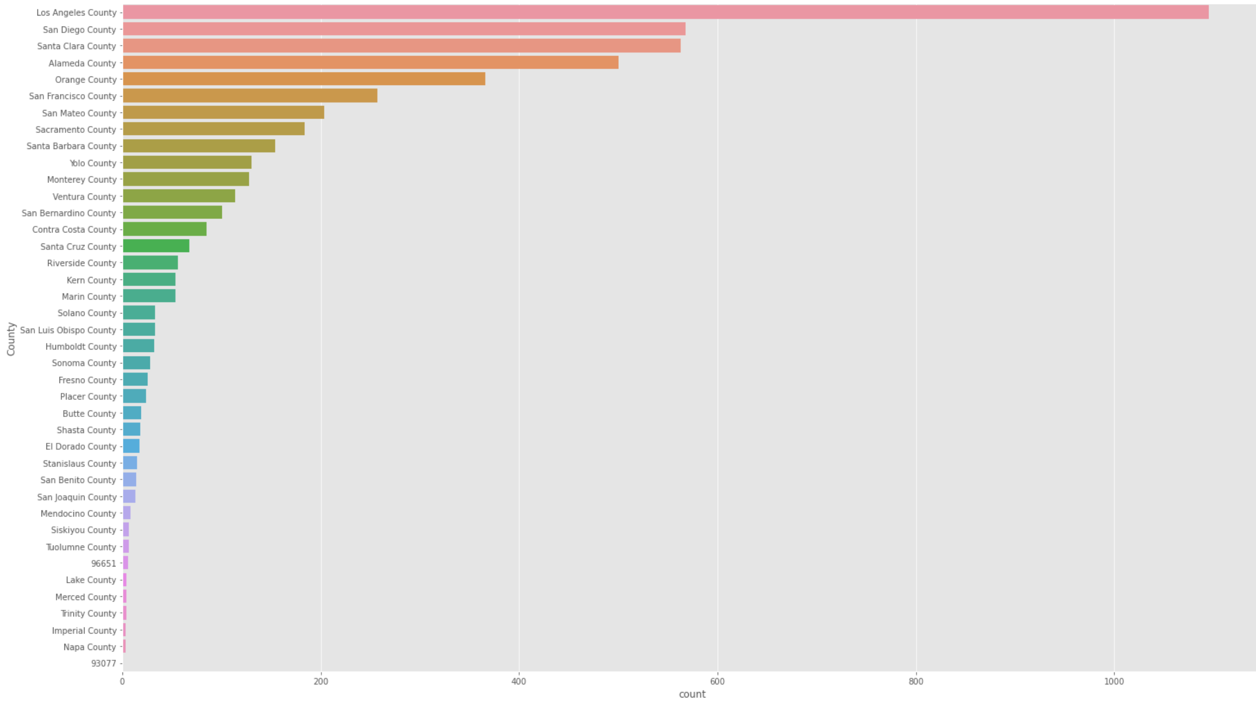


Observation:

- Single customers represents the highest count in our dataset while customers who have family size of 2 reprsents 25.9% of our dataset and customers with family size 3 and 4 represents 20.2% and 24.4% of our dataset respectively.

In [49]:
```
plt.figure(figsize=(25, 15))
sns.countplot(y="County",data=df, order=df["County"].value_counts().index[0:40])
```

Out[49]: <AxesSubplot:xlabel='count', ylabel='County'>



Observation:

- There are so many Counties in ourdatset, we need to bin them into regions to reduce the multivariation and the dimensionality in our model .
- The county with the most customers through our dataset is Las Vegas county.
- The second Largest county in terms of number of customers is San Diego county.

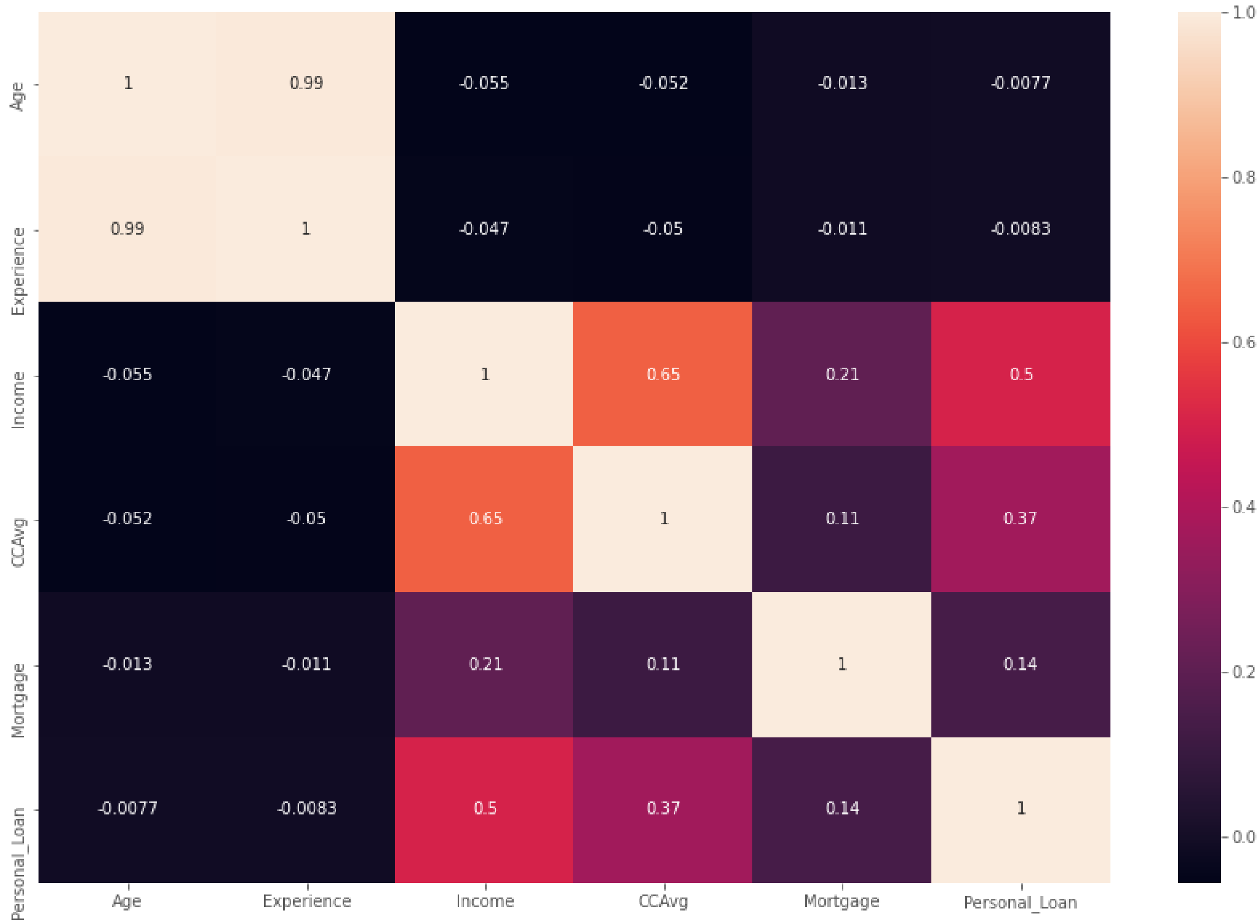
Mutltivariate and Bi Variate Analysis

Let's visualize each inpdenent variable with the dependent variable to better understand the relatioship between them.

In [50]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 5000 non-null   int64
 1   Experience          5000 non-null   int64
 2   Income              5000 non-null   int64
 3   Family              5000 non-null   category
 4   CCAvg               5000 non-null   float64
 5   Education           5000 non-null   category
 6   Mortgage            5000 non-null   int64
 7   Personal_Loan       5000 non-null   int32
 8   Securities_Account  5000 non-null   category
 9   CD_Account          5000 non-null   category
 10  Online              5000 non-null   category
 11  CreditCard          5000 non-null   category
 12  County              5000 non-null   category
dtypes: category(7), float64(1), int32(1), int64(4)
memory usage: 251.4 KB
```

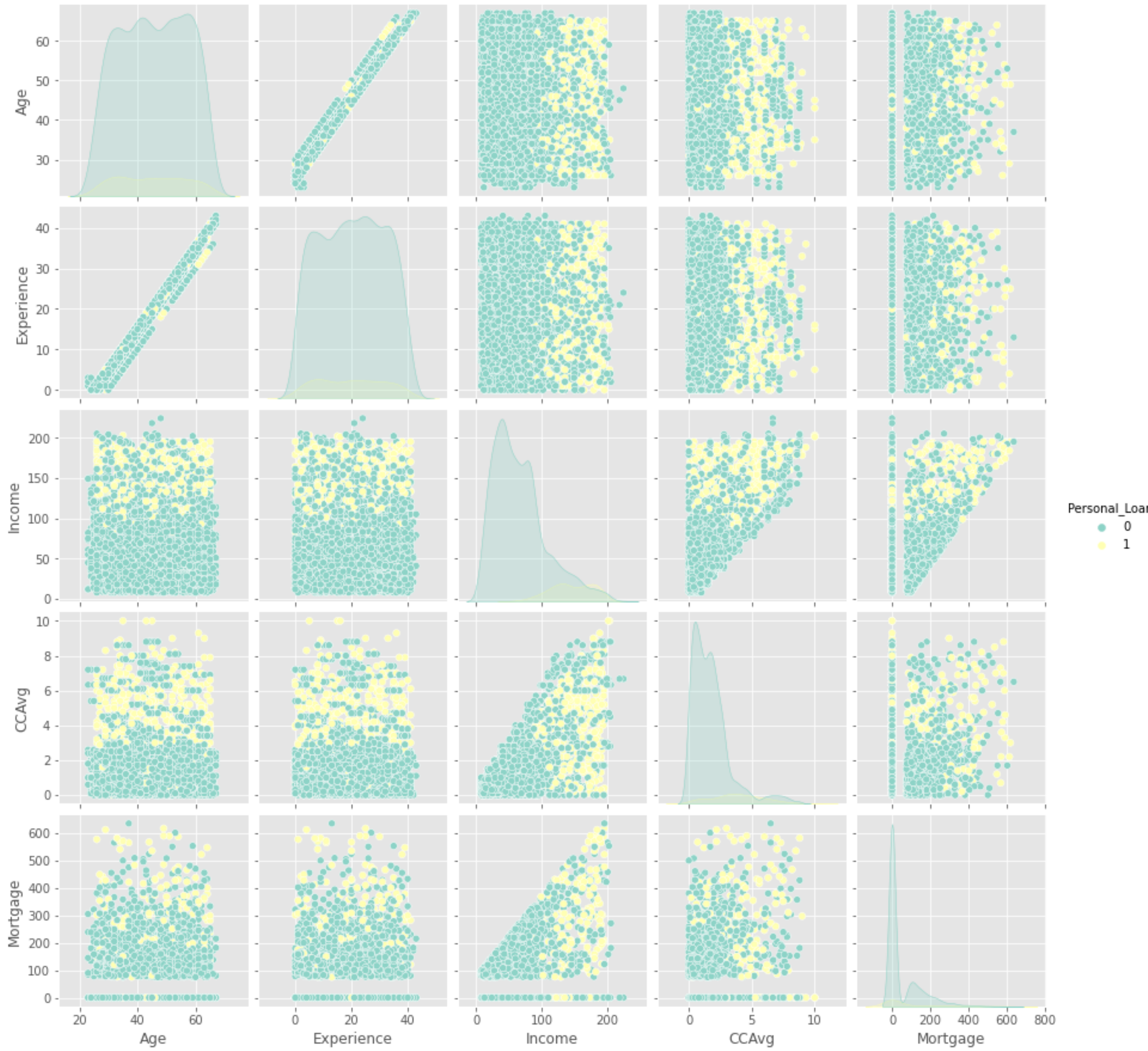
In [51]:
```
sns.set_palette(sns.color_palette("Set2", 8))
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



Observations

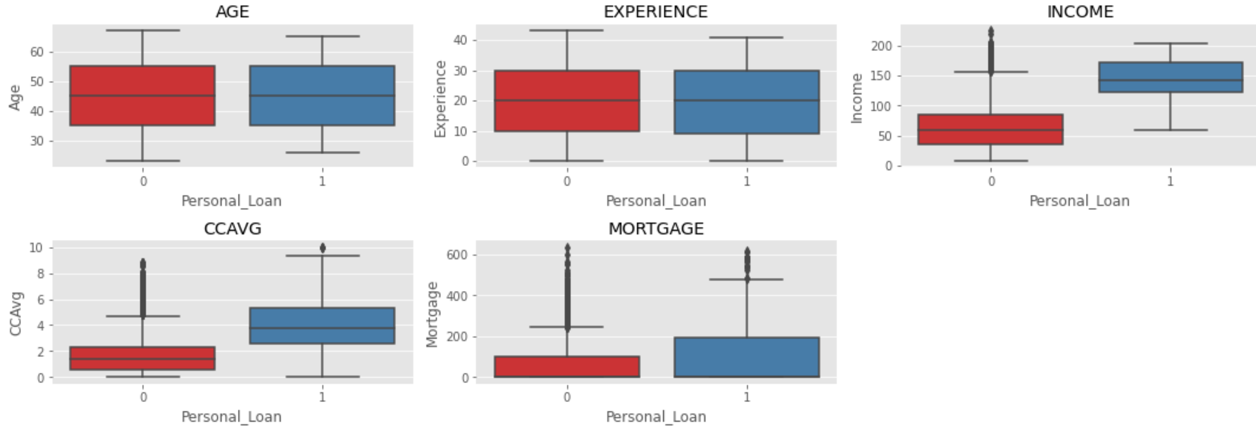
- As expected Age and experience are both Normally distributed and highly correlated and one of them can be dropped.Since we had to handle 0, will drop experience.
- Income and Average spending on credit card are positively corrleated.
- Mortgage has very little correlation with income.

In [57]:
```
sns.set_palette(sns.color_palette("Set3", 8))
sns.pairplot(df, vars=['Age','Experience','Income','CCAvg','Mortgage'],hue="Personal_Lo
plt.show()
```



In [189…

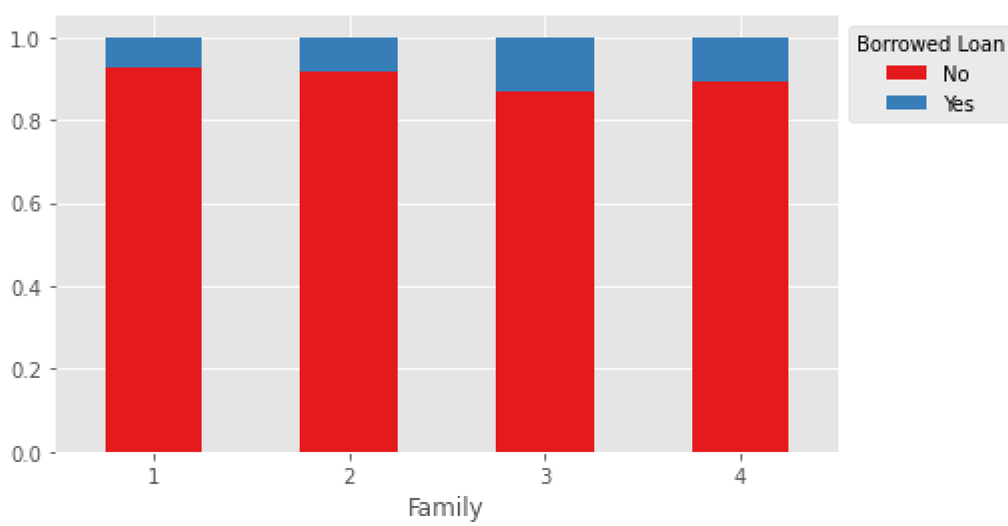
```
numeric_columns = ['Age','Experience','Income','CCAvg','Mortgage']
plt.figure(figsize=(15,25))

sns.set_palette(sns.color_palette("Set1", 8))
for i, variable in enumerate(numeric_columns):
        plt.subplot(10,3,i+1)

        sns.boxplot(x='Personal_Loan',y= df[variable], data=df)
        sns.despine(top=True,right=True,left=True) # to remove side line from graph
        plt.tight_layout()
        plt.title(variable.upper())
```



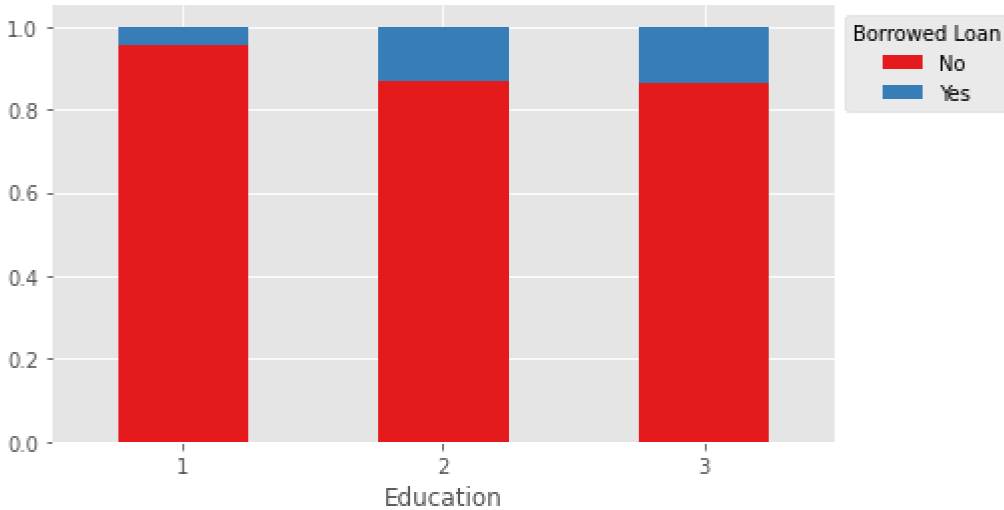
In [63]:
```
## Function to plot stacked bar chart
def stacked_plot(x):
    sns.set_palette(sns.color_palette("Set1", 8))
    tab1 = pd.crosstab(x,df['Personal_Loan'],margins=True)
    print(tab1)
    print('-'*120)
    tab = pd.crosstab(x,df['Personal_Loan'],normalize='index')
    tab.plot(kind='bar',stacked=True,figsize=(7,4))
    plt.xticks(rotation=360)
    labels=["No","Yes"]
    plt.legend(loc='lower left', frameon=False,)
    plt.legend(loc="upper left", labels=labels,title="Borrowed Loan",bbox_to_anchor=(1,
    sns.despine(top=True,right=True,left=True) # to remove side line from graph
    #plt.legend(labels)
    plt.show()
```

In [66]:
```
cat_columns=['Family','Education','Securities_Account','CD_Account','CreditCard','Onlin
for i, variable in enumerate(cat_columns):
        stacked_plot(df[variable])
```

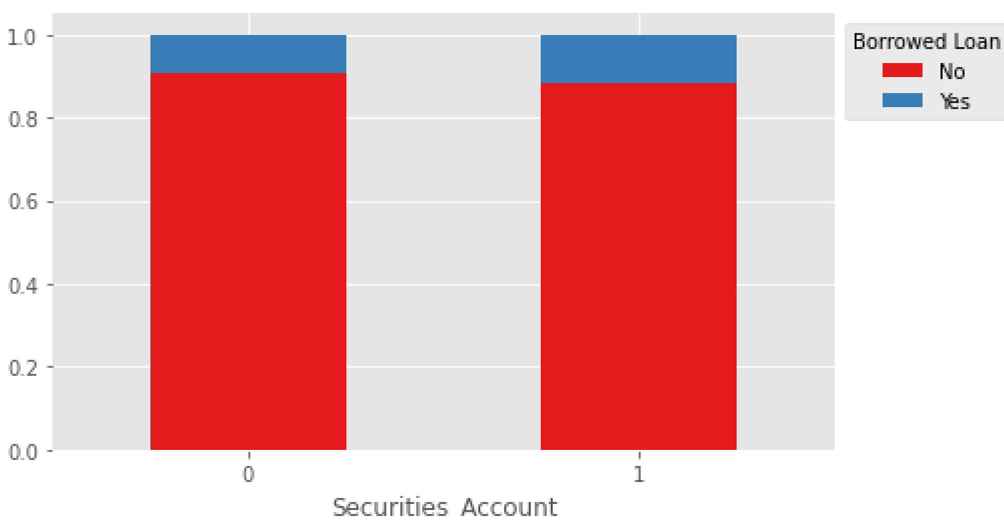
```
Personal_Loan     0    1   All
Family
1              1365  107  1472
2              1190  106  1296
3               877  133  1010
4              1088  134  1222
All            4520  480  5000
------------------------------------------------------------------------------------------
------------------------------
```



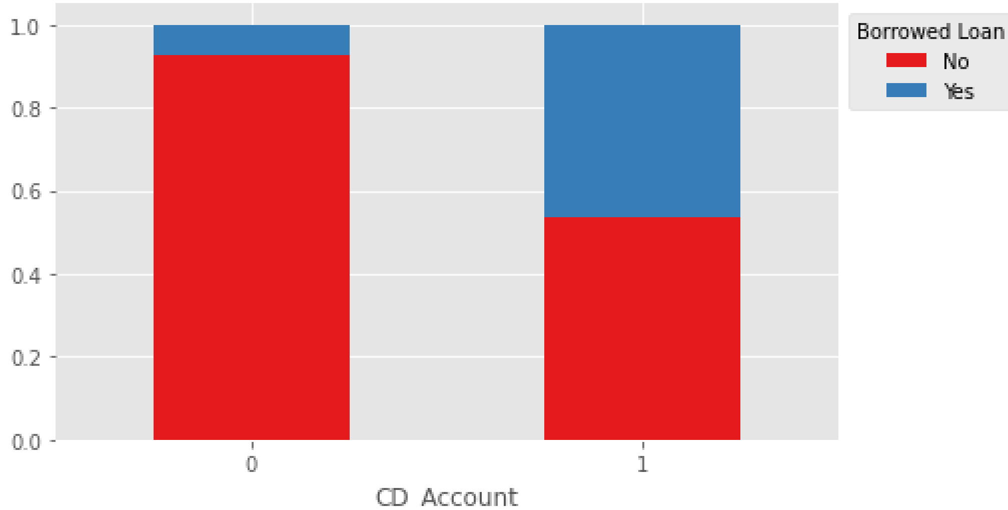
```
Personal_Loan     0    1   All
Education
1              2003   93  2096
2              1221  182  1403
3              1296  205  1501
All            4520  480  5000
------------------------------------------------------------------------------------------
------------------------------
```



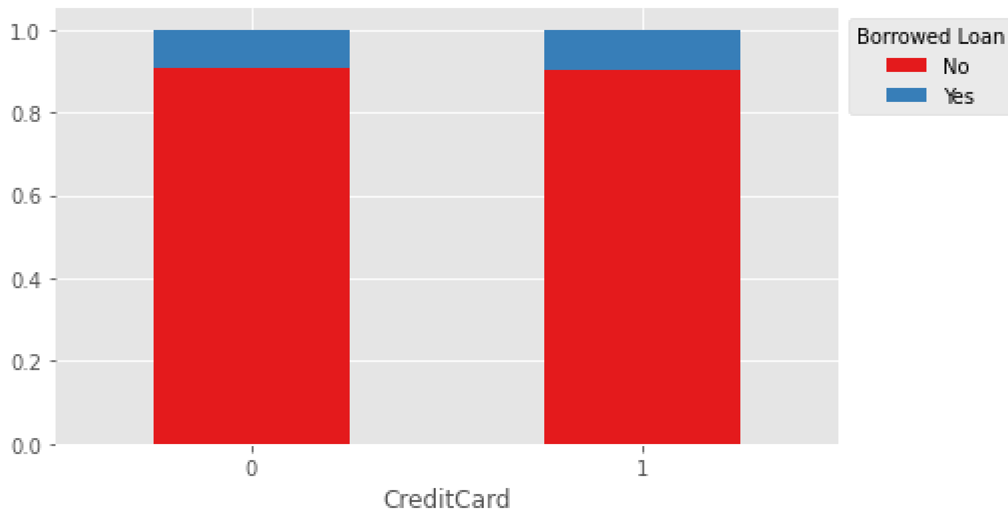

```
Personal_Loan          0    1   All
Securities_Account
0                   4058  420  4478
1                    462   60   522
All                 4520  480  5000
------------------------------------------------------------------------------------------
------------------------------
```



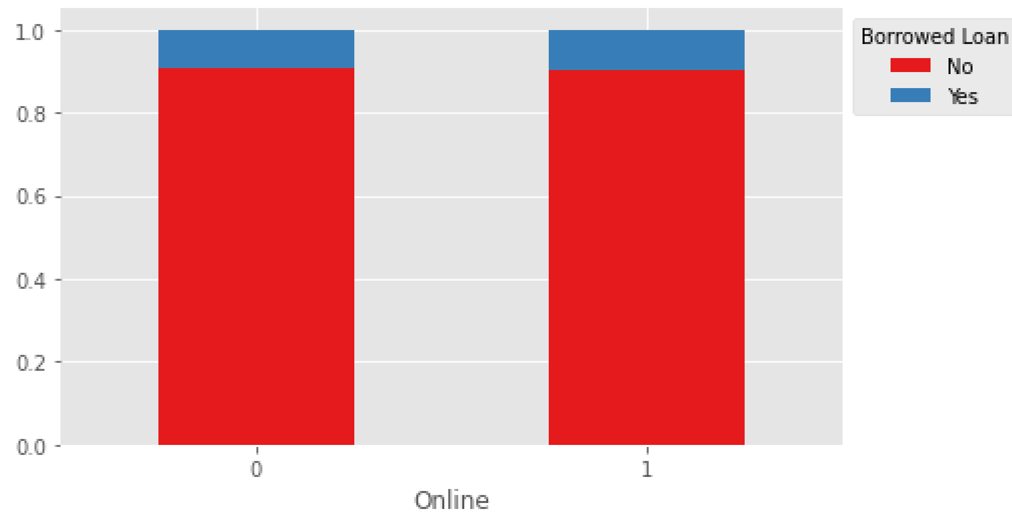

```
Personal_Loan     0    1   All
CD_Account
0              4358  340  4698
1               162  140   302
All            4520  480  5000
------------------------------------------------------------------------------------------
------------------------------
```




```
Personal_Loan     0    1   All
CreditCard
0              3193  337  3530
1              1327  143  1470
All            4520  480  5000
----------------------------------------------------------------------------------------------------------------------------
```




```
Personal_Loan     0    1   All
Online
0              1827  189  2016
1              2693  291  2984
All            4520  480  5000
----------------------------------------------------------------------------------------------------------------------------
```



Let's reduce the dimensionality by assigning each county into it's Main Region

In [67]:
```
counties = {
'Los Angeles County':'Los Angeles Region',
'San Diego County':'Southern',
'Santa Clara County':'Bay Area',
'Alameda County':'Bay Area',
'Orange County':'Southern',
'San Francisco County':'Bay Area',
'San Mateo County':'Bay Area',
'Sacramento County':'Central',
'Santa Barbara County':'Southern',
'Yolo County':'Central',
'Monterey County':'Bay Area',
'Ventura County':'Southern',
'San Bernardino County':'Southern',
'Contra Costa County':'Bay Area',
'Santa Cruz County':'Bay Area',
'Riverside County':'Southern',
'Kern County':'Southern',
'Marin County':'Bay Area',
'San Luis Obispo County':'Southern',
'Solano County':'Bay Area',
'Humboldt County':'Superior',
'Sonoma County':'Bay Area',
'Fresno County':'Central',
'Placer County':'Central',
'Butte County':'Superior',
'Shasta County':'Superior',
'El Dorado County':'Central',
'Stanislaus County':'Central',
'San Benito County':'Bay Area',
'San Joaquin County':'Central',
'Mendocino County':'Superior',
'Tuolumne County':'Central',
'Siskiyou County':'Superior',
'Trinity County':'Superior',
'Merced County':'Central',
'Lake County':'Superior',
'Napa County':'Bay Area',
'Imperial County':'Southern',
93077:'Southern',
```

```
96651:'Bay Area'
}
```

In [68]:
```
df['Regions'] = df['County'].map(counties)
```

In [70]:
```
df['Regions'].unique()
```

Out[70]:
```
array(['Los Angeles Region', 'Bay Area', 'Southern', 'Superior',
       'Central'], dtype=object)
```

Observation:

- Now we can drop the County as we assign all the counties in our datset into 5 main regions

In [71]:
```
df.drop(['County'],axis=1,inplace=True) #droping county
```

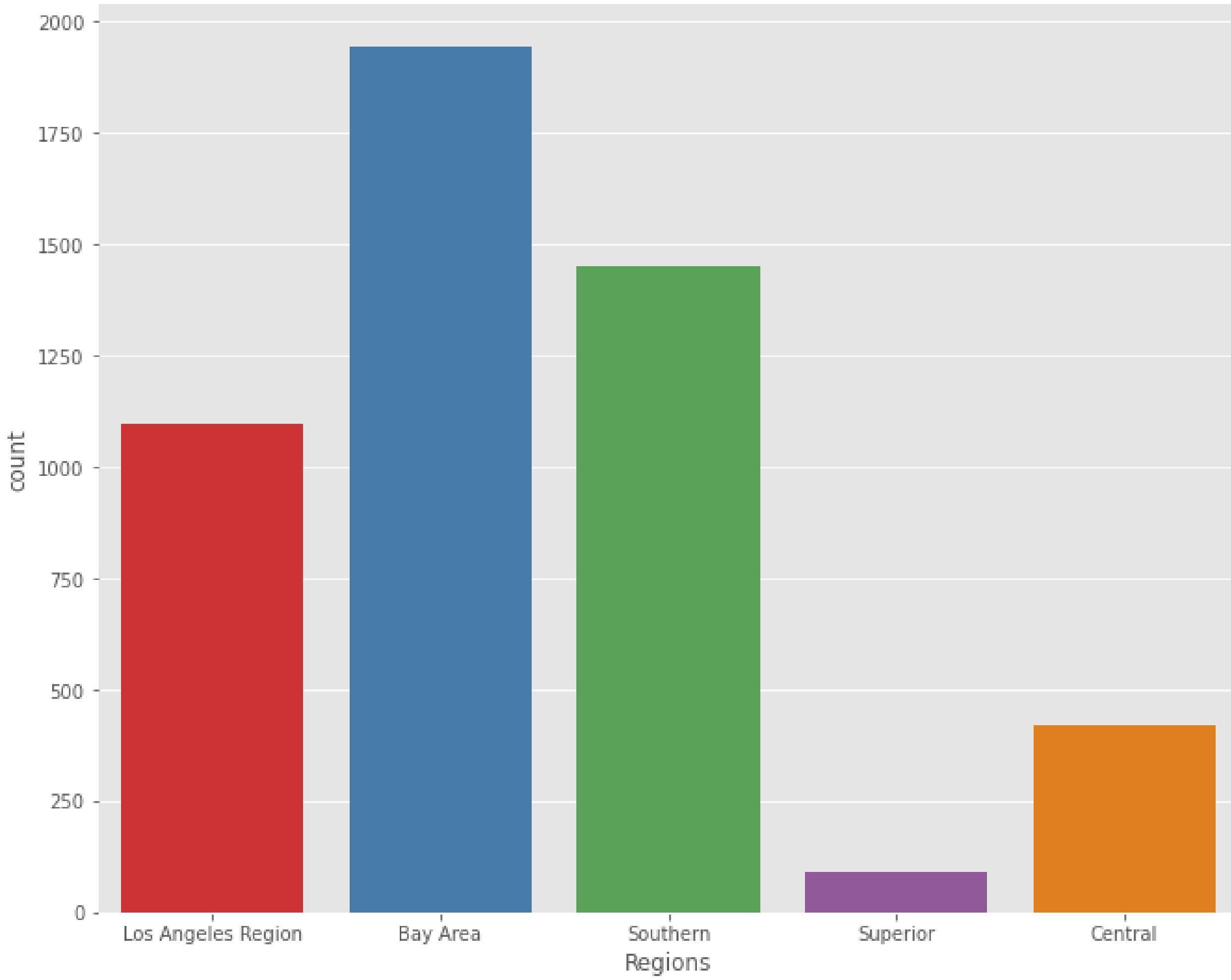
In [72]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 5000 non-null   int64
 1   Experience          5000 non-null   int64
 2   Income              5000 non-null   int64
 3   Family              5000 non-null   category
 4   CCAvg               5000 non-null   float64
 5   Education           5000 non-null   category
 6   Mortgage            5000 non-null   int64
 7   Personal_Loan       5000 non-null   int32
 8   Securities_Account  5000 non-null   category
 9   CD_Account          5000 non-null   category
 10  Online              5000 non-null   category
 11  CreditCard          5000 non-null   category
 12  Regions             5000 non-null   object
dtypes: category(6), float64(1), int32(1), int64(4), object(1)
memory usage: 284.0+ KB
```

In [77]:
```
df['Regions'] = df['Regions'].astype('category')
```

let's visualize our regions to get more insghts

In [74]:
```
plt.figure(figsize=(11,9))
sns.countplot(data=df,x=df['Regions'])
sns.despine(top=True,right=True,left=True) # to remove side line from graph
```



based o our exploratory data analysis we can conclude that :

- Custoemrs with family size of 3 have a higher conversion rate
- The more educated the customers are the higher conversion rate, as we can see that customers with advanced/Professional degree have a higher probability to take a personal loan.
- Customer with Securities Account represents higher conversion rate to take a personal loan.
- Custoemrs who have a CD account shows significant conversion to take a perosnal through our dataset.
- Customer who have an Online Account shows a little bit higher conversion rate than customers who don't have online account.
- Customer who have a Credit card shows a little bit higher conversion rate than those who don't.
- Most of our customers live in bay area, as well as Southern Regions Regions and Los Angles Region comes in the second and third respectively in terms of count among the five main areas of all the customers in the dataset.
- Customers who have a Higher average usage on thier Credit Cards shows a higher conversion rate comparing to others.

- Customers with Higher Income shows higher probability to take a personal.
- Customers who have Mortgage with us tends to take a personal loan more than others.

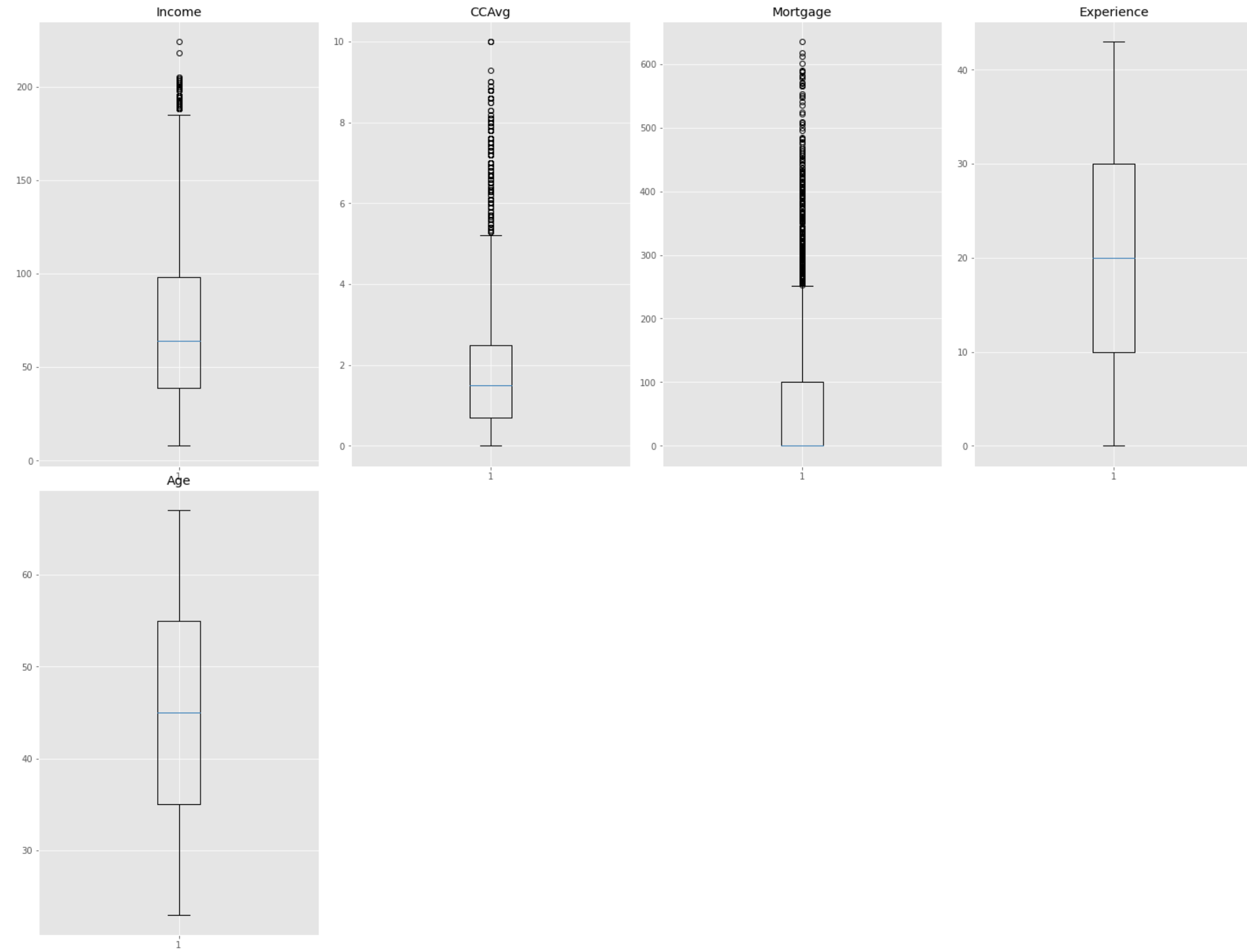
Recommendations on Targeting Personal Loan customers.

- In next Marketing campaign, Exploratorty data analysis recommends to target customers with family size of 3 or more has they have higher probability to take a personal loan.
- we may need to focus on well educated customers as a graudte or professionals if we're seeking in higher conersion rate for Personal loan.
- we need to get closer to customers with CD account and Securities account to understand thier needs and providing them with latest offers on personal loans.
- focusing on customers with High income Profile shows higher conversion rate than other customers, targeting this segment will result in higher conversion rate for the personal loan.
- we can target customers who have a higher average usage and High value of Mortgage as they represents higher probabilty to take a personal loan.
- we need to Focuse on Superior and central areas as they show the least customers through our dataset.

Outlier Detection

In [75]:

```
numeric_columns =['Income','CCAvg','Mortgage','Experience','Age']
# outlier detection using boxplot
plt.figure(figsize=(20,30))

for i, variable in enumerate(numeric_columns):
                     plt.subplot(4,4,i+1)
                     plt.boxplot(df[variable],whis=1.5)
                     plt.tight_layout()
                     plt.title(variable)

plt.show()
```



Observation:

There are small number of outliers in the Income Column and some more Outliers in the Higher part of CCAvg and Mortgage variable.

As a banker,

this could be indication of real balanced dataset as the amount of outliers in each variable is not significant enough to affect our model. so we will keep the outliers without treatment to present real-life case.

In [78]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   Age                 5000 non-null   int64   
 1   Experience          5000 non-null   int64   
 2   Income              5000 non-null   int64   
 3   Family              5000 non-null   category
 4   CCAvg               5000 non-null   float64 
 5   Education           5000 non-null   category
 6   Mortgage            5000 non-null   int64   
 7   Personal_Loan       5000 non-null   int32   
 8   Securities_Account  5000 non-null   category
 9   CD_Account          5000 non-null   category
 10  Online              5000 non-null   category
```

```
 11  CreditCard          5000 non-null   category
 12  Regions             5000 non-null   category
dtypes: category(7), float64(1), int32(1), int64(4)
memory usage: 250.0 KB
```

No need to revert the Binary variables back into numerical values as it's already represented in our dataset with 0s and 1s.

but we will need to Create dummies for Family, Education and Regions columns.

Splitting the dataset

In [94]:
```
# Define Y as dependent variable(personal loan) and X as independent Variables
X =df.drop(['Experience','Personal_Loan'], axis=1)# Drpping Experience column to avoid 
Y = df['Personal_Loan']
```

In [95]:
```
# create dummies for Education, family and Regions columns
Dummies=['Regions','Family','Education']
X=pd.get_dummies(X,columns=Dummies,drop_first=True)
```

In [96]:
```
#Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.30, random_state =
```

Data Scaling using StandardScaler

Let's reduce the scale of the variables to let the model peform better as there is a big scale between Mortgage(value in 1000 Usd and CCAVG as well, and other variables .

In [97]:
```
from sklearn.preprocessing import StandardScaler
# Creating StandardScaler instance
scaler = StandardScaler()

# Fitting Standard Scaller
X_scaler = scaler.fit(X_train)

# Scaling data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)

X_train_scaled_df = pd.DataFrame(X_train_scaled,columns=X_train.columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled,columns=X_test.columns)


X_train_scaled_df.index=np.arange(len(X_train_scaled_df))
X_test_scaled_df.index=np.arange(len(X_test_scaled_df))
y_train.index=np.arange(len(y_train))
y_test.index=np.arange(len(y_test))
```

Building Logistic Regression Model

Objective: Predict whether a liability customer will take personal loan or not.

The main purpose behind our analysis is to identify potential Loan Customers ,so it's important for the bank from busniess prospective is find the customers that are more likely to take a personal loan from the liability customers that are already in the bank.

What Kind of losses that are more significant for the bank to minimize throught our model?

A- Predicting that a customer will take loan while actually he won't.(Loss of Resource)

- False Positive.

B- Predicting that a customer Won't take loan while he will.(Loss of Opportunity)

- False Negative.

In that case it's important from data science prospective to minimize the False negatives.

(the number of customers who will take a loan while the model predict them as not).

So, the right metric to use to check the performance of our logistic model is Recall. the higher the Recall is , the less the number of False negatives.

In [98]:
```
def make_confusion_matrix(y_actual,y_predict,title):
    fig, ax = plt.subplots(1, 1)

    cm = confusion_matrix(y_actual, y_predict, labels=[0,1])
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=["No","Yes"])

    disp.plot(cmap='Reds',ax=ax)
    ax.set_title(title)
    plt.tick_params(axis=u'both', which=u'both',length=0)
    plt.grid(b=None,axis='both',which='both',visible=False)
    plt.show()
```

In [99]:
```
def get_metrics_score(model,X_train_df,X_test_df,y_train_pass,y_test_pass,statsklearn,t
    '''
    Function to calculate different metric scores of the model - Accuracy, Recall, Prec
    model: classifier to predict values of X
    X_train_df, X_test_df: Independent features
    y_train_pass,y_test_pass: Dependent variable
    statsklearn : 0 if calling for Sklearn model else 1
    threshold: thresold for classifiying the observation as 1
    flag: If the flag is set to True then only the print statements showing different w
    roc: If the roc is set to True then only roc score will be displayed. The default v
    '''
    # defining an empty list to store train and test results

    score_list=[]
    if statsklearn==0:
        pred_train = model.predict(X_train_df)
        pred_test = model.predict(X_test_df)
    else:
        pred_train = (model.predict(X_train_df)>threshold)
        pred_test = (model.predict(X_test_df)>threshold)


    pred_train = np.round(pred_train)
    pred_test = np.round(pred_test)

    train_acc = accuracy_score(y_train_pass,pred_train)
    test_acc = accuracy_score(y_test_pass,pred_test)
```

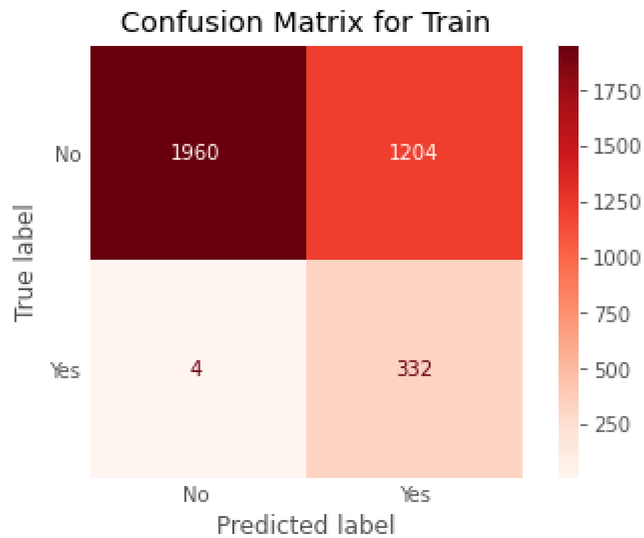
```
    train_recall = recall_score(y_train_pass,pred_train)
    test_recall = recall_score(y_test_pass,pred_test)

    train_precision = precision_score(y_train_pass,pred_train)
    test_precision = precision_score(y_test_pass,pred_test)

    train_f1 = f1_score(y_train_pass,pred_train)
    test_f1 = f1_score(y_test_pass,pred_test)


    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test_

    if flag == True:
        print("\x1b[0;30;47m \033[1mMODEL PERFORMANCE\x1b[0m")
        print("\x1b[0;30;47m \033[1mAccuracy   : Train:\x1b[0m",
              round(accuracy_score(y_train_pass,pred_train),3),
              "\x1b[0;30;47m \033[1mTest:\x1b[0m ",
              round(accuracy_score(y_test_pass,pred_test),3))
        print("\x1b[0;30;47m \033[1mRecall     : Train:\x1b[0m"
              ,round(recall_score(y_train_pass,pred_train),3),
              "\x1b[0;30;47m \033[1mTest:\x1b[0m" ,
              round(recall_score(y_test_pass,pred_test),3))

        print("\x1b[0;30;47m \033[1mPrecision  : Train:\x1b[0m",
              round(precision_score(y_train_pass,pred_train),3),
              "\x1b[0;30;47m \033[1mTest:\x1b[0m ",
              round(precision_score(y_test_pass,pred_test),3))
        print("\x1b[0;30;47m \033[1mF1         : Train:\x1b[0m",
              round(f1_score(y_train_pass,pred_train),3),
              "\x1b[0;30;47m \033[1mTest:\x1b[0m",
              round(f1_score(y_test_pass,pred_test),3))
        make_confusion_matrix(y_train_pass,pred_train,"Confusion Matrix for Train")
        make_confusion_matrix(y_test_pass,pred_test,"Confusion Matrix for Test")

    if roc == True:

        print("\x1b[0;30;47m \033[1mROC-AUC Score  :Train:\x1b[0m: ",
              round(roc_auc_score(y_train_pass,pred_train),3),
              "\x1b[0;30;47m \033[1mTest:\x1b[0m: ",
              round(roc_auc_score(y_test_pass,pred_test),3))

    return score_list # returning the list with train and test scores
```

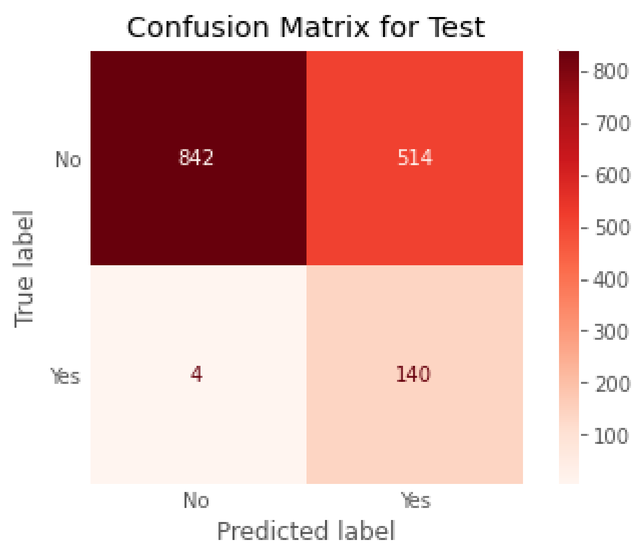
Logistic Regression with Sklearn

In [101…

```
LogisticRegressionModel = LogisticRegression(solver='newton-cg',random_state=1,fit_inte
model  =LogisticRegressionModel.fit(X_train_scaled_df,y_train)

statmodel=0

# Let's check model performances for this model
scores_Sklearn = get_metrics_score(model,X_train_scaled_df,X_test_scaled_df,y_train,y_t
add_score_model(scores_Sklearn)
```

```
MODEL PERFORMANCE
Accuracy   : Train: 0.655  Test:  0.655
Recall     : Train: 0.988  Test: 0.972
Precision  : Train: 0.216  Test:  0.214
F1         : Train: 0.355  Test: 0.351
```



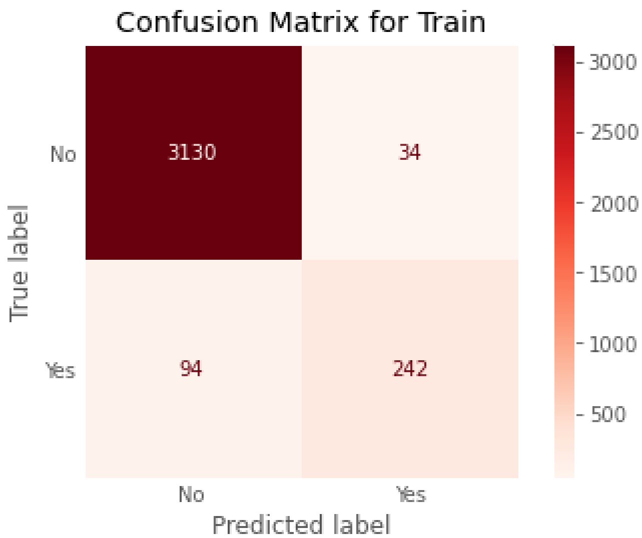


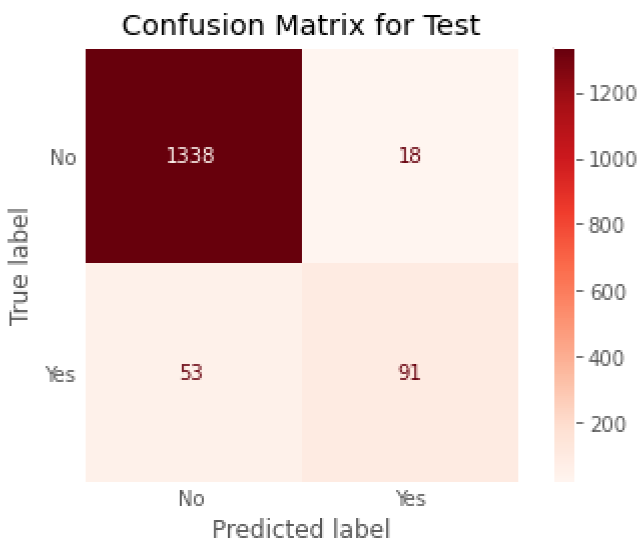
Logistic Regression with Statsmodel

In [102…

```
# adding constant to training and test set
X_train_stat = sm.add_constant(X_train_scaled_df)
X_test_stat = sm.add_constant(X_test_scaled_df)
statmodel=1  #0 for sklearn and 1 for statmodel
logit = sm.Logit( y_train, X_train_stat.astype(float) )
lg = logit.fit(warn_convergence=False)

# Let's check model performances for this model
scores_statmodel = get_metrics_score(lg,X_train_stat,X_test_stat,y_train,y_test,statmod
lg.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.104037
         Iterations 10
 MODEL PERFORMANCE
 Accuracy    : Train: 0.963  Test:  0.953
 Recall      : Train: 0.72  Test: 0.632
 Precision   : Train: 0.877  Test:  0.835
 F1          : Train: 0.791  Test: 0.719
```





Out[102…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3500
Model:	Logit	**Df Residuals:**	3482
Method:	MLE	**Df Model:**	17
Date:	Sat, 17 Jul 2021	**Pseudo R-squ.:**	0.6710
Time:	23:07:47	**Log-Likelihood:**	-364.13
converged:	True	**LL-Null:**	-1106.7
Covariance Type:	nonrobust	**LLR p-value:**	7.710e-306

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-5.5428	0.253	-21.916	0.000	-6.038	-5.047
Age	0.2050	0.101	2.033	0.042	0.007	0.403
Income	3.2256	0.190	16.997	0.000	2.854	3.598
CCAvg	0.2647	0.103	2.568	0.010	0.063	0.467
Mortgage	0.1221	0.077	1.582	0.114	-0.029	0.273

Securities_Account	-0.2942	0.119	-2.472	0.013	-0.528	-0.061
CD_Account	0.8688	0.106	8.175	0.000	0.661	1.077
Online	-0.3209	0.104	-3.092	0.002	-0.524	-0.117
CreditCard	-0.4621	0.124	-3.727	0.000	-0.705	-0.219
Regions_Central	-0.1896	0.118	-1.600	0.110	-0.422	0.043
Regions_Los Angeles Region	-0.0610	0.107	-0.569	0.570	-0.271	0.149
Regions_Southern	0.0338	0.108	0.313	0.754	-0.177	0.245
Regions_Superior	-0.2107	0.175	-1.204	0.229	-0.554	0.132
Family_2	-0.1126	0.128	-0.881	0.378	-0.363	0.138
Family_3	0.7667	0.126	6.073	0.000	0.519	1.014
Family_4	0.6781	0.126	5.392	0.000	0.432	0.925
Education_2	2.0085	0.161	12.509	0.000	1.694	2.323
Education_3	2.0976	0.163	12.896	0.000	1.779	2.416

Possibly complete quasi-separation: A fraction 0.17 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

Testing the Assumptions:

Multicollinearity we needremove multicollinearity from the dataset to get better coefficients and p-values.

We will useVariation Inflation Factor method to remove multicollinearity.

General Rule of thumb: If VIF is 1 then there is no correlation among the predictor and the remaining predictor variables. Whereas if VIF exceeds 5, we say it shows signs of high multi-collinearity. But the purpose of the analysis should dictate which threshold to use.

In [103…

```
# changing datatype of colums to numeric for checking vif
X_train_num = X_train_stat.astype(float).copy()
```

In [104…

```
vif_series1 = pd.Series([variance_inflation_factor(X_train_num.values,i) for i in range
print('Series before feature selection: \n\n{}\n'.format(vif_series1))
```

```
Series before feature selection:

const                        1.00000
Age                          1.01539
Income                       1.82477
CCAvg                        1.68656
Mortgage                     1.05058
Securities_Account           1.14067
CD_Account                   1.33312
Online                       1.04741
CreditCard                   1.10707
```

```
Regions_Central             1.11898
Regions_Los Angeles Region  1.21718
Regions_Southern            1.23693
Regions_Superior            1.03382
Family_2                    1.39864
Family_3                    1.37655
Family_4                    1.42676
Education_2                 1.25622
Education_3                 1.24101
dtype: float64
```

In [107…

```
lg.summary()
```

Out[107…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3500
Model:	Logit	**Df Residuals:**	3482
Method:	MLE	**Df Model:**	17
Date:	Sat, 17 Jul 2021	**Pseudo R-squ.:**	0.6710
Time:	23:11:35	**Log-Likelihood:**	-364.13
converged:	True	**LL-Null:**	-1106.7
Covariance Type:	nonrobust	**LLR p-value:**	7.710e-306

	coef	std err	z	P>|z|	[0.025	0.975]
const	-5.5428	0.253	-21.916	0.000	-6.038	-5.047
Age	0.2050	0.101	2.033	0.042	0.007	0.403
Income	3.2256	0.190	16.997	0.000	2.854	3.598
CCAvg	0.2647	0.103	2.568	0.010	0.063	0.467
Mortgage	0.1221	0.077	1.582	0.114	-0.029	0.273
Securities_Account	-0.2942	0.119	-2.472	0.013	-0.528	-0.061
CD_Account	0.8688	0.106	8.175	0.000	0.661	1.077
Online	-0.3209	0.104	-3.092	0.002	-0.524	-0.117
CreditCard	-0.4621	0.124	-3.727	0.000	-0.705	-0.219
Regions_Central	-0.1896	0.118	-1.600	0.110	-0.422	0.043
Regions_Los Angeles Region	-0.0610	0.107	-0.569	0.570	-0.271	0.149
Regions_Southern	0.0338	0.108	0.313	0.754	-0.177	0.245
Regions_Superior	-0.2107	0.175	-1.204	0.229	-0.554	0.132
Family_2	-0.1126	0.128	-0.881	0.378	-0.363	0.138
Family_3	0.7667	0.126	6.073	0.000	0.519	1.014
Family_4	0.6781	0.126	5.392	0.000	0.432	0.925
Education_2	2.0085	0.161	12.509	0.000	1.694	2.323
Education_3	2.0976	0.163	12.896	0.000	1.779	2.416

Possibly complete quasi-separation: A fraction 0.17 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

Observation: We can notice that there are some variables have a very high p-values which mean it doesn't add any value to our model , let's drop them all to remove the multicoliinearity except for CCAvg as it's an interested variable as per the EDA .

In [108…

```
#Dropping Regions Variables
X_train1 = X_train_stat.drop(['Regions_Central', 'Regions_Los Angeles Region', 'Regions_
X_test1= X_test_stat.drop(['Regions_Central', 'Regions_Los Angeles Region', 'Regions_So
logit1 = sm.Logit(y_train, X_train1.astype(float))
lg1 = logit1.fit(warn_convergence =False)

lg1.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.104794
         Iterations 10
```

Out[108…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3500
Model:	Logit	**Df Residuals:**	3486
Method:	MLE	**Df Model:**	13
Date:	Sat, 17 Jul 2021	**Pseudo R-squ.:**	0.6686
Time:	23:15:08	**Log-Likelihood:**	-366.78
converged:	True	**LL-Null:**	-1106.7
Covariance Type:	nonrobust	**LLR p-value:**	9.423e-309

	coef	std err	z	P>|z|	[0.025	0.975]
const	-5.5007	0.250	-22.045	0.000	-5.990	-5.012
Age	0.2048	0.100	2.040	0.041	0.008	0.402
Income	3.2040	0.188	17.083	0.000	2.836	3.572
CCAvg	0.2703	0.102	2.643	0.008	0.070	0.471
Mortgage	0.1186	0.077	1.549	0.121	-0.031	0.269
Securities_Account	-0.2856	0.119	-2.410	0.016	-0.518	-0.053
CD_Account	0.8737	0.106	8.276	0.000	0.667	1.081
Online	-0.3372	0.103	-3.262	0.001	-0.540	-0.135
CreditCard	-0.4678	0.124	-3.780	0.000	-0.710	-0.225
Family_2	-0.1103	0.127	-0.868	0.385	-0.360	0.139
Family_3	0.7688	0.125	6.156	0.000	0.524	1.014
Family_4	0.6695	0.125	5.348	0.000	0.424	0.915

Education_2	1.9986	0.159	12.537	0.000	1.686	2.311
Education_3	2.0899	0.162	12.900	0.000	1.772	2.407

Possibly complete quasi-separation: A fraction 0.16 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

In [109…

```
X_train2 = X_train1.drop(['Age'], axis = 1)
X_test2= X_test1.drop(['Age'], axis = 1)
logit2 = sm.Logit(y_train, X_train2.astype(float))
lg2 = logit2.fit()
lg2.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.105393
         Iterations 10
```

Out[109…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3500
Model:	Logit	**Df Residuals:**	3487
Method:	MLE	**Df Model:**	12
Date:	Sat, 17 Jul 2021	**Pseudo R-squ.:**	0.6667
Time:	23:22:32	**Log-Likelihood:**	-368.87
converged:	True	**LL-Null:**	-1106.7
Covariance Type:	nonrobust	**LLR p-value:**	6.661e-309

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-5.4640	0.247	-22.146	0.000	-5.948	-4.980
Income	3.1811	0.186	17.107	0.000	2.817	3.546
CCAvg	0.2483	0.102	2.429	0.015	0.048	0.449
Mortgage	0.1137	0.076	1.488	0.137	-0.036	0.264
Securities_Account	-0.2963	0.119	-2.497	0.013	-0.529	-0.064
CD_Account	0.8834	0.105	8.384	0.000	0.677	1.090
Online	-0.3381	0.103	-3.274	0.001	-0.540	-0.136
CreditCard	-0.4610	0.123	-3.734	0.000	-0.703	-0.219
Family_2	-0.1174	0.126	-0.928	0.353	-0.365	0.130
Family_3	0.7591	0.125	6.079	0.000	0.514	1.004
Family_4	0.6559	0.125	5.246	0.000	0.411	0.901
Education_2	1.9831	0.159	12.506	0.000	1.672	2.294
Education_3	2.0693	0.161	12.884	0.000	1.755	2.384

Possibly complete quasi-separation: A fraction 0.16 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

In [110…

```
X_train3 = X_train2.drop(['Mortgage'], axis = 1)
X_test3= X_test2.drop(['Mortgage'], axis = 1)
logit3 = sm.Logit(y_train, X_train3.astype(float))
lg3 = logit3.fit()
lg3.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.105708
         Iterations 10
```

Out[110…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3500
Model:	Logit	**Df Residuals:**	3488
Method:	MLE	**Df Model:**	11
Date:	Sat, 17 Jul 2021	**Pseudo R-squ.:**	0.6657
Time:	23:24:55	**Log-Likelihood:**	-369.98
converged:	True	**LL-Null:**	-1106.7
Covariance Type:	nonrobust	**LLR p-value:**	1.682e-309

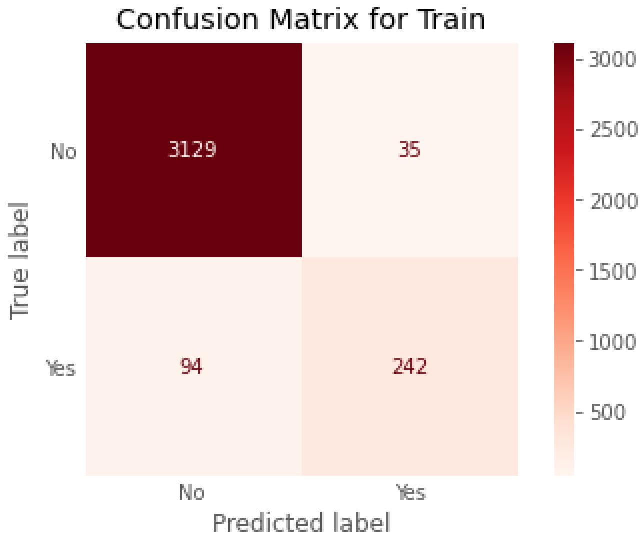
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-5.4544	0.246	-22.159	0.000	-5.937	-4.972
Income	3.2038	0.186	17.257	0.000	2.840	3.568
CCAvg	0.2328	0.101	2.296	0.022	0.034	0.432
Securities_Account	-0.2993	0.118	-2.527	0.011	-0.531	-0.067
CD_Account	0.8944	0.105	8.501	0.000	0.688	1.101
Online	-0.3344	0.103	-3.248	0.001	-0.536	-0.133
CreditCard	-0.4635	0.123	-3.767	0.000	-0.705	-0.222
Family_2	-0.1057	0.126	-0.839	0.401	-0.353	0.141
Family_3	0.7623	0.125	6.088	0.000	0.517	1.008
Family_4	0.6637	0.125	5.307	0.000	0.419	0.909
Education_2	1.9731	0.158	12.489	0.000	1.663	2.283
Education_3	2.0607	0.160	12.878	0.000	1.747	2.374

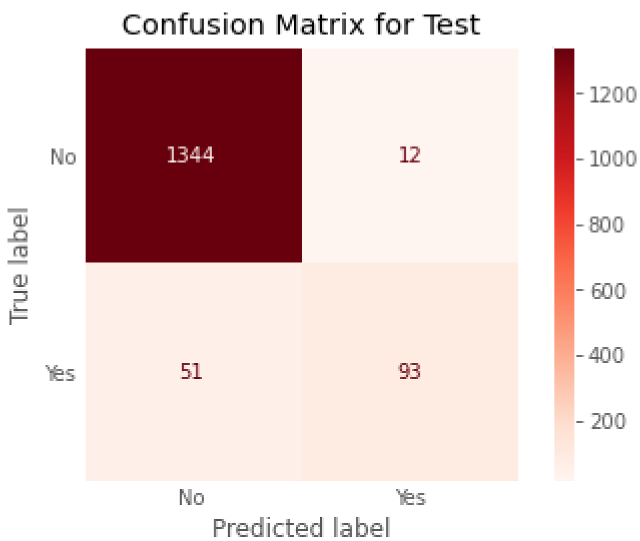
Possibly complete quasi-separation: A fraction 0.15 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

In [111…

```
# Let's check model performances for this model and add the scores to our comparisons

scores_statmodel = get_metrics_score(lg3,X_train3,X_test3,y_train,y_test,statmodel)
add_score_model(scores_statmodel)
```

```
MODEL PERFORMANCE
Accuracy   : Train: 0.963  Test:  0.958
Recall     : Train: 0.72  Test: 0.646
Precision  : Train: 0.874  Test:  0.886
F1         : Train: 0.79  Test: 0.747
```





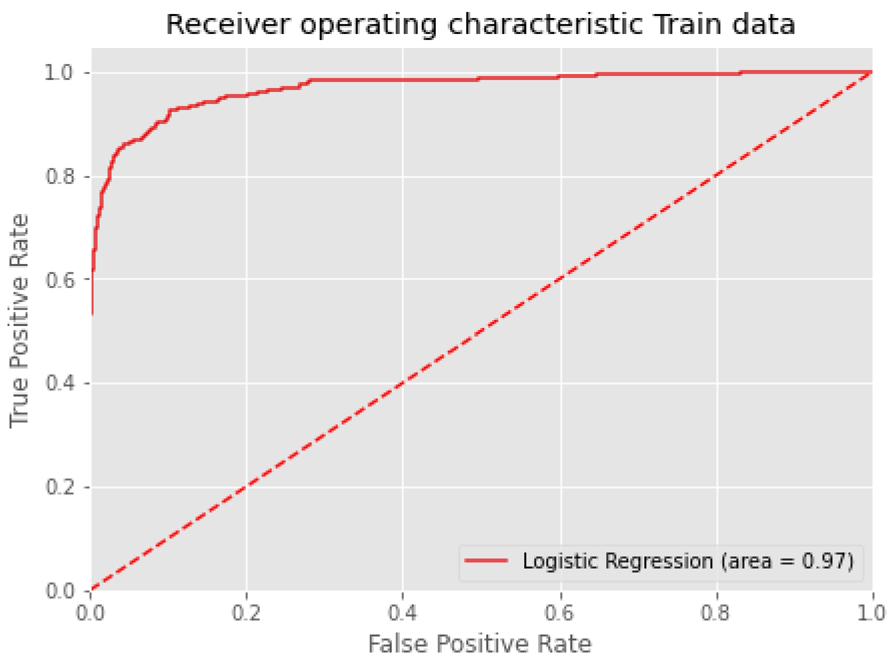
ROC-AUC curve

Roc -Auc curve on Train dataset.

In [112…

```
logit_roc_auc_train = roc_auc_score(y_train, lg3.predict(X_train3))
fpr, tpr, thresholds = roc_curve(y_train, lg3.predict(X_train3))
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
```

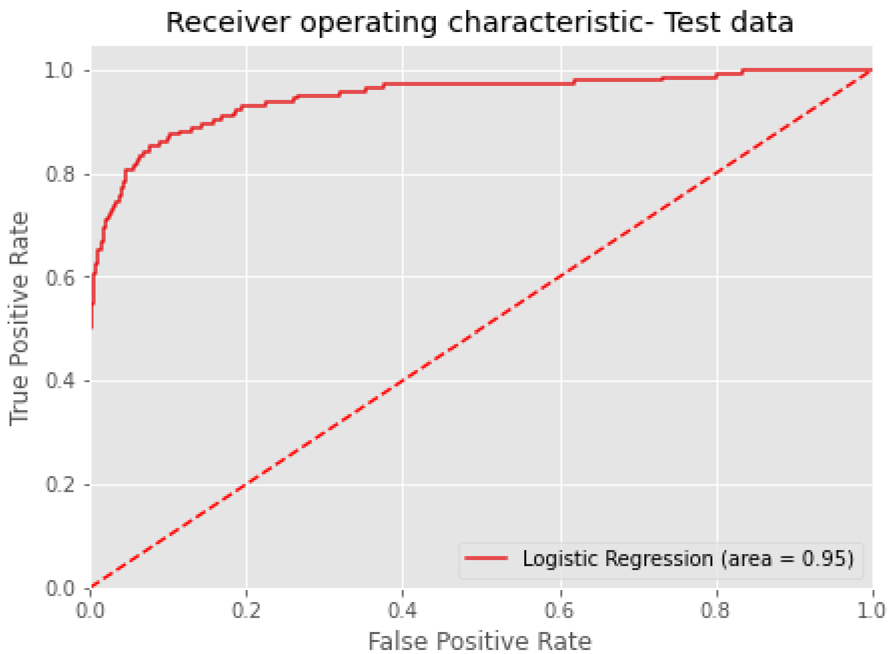
```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic Train data')
plt.legend(loc="lower right")
plt.show()
```



Roc-AUC curve on Test dataset

In [114...

```
logit_roc_auc_test = roc_auc_score(y_test, lg3.predict(X_test3))
fpr, tpr, thresholds = roc_curve(y_test, lg3.predict(X_test3))
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc_test)
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- Test data')
plt.legend(loc="lower right")
plt.show()
```



Observation: ROC-AUC value on test dataset is 0.95

Coefficient interpretations

- Coefficient of Age, Income, Education, Family,CCavg,CD account, are positive , so any increase in these will lead to increase in chances of a person borrowing loan.
- Coefficient of Securities account,online ,Credit card are negative, so any increase in these will lead to decrease in chances of a person borrowing a loan.

Let's calculate the odds ratio

In [116…

```
#Calculate Odds Ratio, probability
##create a data frame to collate Odds ratio, probability and p-value of the coef
lgcoef = pd.DataFrame(lg3.params, columns=['coef'])
lgcoef.loc[:, "Odds Ratio"] = np.exp(lgcoef.coef)
lgcoef['Probability'] = lgcoef['Odds Ratio']/(1+lgcoef['Odds Ratio'])
lgcoef['Percentage Change of Odds']=(np.exp(lg3.params)-1)*100
lgcoef['pval']=lg3.pvalues
pd.options.display.float_format = '{:.2f}'.format
lgcoef = lgcoef.sort_values(by="Odds Ratio", ascending=False)
lgcoef
```

Out[116…

	coef	Odds Ratio	Probability	Percentage Change of Odds	pval
Income	3.20	24.63	0.96	2362.55	0.00
Education_3	2.06	7.85	0.89	685.17	0.00
Education_2	1.97	7.19	0.88	619.28	0.00
CD_Account	0.89	2.45	0.71	144.59	0.00
Family_3	0.76	2.14	0.68	114.33	0.00
Family_4	0.66	1.94	0.66	94.19	0.00
CCAvg	0.23	1.26	0.56	26.21	0.02

	coef	Odds Ratio	Probability	Percentage Change of Odds	pval
Family_2	-0.11	0.90	0.47	-10.03	0.40
Securities_Account	-0.30	0.74	0.43	-25.87	0.01
Online	-0.33	0.72	0.42	-28.42	0.00
CreditCard	-0.46	0.63	0.39	-37.10	0.00
const	-5.45	0.00	0.00	-99.57	0.00

Objective 2 :Which variables are most significant.?

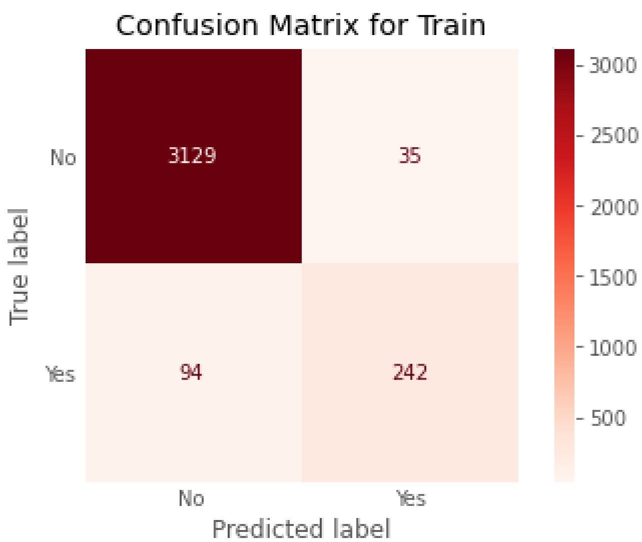
we can see that the Most Significant independent variables that affect Personal loan (dependent variable) are:

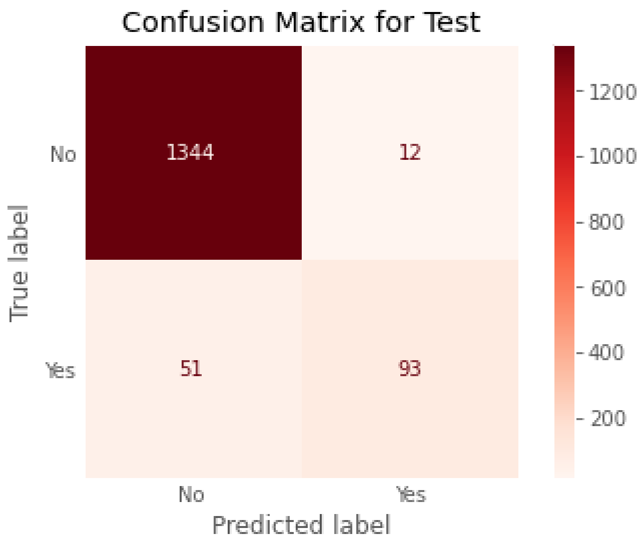
Income, Family, Educatin, CD_Account and CCAvg.

In [117…

```
# Let's check model performances for this model
scores_LR = get_metrics_score(lg3,X_train3,X_test3,y_train,y_test,statmodel)
```

```
MODEL PERFORMANCE
Accuracy   : Train: 0.963  Test:  0.958
Recall     : Train: 0.72  Test: 0.646
Precision  : Train: 0.874  Test:  0.886
F1         : Train: 0.79  Test: 0.747
```





Insights:

True Positives:

Reality: A customer wanted to take personal Loan. Model Prediction: The customer will take personal loan. Outcome: The model is good.

True Negatives:

Reality: A customer didn't wanted to take personal loan. Model Prediction: The customer will not take personal loan. Outcome: The business is unaffected .

False Positives :

Reality: A customer didn't want to take personal loan. Model Prediction: The customer will take personal loan. Outcome: The team which is targeting the potential customers would waste their resources on the customers who will not be buying a personal loan.

False Negatives:

Reality: A customer wanted to take personal Loan. Model Prediction: The customer will not take personal loan. Outcome: The potential customer is missed by the salesteam. This is loss of oppurtunity. The purpose of campaign was to target such customers. If team knew about this customers, they could have offered some good APR /interest rates.

Right Metric to use: Here not able to identify a potential customer is the biggest loss we can face. Hence, Recall is the right metric to check the performance of the model .We have recall as 72 on train and 64 on test. False negative are 64 and 51 on train and test. We can further improve this score using Optimal threshold for ROC AUC curve and precision recall curve

Optimal Threshold Using AUC-ROC curve

In [119…
```
# Optimal threshold as per AUC-ROC curve
# The optimal cut off would be where tpr is high and fpr is low
#fpr, tpr, thresholds = metrics.roc_curve(y_test, lg2.predict(X_test2))
```

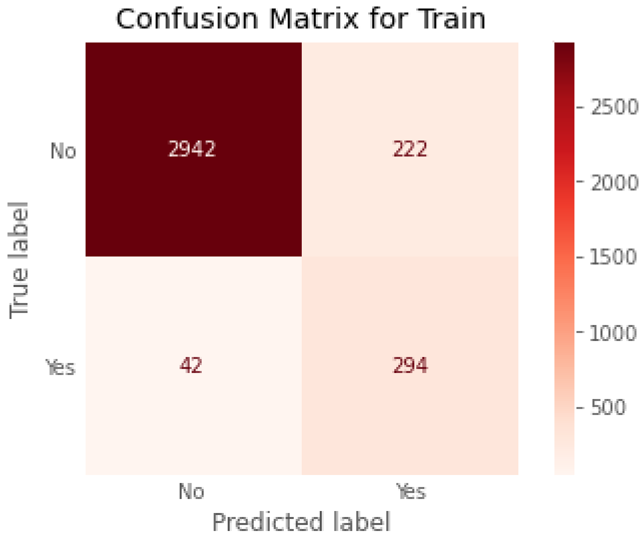
```
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold_auc_roc = thresholds[optimal_idx]
print(optimal_threshold_auc_roc)
```

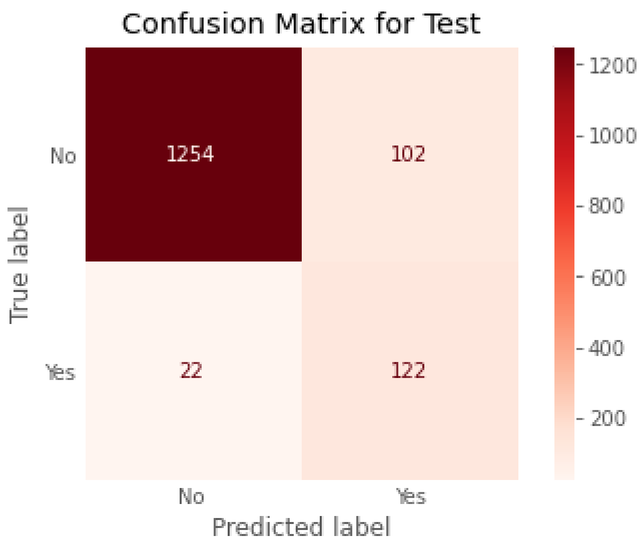
```
0.12366057291718258
```

In [121…

```
scores_statmodel = get_metrics_score(lg3,X_train3,X_test3,y_train,y_test,statmodel,thre
add_score_model(scores_statmodel)
```

```
MODEL PERFORMANCE
Accuracy   : Train: 0.925  Test:  0.917
Recall     : Train: 0.875  Test: 0.847
Precision  : Train: 0.57  Test:  0.545
F1         : Train: 0.69  Test: 0.663
```






```
ROC-AUC Score  :Train::  0.902  Test::  0.886
```

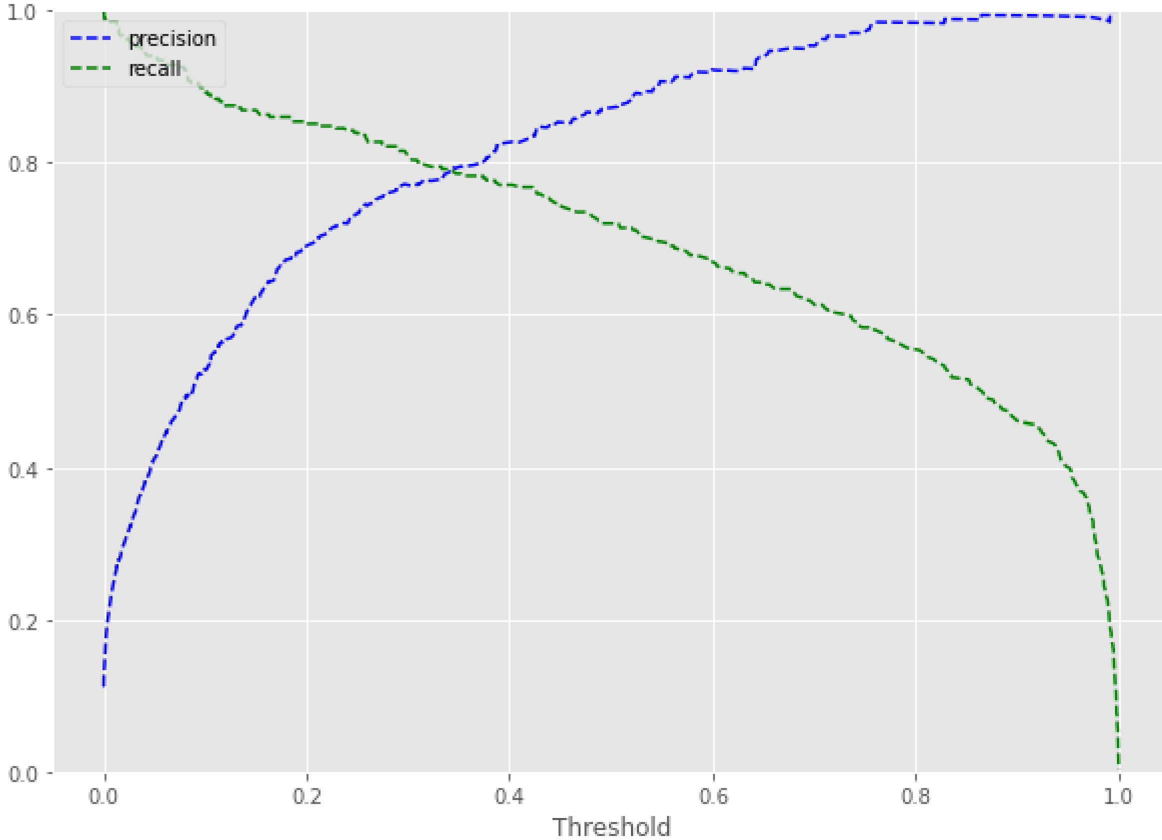
With 0.092 Threshold the Recall score has improved from 64.6% to 84.7% on test data with 91% accuracy. Also False negative values has decreased to 22 from 51 for test dataset.

ROC-AUC score is 0.89 on test se which is good.

In [147…

```
y_scores=lg3.predict(X_train3)
prec, rec, tre = precision_recall_curve(y_train, y_scores,)

def plot_prec_recall_vs_tresh(precisions, recalls, thresholds):
```

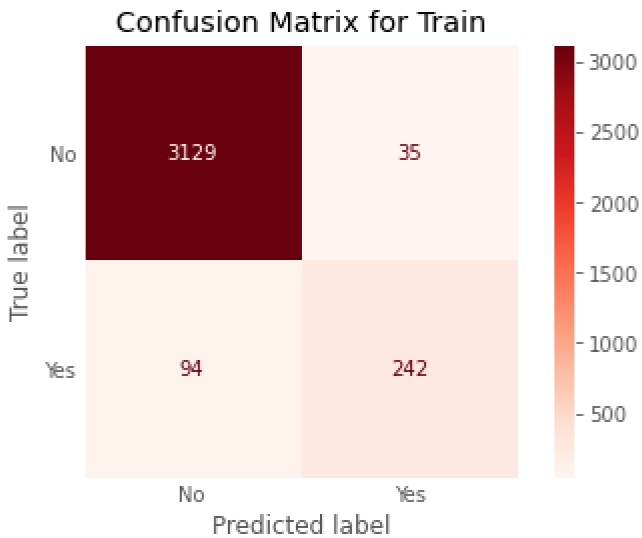
```
    plt.plot(thresholds, precisions[:-1], 'b--', label='precision')
    plt.plot(thresholds, recalls[:-1], 'g--', label = 'recall')
    plt.xlabel('Threshold')
    plt.legend(loc='upper left')
    plt.ylim([0,1])
plt.figure(figsize=(10,7))
plot_prec_recall_vs_tresh(prec, rec, tre)
plt.show()
```

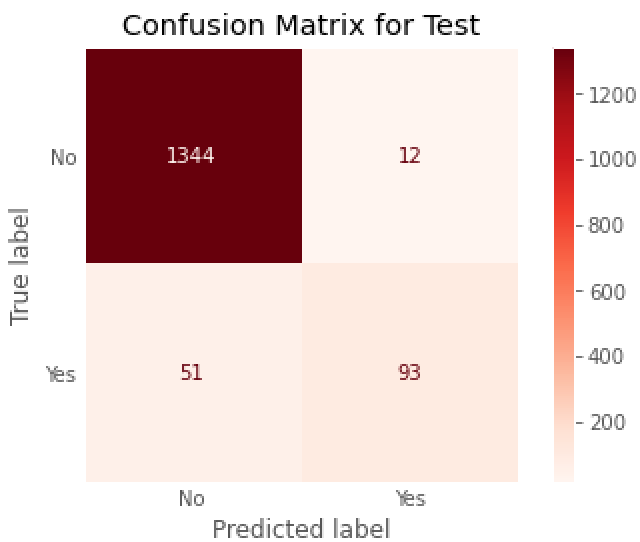


In [148…

```
optimal_threshold_curve = 0.3

scores_opt_curve = get_metrics_score(lg3,X_train3,X_test3,y_train,y_test,statmodel,thre
add_score_model(scores_opt_curve)
```

```
MODEL PERFORMANCE
Accuracy   : Train: 0.963  Test:  0.958
Recall     : Train: 0.72  Test: 0.646
Precision  : Train: 0.874  Test:  0.886
F1         : Train: 0.79  Test: 0.747
```






```
 ROC-AUC Score  :Train::  0.855  Test::  0.818
```

With this model the False negative cases have gone up and recall for test is 0.646 with 95.8 % accuracy. Model is performing well on training and test set. Model has given a balanced performance, if the bank wishes to maintain a balance between recall and precision this model can be used. Area under the curve has decreased as compared to the initial model but the performance is generalized on training and test set.

Building Decision Tree

to classify the significant independent varibles to the Personal Loan dependent variables.

In [157…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 5000 non-null   int64
 1   Experience          5000 non-null   int64
 2   Income              5000 non-null   int64
 3   Family              5000 non-null   category
 4   CCAvg               5000 non-null   float64
```

```
 5   Education           5000 non-null   category
 6   Mortgage            5000 non-null   int64
 7   Personal_Loan       5000 non-null   int32
 8   Securities_Account  5000 non-null   category
 9   CD_Account          5000 non-null   category
 10  Online              5000 non-null   category
 11  CreditCard          5000 non-null   category
 12  Regions             5000 non-null   category
dtypes: category(7), float64(1), int32(1), int64(4)
memory usage: 250.0 KB
```

In [159…

```
X_dt = df.drop('Personal_Loan', axis=1)
y_dt = df['Personal_Loan']
```

In [161…

```
oneHotCols=X_dt.select_dtypes(exclude='number').columns.to_list()
X_dt=pd.get_dummies(X_dt,columns=oneHotCols,drop_first=True)
# Spliting data set
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(X_dt, y_dt, test_size=0
```

Observations on dataset:

- Building a decision tree with imbalanced data as the percentage of people who took loan represent almot 9.5% against 90.5% will end up the decision tree to become more biased toward the dominant classes.
- To handle this imbalanced data set,we can pass a dictionary {0:0.15,1:0.85} to the model to specify the weight of each class and the decision tree will give more weightage to class 1.
- class_weight is a hyperparameter for the decision tree classifier.
- Since not being able to identify a potential customer is the biggest loss as mentioned earlier with logistic regression. Hence, recall is the right metric to check the performance of the model.

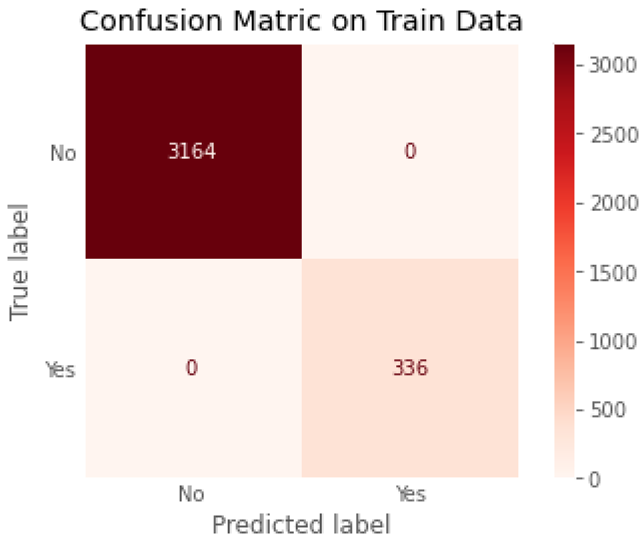
In [163…

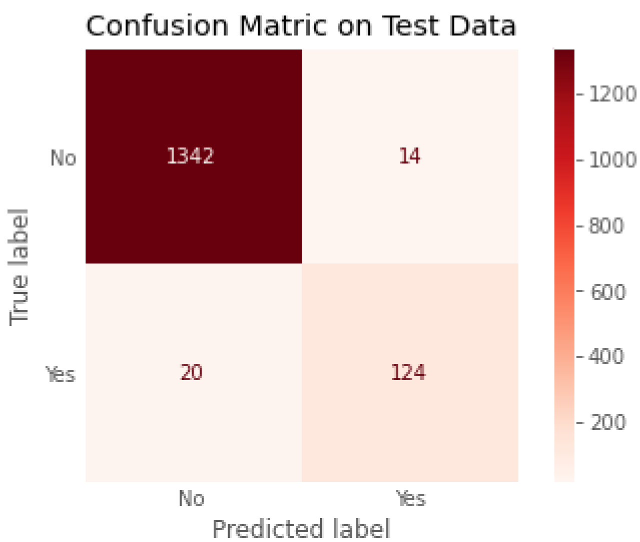
```
##  Function to calculate recall score
def get_recall_score(model):
    '''
    model : classifier to predict values of X

    '''
    ytrain_predict = model.predict(X_train_dt)
    ytest_predict = model.predict(X_test_dt)
    # accuracy on training set
    print("\x1b[0;30;47m \033[1mAccuracy : Train :\033[0m",
          model.score(X_train_dt,y_train_dt),
          "\x1b[0;30;47m \033[1mTest:\033[0m",
          model.score(X_test_dt,y_test_dt))
# accuracy on training set
    print("\x1b[0;30;47m \033[1mRecall   : Train :\033[0m",
          metrics.recall_score(y_train_dt,ytrain_predict),
          "\x1b[0;30;47m \033[1mTest:\033[0m",
          metrics.recall_score(y_test_dt,ytest_predict))
    make_confusion_matrix(y_train_dt,ytrain_predict,"Confusion Matric on Train Data")
    make_confusion_matrix(y_test_dt,ytest_predict,"Confusion Matric on Test Data")
```

In [164…

```
#since data is imbalanced adding weights
model = DecisionTreeClassifier(criterion = 'gini',class_weight={0:0.15,1:0.85}, random_
model.fit(X_train_dt, y_train_dt)
get_recall_score(model)
```

```
Accuracy : Train : 1.0  Test: 0.9773333333333334
Recall   : Train : 1.0  Test: 0.8611111111111112
```





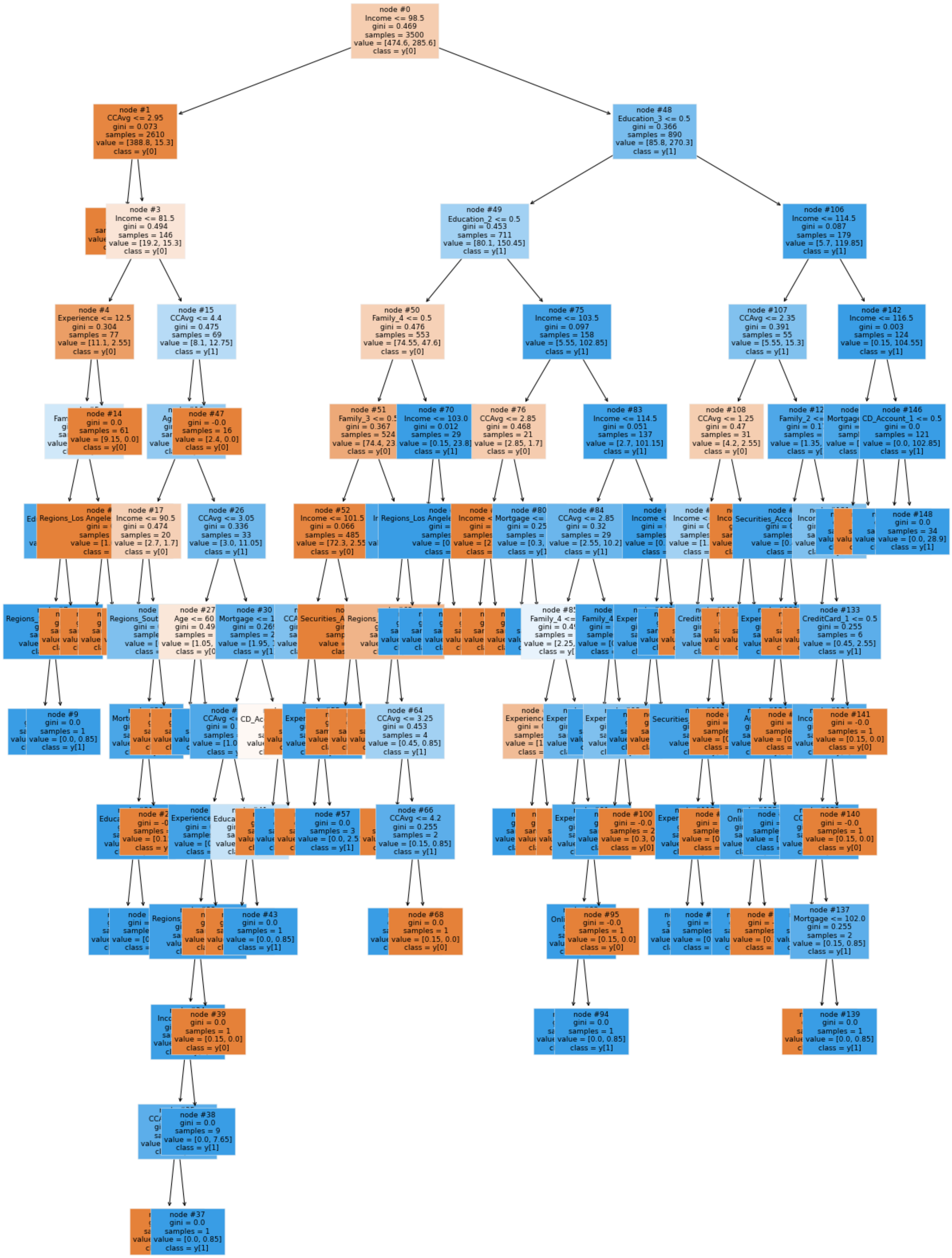
In [166…

```
column_names = list(X_dt.columns)
feature_names = column_names
print(feature_names)
```

```
['Age', 'Experience', 'Income', 'CCAvg', 'Mortgage', 'Family_2', 'Family_3', 'Family_4', 'Education_2', 'Education_3', 'Securities_Account_1', 'CD_Account_1', 'Online_1', 'CreditCard_1', 'Regions_Central', 'Regions_Los Angeles Region', 'Regions_Southern', 'Regions_Superior']
```

In [167…

```
plt.figure(figsize=(20,30))
from sklearn import tree
from sklearn.model_selection import GridSearchCV
out = tree.plot_tree(model,feature_names=feature_names,filled=True,fontsize=9,node_ids=
for o in out:
     arrow = o.arrow_patch
     if arrow is not None:
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```



In [168…

```python
# Text report showing the rules of a decision tree -

print(tree.export_text(model,feature_names=feature_names,show_weights=True))
```

```
|--- Income <= 98.50
```

```
|   |--- CCAvg <= 2.95
|   |   |--- weights: [369.60, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- Income <= 81.50
|   |   |   |--- Experience <= 12.50
|   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |--- Education_3 <= 0.50
|   |   |   |   |   |   |--- Regions_Southern <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |--- Regions_Southern >  0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |--- Education_3 >  0.50
|   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |--- Regions_Los Angeles Region <= 0.50
|   |   |   |   |   |   |--- weights: [1.50, 0.00] class: 0
|   |   |   |   |   |--- Regions_Los Angeles Region >  0.50
|   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |--- Experience >  12.50
|   |   |   |   |--- weights: [9.15, 0.00] class: 0
|   |   |--- Income >  81.50
|   |   |   |--- CCAvg <= 4.40
|   |   |   |   |--- Age <= 46.00
|   |   |   |   |   |--- Income <= 90.50
|   |   |   |   |   |   |--- weights: [2.10, 0.00] class: 0
|   |   |   |   |   |--- Income >  90.50
|   |   |   |   |   |   |--- Regions_Southern <= 0.50
|   |   |   |   |   |   |   |--- Mortgage <= 89.00
|   |   |   |   |   |   |   |   |--- Education_3 <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |   |--- Education_3 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |--- Mortgage >  89.00
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |--- Regions_Southern >  0.50
|   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |--- Age >  46.00
|   |   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |   |--- Age <= 60.00
|   |   |   |   |   |   |   |--- weights: [1.05, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  60.00
|   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |   |--- Mortgage <= 154.00
|   |   |   |   |   |   |   |--- CCAvg <= 4.20
|   |   |   |   |   |   |   |   |--- Experience <= 39.50
|   |   |   |   |   |   |   |   |   |--- Regions_Superior <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- Income <= 82.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- Income >  82.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 7.65] class: 1
|   |   |   |   |   |   |   |   |   |--- Regions_Superior >  0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Experience >  39.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  4.20
|   |   |   |   |   |   |   |   |--- Education_2 <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.60, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Education_2 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |--- Mortgage >  154.00
|   |   |   |   |   |   |   |--- CD_Account_1 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.90, 0.00] class: 0
|   |   |   |   |   |   |   |--- CD_Account_1 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
```

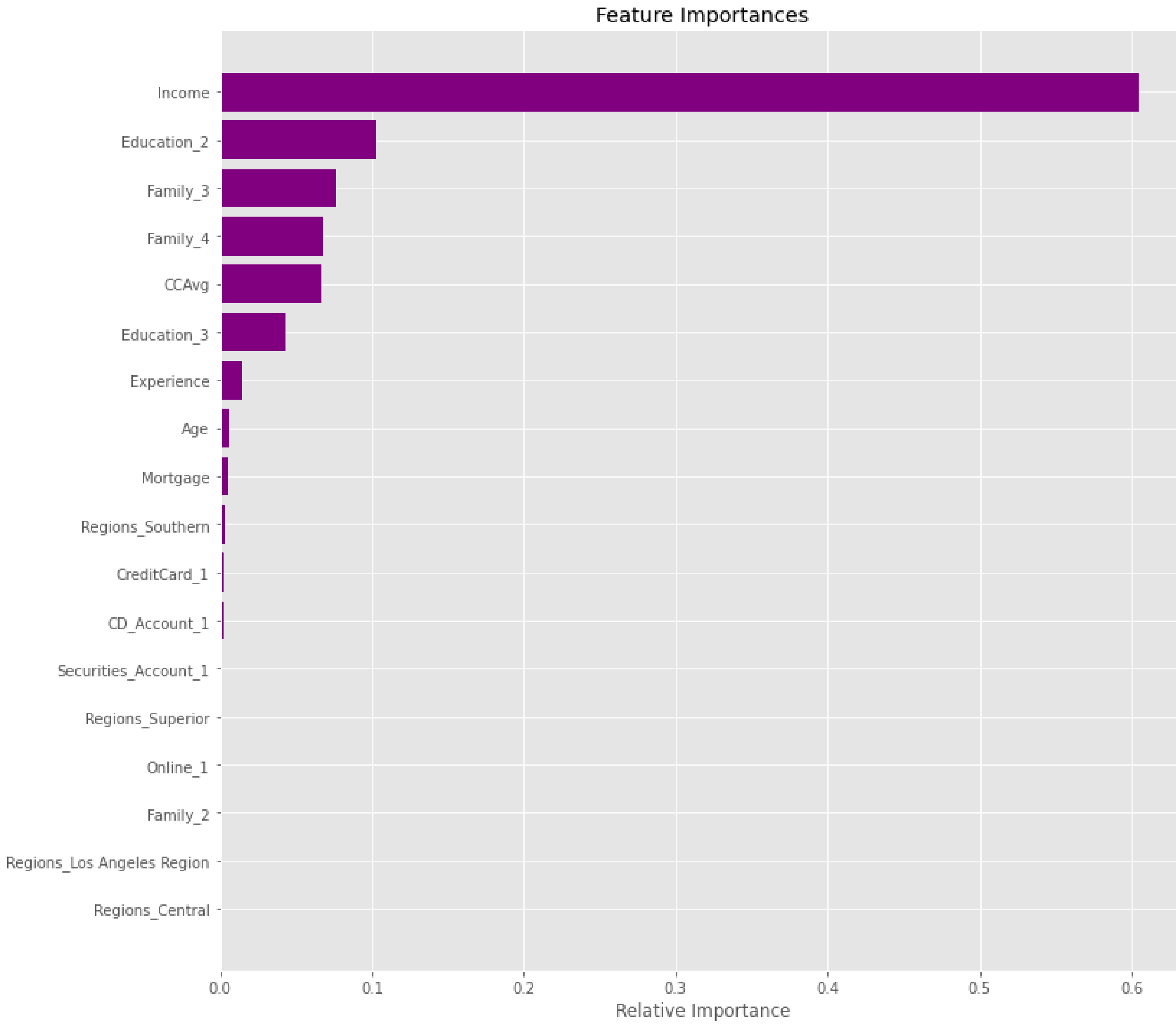
```
|   |   |   |--- CCAvg >  4.40
|   |   |   |   |--- weights: [2.40, 0.00] class: 0
|--- Income >  98.50
|   |--- Education_3 <= 0.50
|   |   |--- Education_2 <= 0.50
|   |   |   |--- Family_4 <= 0.50
|   |   |   |   |--- Family_3 <= 0.50
|   |   |   |   |   |--- Income <= 101.50
|   |   |   |   |   |   |--- CCAvg <= 2.95
|   |   |   |   |   |   |   |--- weights: [0.75, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  2.95
|   |   |   |   |   |   |   |--- Experience <= 16.00
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  16.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.55] class: 1
|   |   |   |   |   |--- Income >  101.50
|   |   |   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |   |   |--- weights: [64.50, 0.00] class: 0
|   |   |   |   |   |   |--- Securities_Account_1 >  0.50
|   |   |   |   |   |   |   |--- weights: [6.90, 0.00] class: 0
|   |   |   |   |--- Family_3 >  0.50
|   |   |   |   |   |--- Income <= 118.00
|   |   |   |   |   |   |--- Regions_Southern <= 0.50
|   |   |   |   |   |   |   |--- weights: [1.65, 0.00] class: 0
|   |   |   |   |   |   |--- Regions_Southern >  0.50
|   |   |   |   |   |   |   |--- CCAvg <= 3.25
|   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  3.25
|   |   |   |   |   |   |   |   |--- CCAvg <= 4.20
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |   |--- CCAvg >  4.20
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |--- Income >  118.00
|   |   |   |   |   |   |--- weights: [0.00, 20.40] class: 1
|   |   |   |--- Family_4 >  0.50
|   |   |   |   |--- Income <= 103.00
|   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |--- Income >  103.00
|   |   |   |   |   |--- Regions_Los Angeles Region <= 0.50
|   |   |   |   |   |   |--- weights: [0.00, 18.70] class: 1
|   |   |   |   |   |--- Regions_Los Angeles Region >  0.50
|   |   |   |   |   |   |--- weights: [0.00, 5.10] class: 1
|   |   |--- Education_2 >  0.50
|   |   |   |--- Income <= 103.50
|   |   |   |   |--- CCAvg <= 2.85
|   |   |   |   |   |--- Income <= 99.50
|   |   |   |   |   |   |--- weights: [0.60, 0.00] class: 0
|   |   |   |   |   |--- Income >  99.50
|   |   |   |   |   |   |--- weights: [1.95, 0.00] class: 0
|   |   |   |   |--- CCAvg >  2.85
|   |   |   |   |   |--- Mortgage <= 41.00
|   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |--- Mortgage >  41.00
|   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |--- Income >  103.50
|   |   |   |   |--- Income <= 114.50
|   |   |   |   |   |--- CCAvg <= 2.85
|   |   |   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |   |   |--- Experience <= 4.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |--- Experience >  4.50
|   |   |   |   |   |   |   |   |--- weights: [1.80, 0.00] class: 0
|   |   |   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |   |   |--- Experience <= 9.50
|   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
```

```
|   |   |   |   |   |   |   |--- Experience >  9.50
|   |   |   |   |   |   |   |   |--- Experience <= 35.50
|   |   |   |   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |   |--- Experience >  35.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  2.85
|   |   |   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 6.80] class: 1
|   |   |   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |   |   |--- Experience <= 20.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |--- Experience >  20.50
|   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |--- Income >  114.50
|   |   |   |   |   |--- Income <= 116.50
|   |   |   |   |   |   |--- Experience <= 10.50
|   |   |   |   |   |   |   |--- weights: [0.00, 2.55] class: 1
|   |   |   |   |   |   |--- Experience >  10.50
|   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |--- Income >  116.50
|   |   |   |   |   |   |--- weights: [0.00, 88.40] class: 1
|   |--- Education_3 >  0.50
|   |   |--- Income <= 114.50
|   |   |   |--- CCAvg <= 2.35
|   |   |   |   |--- CCAvg <= 1.25
|   |   |   |   |   |--- Income <= 108.00
|   |   |   |   |   |   |--- weights: [0.75, 0.00] class: 0
|   |   |   |   |   |--- Income >  108.00
|   |   |   |   |   |   |--- CreditCard_1 <= 0.50
|   |   |   |   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |   |   |   |--- Experience <= 12.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |   |--- Experience >  12.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |--- Securities_Account_1 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |--- CreditCard_1 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |--- CCAvg >  1.25
|   |   |   |   |   |--- Income <= 99.50
|   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |--- Income >  99.50
|   |   |   |   |   |   |--- weights: [2.55, 0.00] class: 0
|   |   |   |--- CCAvg >  2.35
|   |   |   |   |--- Family_2 <= 0.50
|   |   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |   |--- Experience <= 39.00
|   |   |   |   |   |   |   |--- Age <= 32.00
|   |   |   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |--- Age >  32.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 8.50] class: 1
|   |   |   |   |   |   |--- Experience >  39.00
|   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |--- Securities_Account_1 >  0.50
|   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |--- Family_2 >  0.50
|   |   |   |   |   |--- Income <= 100.00
|   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |--- Income >  100.00
```

```
|   |   |   |   |   |   |--- CreditCard_1 <= 0.50
|   |   |   |   |   |   |   |--- Income <= 112.00
|   |   |   |   |   |   |   |   |--- CCAvg <= 4.20
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |   |--- CCAvg >  4.20
|   |   |   |   |   |   |   |   |   |--- Mortgage <= 102.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Mortgage >  102.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |--- Income >  112.00
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |--- CreditCard_1 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |--- Income >  114.50
|   |   |   |--- Income <= 116.50
|   |   |   |   |--- Mortgage <= 94.50
|   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |--- Mortgage >  94.50
|   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |--- Income >  116.50
|   |   |   |   |--- CD_Account_1 <= 0.50
|   |   |   |   |   |--- weights: [0.00, 73.95] class: 1
|   |   |   |   |--- CD_Account_1 >  0.50
|   |   |   |   |   |--- weights: [0.00, 28.90] class: 1
```

In [169…

```
importances = model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='purple', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- Using GridSearch for Hyperparameter tuning of our tree model
- Grid search is a tuning technique that attempts to compute the optimum values of hyperparameters.
- It is an exhaustive search that is performed on a the specific parameter values of a model.
- The parameters of the estimator/model used to apply these methods are optimized by cross-validated - grid-search over a parameter grid.
- Let's see if we can improve our model performance even more.

In [170…

```
# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1)

# Grid of parameters to choose from

parameters = {'max_depth': np.arange(1,10),
              'min_samples_leaf': [1, 2, 5, 7, 10,15,20],
              'max_leaf_nodes' : [5, 10,15,20,25,30],
              }

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
```

```
grid_obj = GridSearchCV(estimator, parameters, scoring=acc_scorer,cv=5)
grid_obj = grid_obj.fit(X_train_dt, y_train_dt)

# Set the clf to the best combination of parameters
estimator = grid_obj.best_estimator_
estimator
```

Out[170…

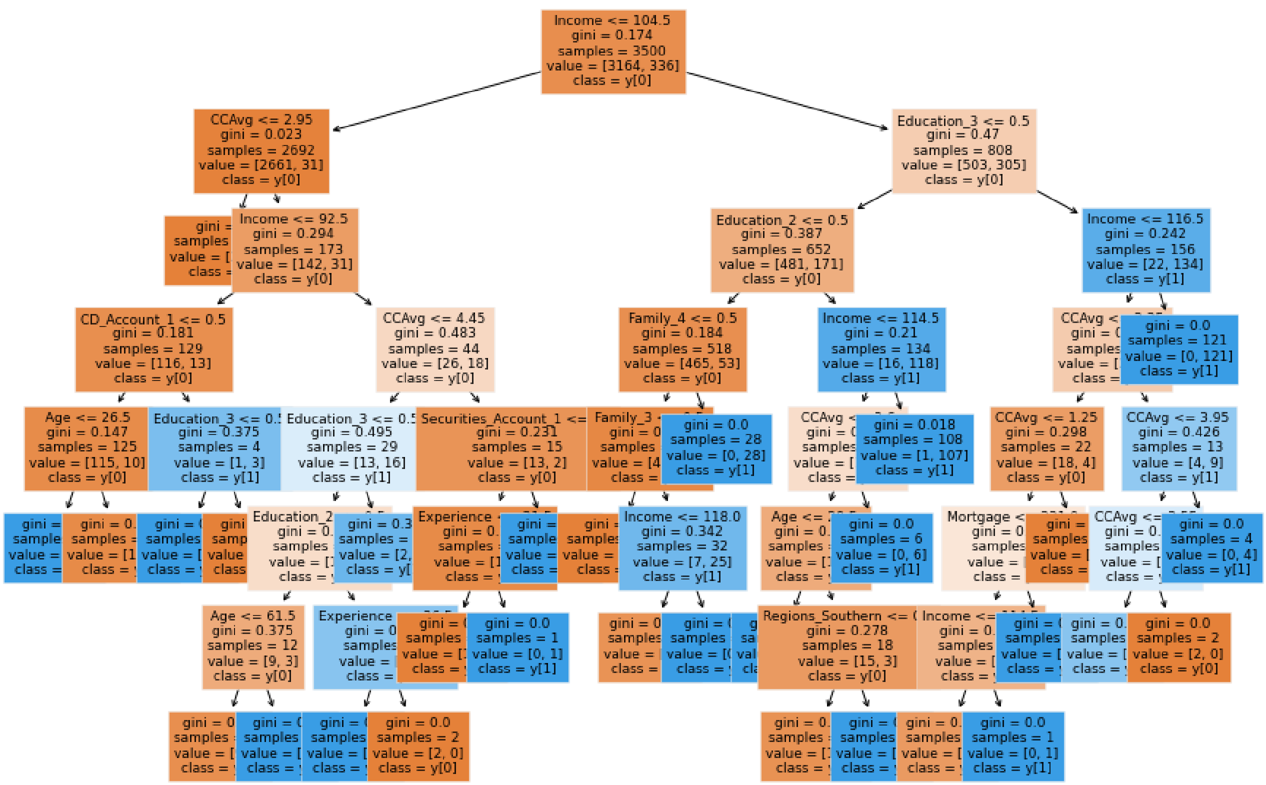
```
DecisionTreeClassifier(max_depth=9, max_leaf_nodes=30, random_state=1)
```

In [171…

```
# Fit the best algorithm to the data.
estimator.fit(X_train_dt, y_train_dt)
ytrain_predict=estimator.predict(X_train_dt)
ytest_predict=estimator.predict(X_test_dt)
```

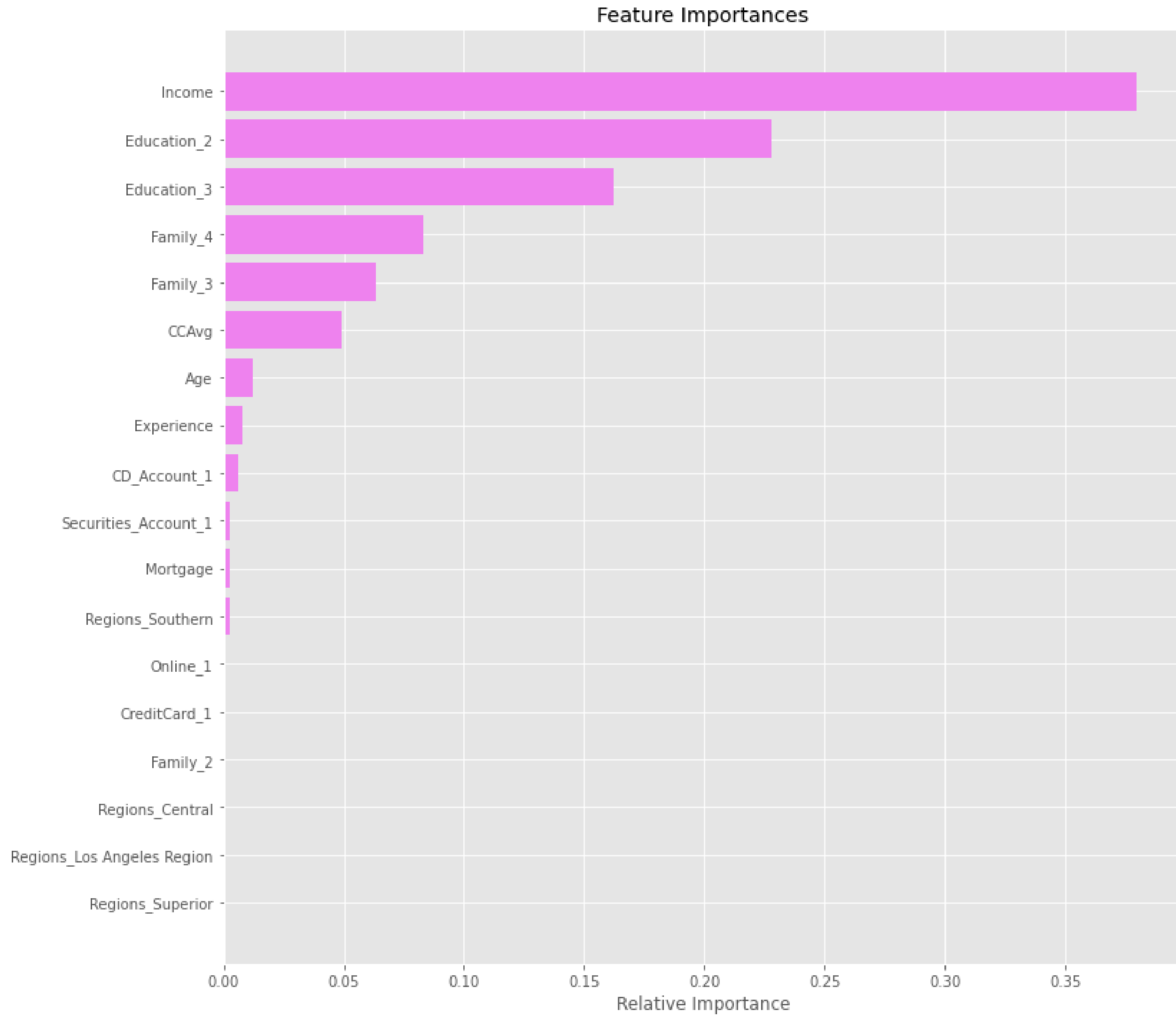
In [173…

```
plt.figure(figsize=(15,10))

out = tree.plot_tree(estimator,feature_names=feature_names,filled=True,fontsize=9,node_
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```



In [174…

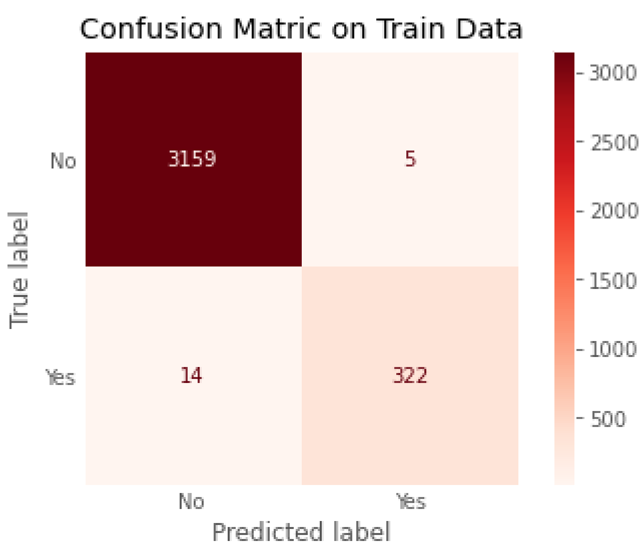
```
importances = estimator.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

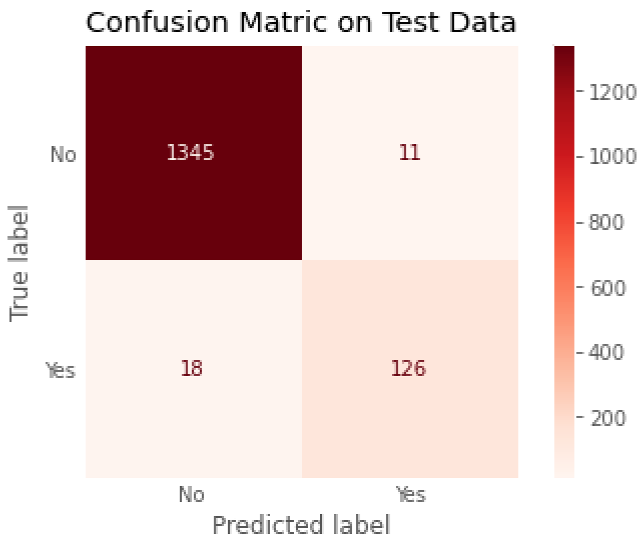


In [175…

```
get_recall_score(estimator)
```

```
Accuracy : Train : 0.9945714285714286  Test: 0.9806666666666667
Recall   : Train : 0.9583333333333334  Test: 0.875
```





- With HyperParameter max_depth=6, max_leaf_nodes=20, min_samples_leaf=7 the overfitting on train has reduced, but the recall for test has not improved.
- Important features are Income,Education 2 and Education 3, Family 4, Family 3, CCavg & Age.
- false negatives are 18.We don't want to loose opportunity in - predicting this customers. so Let see if instead of pre pruning , post pruning helps in reducing false negative.

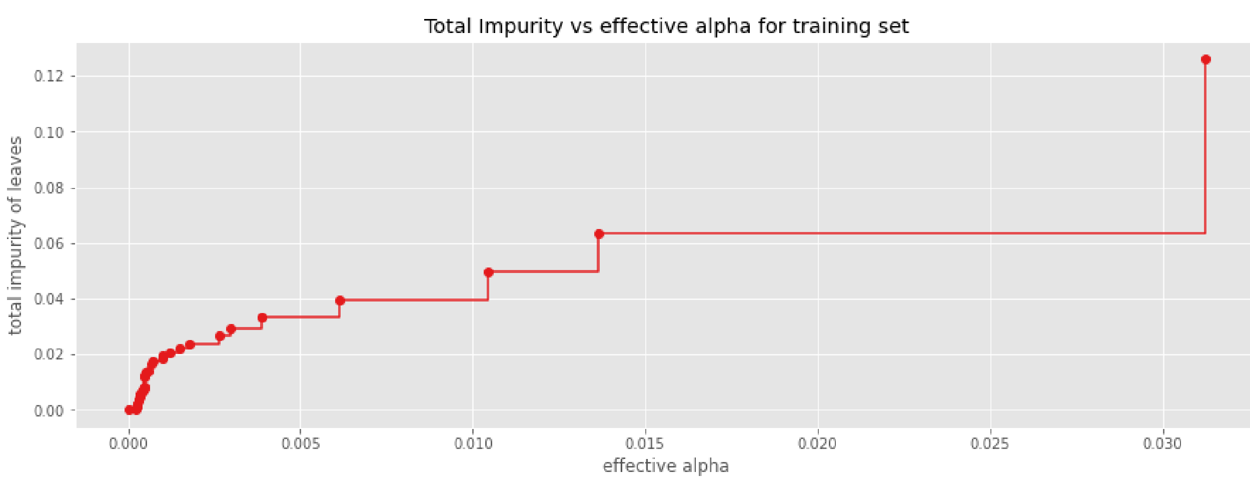
Cost Complexity Pruning

In [177...

```
clf = DecisionTreeClassifier(random_state=1)
path = clf.cost_complexity_pruning_path(X_train_dt, y_train_dt)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

In [178...

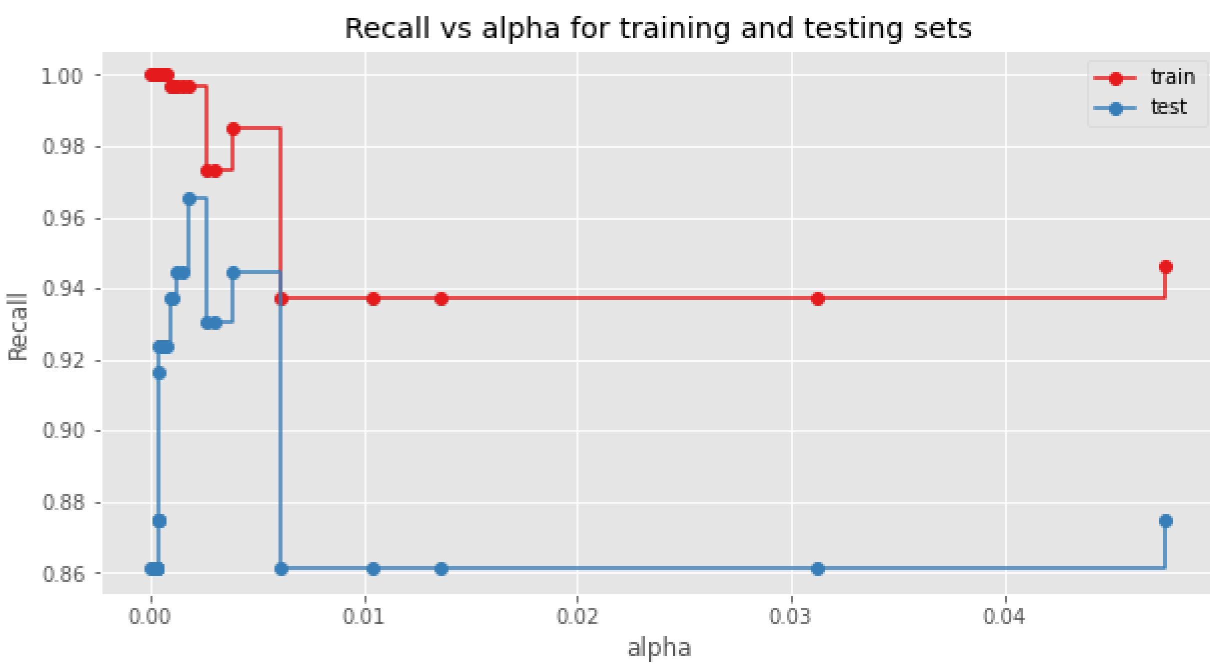
```
fig, ax = plt.subplots(figsize=(15,5))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.show()
```



Next, we train a decision tree using the effective alphas. We will set these values of alpha and pass it to the ccp_alpha parameter of our DecisionTreeClassifier. By looping over the alphas array, we will find the accuracy on both Train and Test parts of our dataset.

In [179…
```
clfs = []
accuracy_train=[]
accuracy_test=[]
recall_train=[]
recall_test=[]
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha,class_weight = {0:
    clf.fit(X_train_dt, y_train_dt)
    y_train_pred=clf.predict(X_train_dt)
    y_test_pred=clf.predict(X_test_dt)
    accuracy_train.append(clf.score(X_train_dt,y_train_dt))
    accuracy_test.append(clf.score(X_test_dt,y_test_dt))
    recall_train.append(metrics.recall_score(y_train_dt,y_train_pred))
    recall_test.append(metrics.recall_score(y_test_dt,y_test_pred))
    clfs.append(clf)
```

In [180…
```
fig, ax = plt.subplots(figsize=(10,5))
ax.set_xlabel("alpha")
ax.set_ylabel("Recall")
ax.set_title("Recall vs alpha for training and testing sets")
ax.plot(ccp_alphas, recall_train, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, recall_test, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



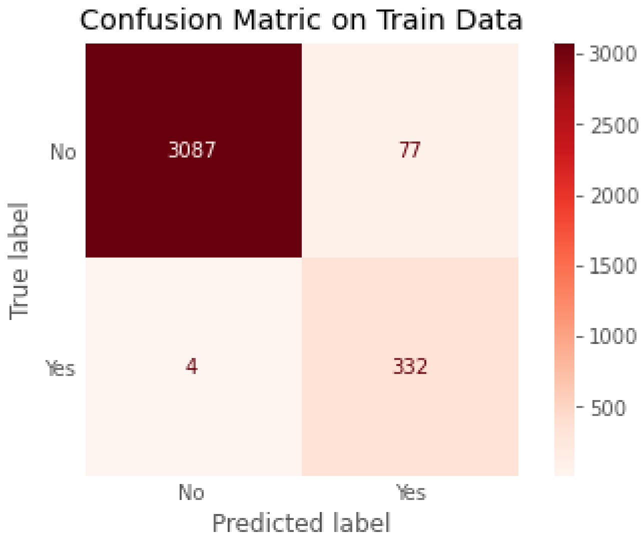
Creating model with 0.002 ccp_alpha

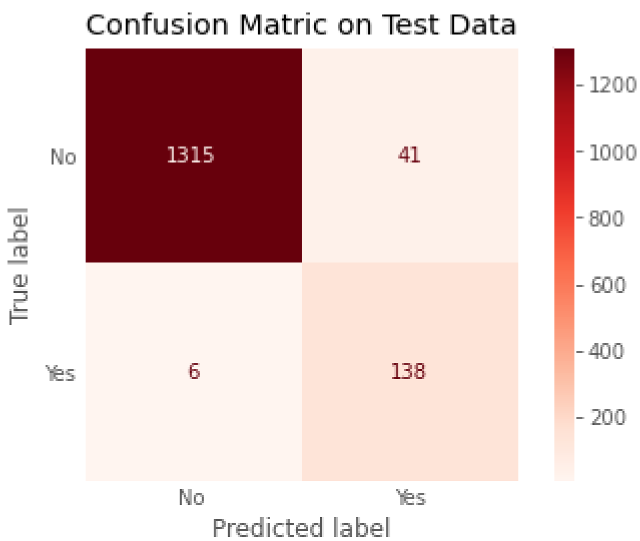
In [181…
```
best_model = DecisionTreeClassifier(ccp_alpha=0.002,
                        class_weight={0: 0.15, 1: 0.85}, random_state=1)
best_model.fit(X_train_dt, y_train_dt)
```

Out[181…
```
DecisionTreeClassifier(ccp_alpha=0.002, class_weight={0: 0.15, 1: 0.85},
                       random_state=1)
```

In [182…
```
get_recall_score(best_model)
```

```
Accuracy : Train : 0.9768571428571429  Test: 0.9686666666666667
Recall   : Train : 0.9880952380952381  Test: 0.9583333333333334
```

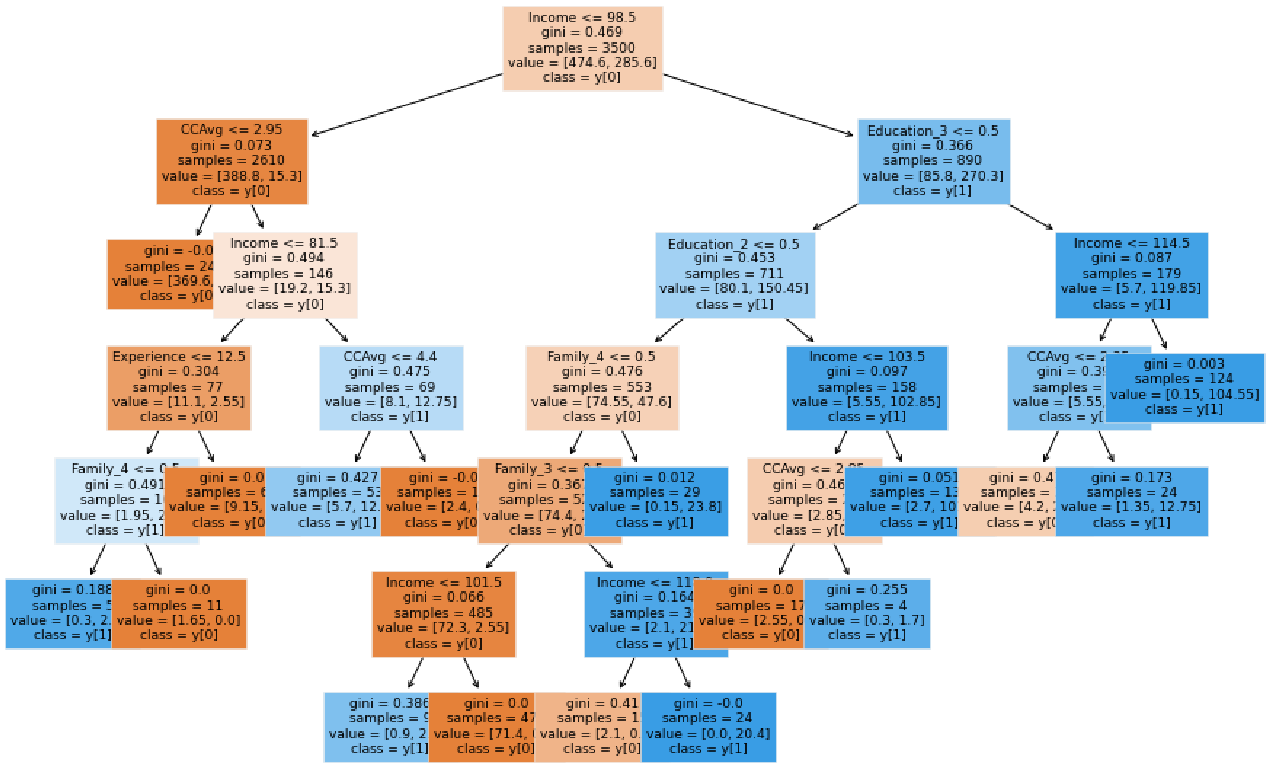




The Recall on train and test indicate we have created a generalized model. with 96 % accuracy and reduced False negatives

In [183…

```
plt.figure(figsize=(15,10))

out = tree.plot_tree(best_model,feature_names=feature_names,filled=True,fontsize=9,node
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```



In [184…

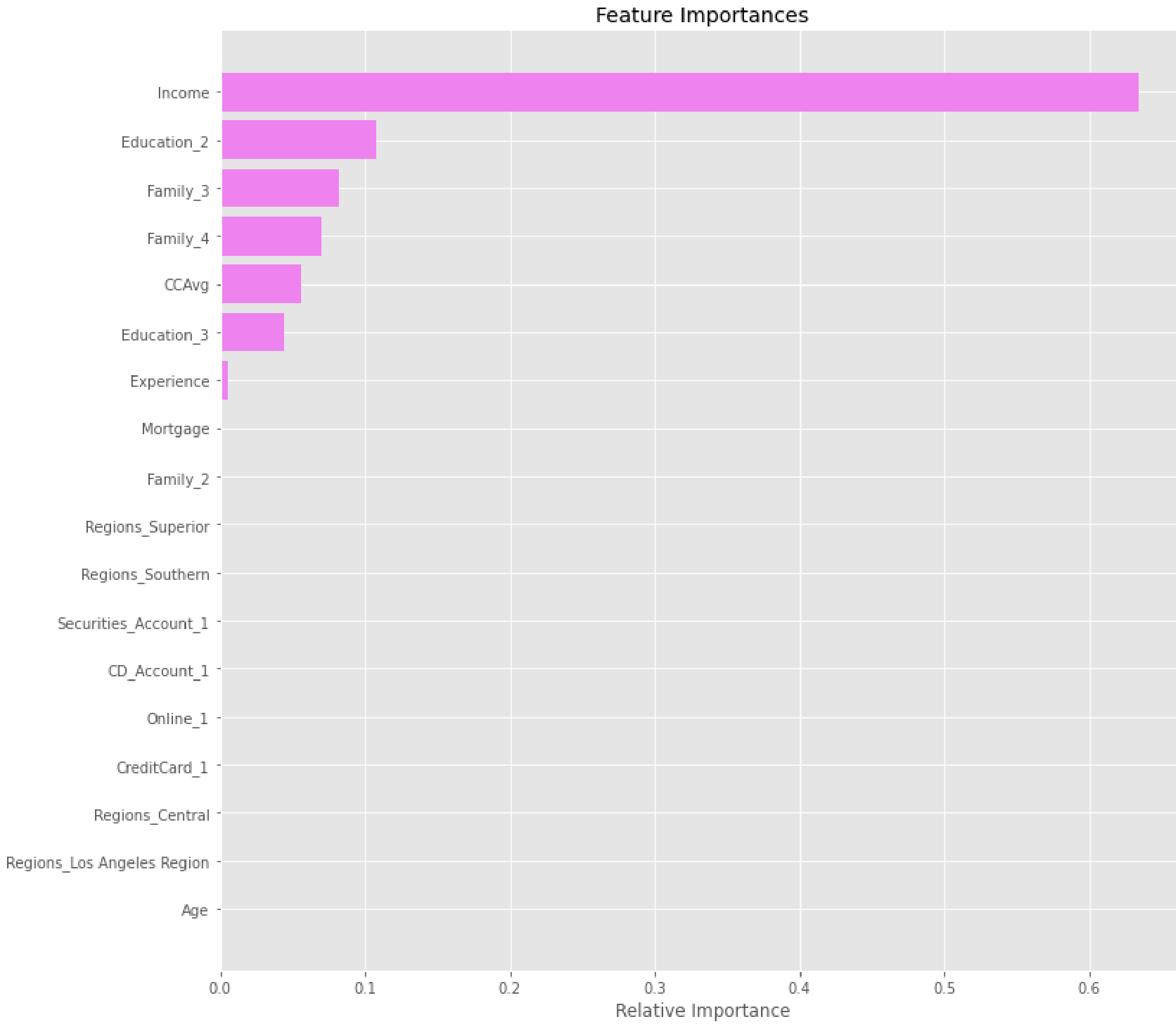
```python
# Text report showing the rules of a decision tree -

print(tree.export_text(best_model,feature_names=feature_names,show_weights=True))
```

```
|--- Income <= 98.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [369.60, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- Income <= 81.50
|   |   |   |--- Experience <= 12.50
|   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |--- weights: [0.30, 2.55] class: 1
|   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |--- weights: [1.65, 0.00] class: 0
|   |   |   |--- Experience >  12.50
|   |   |   |   |--- weights: [9.15, 0.00] class: 0
|   |   |--- Income >  81.50
|   |   |   |--- CCAvg <= 4.40
|   |   |   |   |--- weights: [5.70, 12.75] class: 1
|   |   |   |--- CCAvg >  4.40
|   |   |   |   |--- weights: [2.40, 0.00] class: 0
|--- Income >  98.50
|   |--- Education_3 <= 0.50
|   |   |--- Education_2 <= 0.50
|   |   |   |--- Family_4 <= 0.50
|   |   |   |   |--- Family_3 <= 0.50
|   |   |   |   |   |--- Income <= 101.50
|   |   |   |   |   |   |--- weights: [0.90, 2.55] class: 1
|   |   |   |   |   |--- Income >  101.50
|   |   |   |   |   |   |--- weights: [71.40, 0.00] class: 0
|   |   |   |   |--- Family_3 >  0.50
|   |   |   |   |   |--- Income <= 118.00
|   |   |   |   |   |   |--- weights: [2.10, 0.85] class: 0
|   |   |   |   |   |--- Income >  118.00
|   |   |   |   |   |   |--- weights: [0.00, 20.40] class: 1
|   |   |   |--- Family_4 >  0.50
```

```
|   |   |   |   |--- weights: [0.15, 23.80] class: 1
|   |   |--- Education_2 >  0.50
|   |   |   |--- Income <= 103.50
|   |   |   |   |--- CCAvg <= 2.85
|   |   |   |   |   |--- weights: [2.55, 0.00] class: 0
|   |   |   |   |--- CCAvg >  2.85
|   |   |   |   |   |--- weights: [0.30, 1.70] class: 1
|   |   |   |--- Income >  103.50
|   |   |   |   |--- weights: [2.70, 101.15] class: 1
|   |--- Education_3 >  0.50
|   |   |--- Income <= 114.50
|   |   |   |--- CCAvg <= 2.35
|   |   |   |   |--- weights: [4.20, 2.55] class: 0
|   |   |   |--- CCAvg >  2.35
|   |   |   |   |--- weights: [1.35, 12.75] class: 1
|   |   |--- Income >  114.50
|   |   |   |--- weights: [0.15, 104.55] class: 1
```

In [185…

```
importances = best_model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- We are getting a higher recall on test data between 0.002 to 0.005. Will choosed alpha as 0.002.
- The Recall on train and test indicate we have created a generalized model. with 96 % accuracy and reduced False negatives.
- Important features : Income, Graduate education, Family member 3 and 4, Ccavg, Advanced education, Age.
- This is the best model as false negative is only 6 on Testdata.

In [186…

```
comparison_frame = pd.DataFrame({'Model':['Logisitic Regression with Optimal Threshold
                                          'Initial decision tree model',
                                          'Decision treee with hyperparameter tuning',
                                          'Decision tree with post-pruning'],
                                          'Train_accuracy':[0.92,1,0.99,0.98],
                                          'Test_accuracy':[0.91,0.98,0.98,0.97],
                                          'Train_Recall':[0.90,1,0.92,0.98],
                                          'Test_Recall':[0.88,0.86,0.84,0.96]})

comparison_frame
```

Out[186…

	Model	Train_accuracy	Test_accuracy	Train_Recall	Test_Recall
0	Logisitic Regression with Optimal Threshold 0.104	0.92	0.91	0.90	0.88

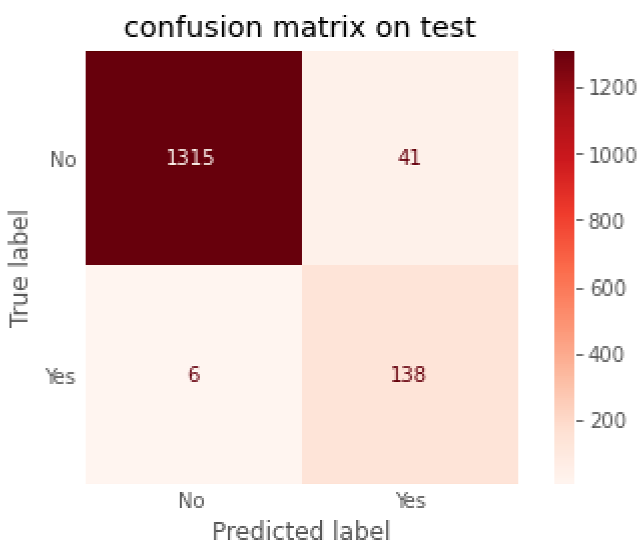
	Model	Train_accuracy	Test_accuracy	Train_Recall	Test_Recall
1	Initial decision tree model	1.00	0.98	1.00	0.86
2	Decision treee with hyperparameter tuning	0.99	0.98	0.92	0.84
3	Decision tree with post-pruning	0.98	0.97	0.98	0.96

- Decision tree model post pruning has given us best recall scores on data with 97% accuracy . -- * Exploratory data analysis also suggested income and education were important features in deciding if person will borrow personal loan. so choosing Decision Tree with post-pruning for our prediction.

In [187…

```
y_pred = best_model.predict(X_test_dt)
print(classification_report(y_test_dt,y_pred))
make_confusion_matrix(y_test,y_pred,"confusion matrix on test")
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.98      1356
           1       0.77      0.96      0.85       144

    accuracy                           0.97      1500
   macro avg       0.88      0.96      0.92      1500
weighted avg       0.97      0.97      0.97      1500
```



Observation

After Post Pruning ,the false negative has reduced to 6.The accuracy on test data is 96% & Recall is 96% after choosing optimal cc-alpha

Conclusions

- We analyzed the Personal Loan campaign data using EDA and by using different models like Logistic Regression and Decision Tree Classifier to build a likelihood of Customer buying Loan.
- First we built model using Logistic Regression and performance metric used was Recall. The most important features for classification were Income,Education, CD account ,Family and

CCAvg .

- Coefficient of Income, Graduate and Advanced Education, Family_3,Family 4,CCavg,CD account,Age, are positive , ie a one unit increase in these will lead to increase in chances of a person borrowing loan
- Coefficient of Securities account,online ,Family_2 credit card are negative increase in these will lead to decrease in chances of a person borrowing a loan.
- We also improved the performance using ROC-AUC curve and optimal threshold .This was best model with high recall and accuracy .
- Decision tree can easily overfit. They require less datapreprocessing compared to logistic Regression and are easy to understand.
- We used decision trees with prepruning and post pruning. The Post pruning model gave 96 % recall with 96% accuracy.
- Income, Customers with graduate degree, customers having 3 family members are some of the most important variables in predicting if the customers will purchase a personal loan

```
In [ ]:
```