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, and 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).

A campaign that the bank ran last year for liability customers showed a healthy conversion rate of over 9% success. This has encouraged the retail marketing department to devise campaigns with better target marketing to increase the success ratio.

Objective

1. 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.

Variables to be analyzed

- 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)?

Importing modules and packages

In [2]:

```
#import the important packages
import warnings
warnings.filterwarnings('ignore')
import pandas as pd #library used for data manipulation and analysis
import numpy as np # library used for working with arrays.
import matplotlib.pyplot as plt # library for plots and visualisations
import seaborn as sns # library for visualisations
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from scipy.stats import pearsonr
%matplotlib inline

import scipy.stats as stats # this library contains a large number of probability distributions as well as a grow
```

In [3]:

```
!pip install uszipcode
```

```
Requirement already satisfied: uszipcode in d:\anaconda\lib\site-packages (0.2.6)
Requirement already satisfied: pathlib-mate in d:\anaconda\lib\site-packages (from uszipcode) (1.0.1)
Requirement already satisfied: SQLAlchemy in d:\anaconda\lib\site-packages (from uszipcode) (1.3.20)
Requirement already satisfied: attrs in d:\anaconda\lib\site-packages (from uszipcode) (20.3.0)
Requirement already satisfied: requests in d:\anaconda\lib\site-packages (from uszipcode) (2.24.0)
Requirement already satisfied: atomicwrites in d:\anaconda\lib\site-packages (from pathlib-mate->uszipcode) (1.4.
0)
Requirement already satisfied: six in d:\anaconda\lib\site-packages (from pathlib-mate->uszipcode) (1.15.0)
Requirement already satisfied: autopep8 in d:\anaconda\lib\site-packages (from pathlib-mate->uszipcode) (1.5.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in d:\anaconda\lib\site-packages (from req
uests->uszipcode) (1.25.11)
Requirement already satisfied: chardet<4,>=3.0.2 in d:\anaconda\lib\site-packages (from requests->uszipcode) (3.0
.4)
Requirement already satisfied: certifi>=2017.4.17 in d:\anaconda\lib\site-packages (from requests->uszipcode) (20
```

```
20.6.20)
Requirement already satisfied: idna<3,>=2.5 in d:\anaconda\lib\site-packages (from requests->uszipcode) (2.10)
Requirement already satisfied: toml in d:\anaconda\lib\site-packages (from autopep8->pathlib-mate->uszipcode) (0.
10.1)
Requirement already satisfied: pycodestyle>=2.6.0 in d:\anaconda\lib\site-packages (from autopep8->pathlib-mate->
uszipcode) (2.6.0)
```

In [4]:
```
from uszipcode import Zipcode, SearchEngine
```

Reading and pre-processing the data

In [5]:
```
data = pd.read_csv('Loan_Modelling.csv', index_col = 0) #reading the data
np.random.seed(1)
data.sample(n = 10) #random sample of 10 values
```

Out[5]:

	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Account	CD_Account	Online	Cred
ID													
2765	31	5	84	91320	1	2.9	3	105	0	0	0	0	
4768	35	9	45	90639	3	0.9	1	101	0	1	0	0	
3815	34	9	35	94304	3	1.3	1	0	0	0	0	0	
3500	49	23	114	94550	1	0.3	1	286	0	0	0	1	
2736	36	12	70	92131	3	2.6	2	165	0	0	0	1	
3923	31	4	20	95616	4	1.5	2	0	0	0	0	1	
2702	50	26	55	94305	1	1.6	2	0	0	0	0	1	
1180	36	11	98	90291	3	1.2	3	0	0	1	0	0	
933	51	27	112	94720	3	1.8	2	0	0	1	1	1	
793	41	16	98	93117	1	4.0	3	0	0	0	0	0	

In [6]:
```
data.shape  #the shape of the data
```

Out[6]: (5000, 13)

In [7]:
```
data.info() #info of all the variables
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 1 to 5000
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   ZIPCode             5000 non-null   int64  
 4   Family              5000 non-null   int64  
 5   CCAvg               5000 non-null   float64
 6   Education           5000 non-null   int64  
 7   Mortgage            5000 non-null   int64  
 8   Personal_Loan       5000 non-null   int64  
 9   Securities_Account  5000 non-null   int64  
 10  CD_Account          5000 non-null   int64  
 11  Online              5000 non-null   int64  
 12  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(12)
memory usage: 546.9 KB
```

Observation:

1. THere are a total of 5000 entris with 13 variables
2. Variables like Personal_loan, Securities_Account, CD_Account, Online, CreditCard need to be addressed as categorical variables.

In [8]:
```
data.isnull().sum().sort_values(ascending=False) #sum of all the null values per variable
```

Out[8]:
```
CreditCard            0
Online                0
CD_Account            0
```

```
Securities_Account    0
Personal_Loan         0
Mortgage              0
Education             0
CCAvg                 0
Family                0
ZIPCode               0
Income                0
Experience            0
Age                   0
dtype: int64
```

Observation:

1. No null values are observed within all the variables.

In [9]:
```
data['Personal_Loan'] = data['Personal_Loan'].astype('category')
data['Securities_Account'] = data['Securities_Account'].astype('category')
data['CD_Account'] = data['CD_Account'].astype('category') #converting to categorical variables
data['Online'] = data['Online'].astype('category')
data['Family'] = data['Family'].astype('category')
data['Education'] = data['Education'].astype('category')
data['CreditCard'] = data['CreditCard'].astype('category')
```

In [10]:
```
zip=data['ZIPCode'] #converting all the ZIp codes to major cities
zip=zip.astype(int)
zip=zip.astype(str)
major_city = []
for i in zip:
    search = SearchEngine()
    zipcode = search.by_zipcode(i).major_city
    major_city.append(zipcode)
```

In [11]:
```
data = data.assign(Major_city=major_city) #adding the major_cities column to the main data set.
data.drop(['ZIPCode'],axis=1,inplace=True)
data['Major_city'] = data['Major_city'].astype('category')
```

In [12]:
```
data.nunique(dropna = False) #unique values from each variable
```

Out[12]:
```
Age                    45
Experience             47
Income                162
Family                  4
CCAvg                 108
Education               3
Mortgage              347
Personal_Loan           2
Securities_Account      2
CD_Account              2
Online                  2
CreditCard              2
Major_city            245
dtype: int64
```

In [13]:
```
data.info() #checking the data info now
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 1 to 5000
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  Major_city          4966 non-null   category
dtypes: category(8), float64(1), int64(4)
memory usage: 291.0 KB
```

In [14]:
```python
data.isnull().sum().sort_values(ascending=False)
```
Out[14]:
```
Major_city            34
CreditCard             0
Online                 0
CD_Account             0
Securities_Account     0
Personal_Loan          0
Mortgage               0
Education              0
CCAvg                  0
Family                 0
Income                 0
Experience             0
Age                    0
dtype: int64
```

Observation:

1. After converting the zip-codes to major cities, some of the major-city values are observed to be none, this variable will be taken as categorical variables, so none can be a type of category

In [15]:
```python
data['Major_city'] = data['Major_city'].astype(str).replace('nan', 'is_missing').astype('category') #converting t
```
In [16]:
```python
num = data._get_numeric_data() #some values in the Experience column are negative, so converting the values to 0.
num[num < 0] = 0
```
In [17]:
```python
print(data.describe().T) #statistics of all the variables
```
```
             count       mean         std   min   25%   50%    75%    max
Age         5000.0  45.338400   11.463166  23.0  35.0  45.0   55.0   67.0
Experience  5000.0  20.119600   11.440484   0.0  10.0  20.0   30.0   43.0
Income      5000.0  73.774200   46.033729   8.0  39.0  64.0   98.0  224.0
CCAvg       5000.0   1.937938    1.747659   0.0   0.7   1.5    2.5   10.0
Mortgage    5000.0  56.498800  101.713802   0.0   0.0   0.0  101.0  635.0
```

Observation:

1. There are no negative values observed, data has been cleaned.
2. High variance in teh Income data, there is a big range in the income data.
3. The values in the Mortgage are observed towards the 75th percentile

In [18]:
```python
print(data.describe(include = 'category')) #categorical variables statistics
```
```
        Family  Education  Personal_Loan  Securities_Account  CD_Account  \
count     5000       5000           5000                5000        5000
unique       4          3              2                   2           2
top          1          1              0                   0           0
freq      1472       2096           4520                4478        4698

        Online  CreditCard   Major_city
count     5000        5000         5000
unique       2           2          245
top          1           0  Los Angeles
freq      2984        3530          375
```

Observation:

1. Family has 4 unique values, with 1 being the highest.
2. Education has the 3 unqiue valus with 1 being the highest.
3. Personal Loan, the most frequent is 0.
4. Secruities_acoount, CD_account, Online, CreditCard all have values of 2.
5. There are 245 unique Major Cities, if the variable isn't providing useful information, it may be dropped.

EDA

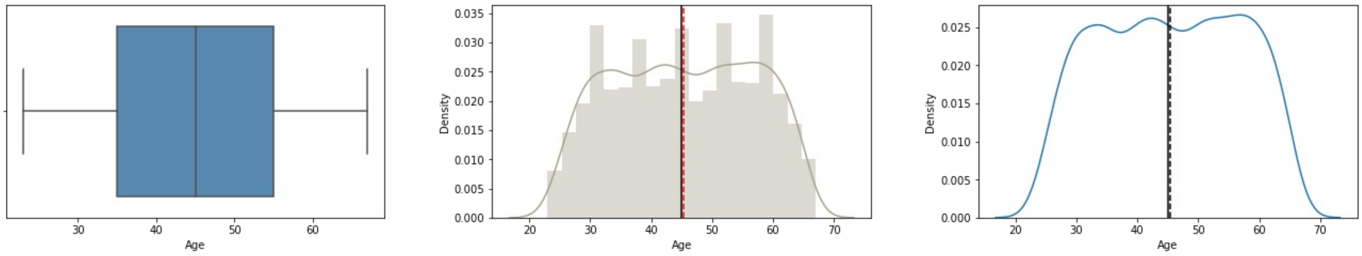
In [19]:
```python
def histogram_boxplot(feature):
    """ Boxplot and histogram combined
    feature: 1-d feature array
    """
```

```
    figure, (ax_box2, ax_hist2, ax_hist3) = plt.subplots(
        nrows = 1, ncols=3,# Number of rows of the subplot grid= 2
        figsize  = (20,5)) # creating the 2 subplots
    figure.tight_layout(pad = 7)
    sns.boxplot(x = feature,ax=ax_box2, color = '#4B8BBE', orient = 'v') # boxplot will be created
    sns.distplot(feature, kde=True, ax=ax_hist2, color = '#a9a38f') # For histogram
    sns.distplot(feature, kde= True, ax=ax_hist3, hist = False) #Making an outline of the histogram
    ax_hist2.axvline(np.mean(feature), color='r', linestyle='--') # Add mean to the histogram
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-') # Add median to the histogram
    ax_hist3.axvline(np.mean(feature), color = 'black', linestyle = '--') #Adding mean to second histogram
    ax_hist3.axvline(np.median(feature), color='black', linestyle='-') #Adding median to second histogram
```

Univariate Analysis

Observations on Age

In [20]:
```
histogram_boxplot(data.Age)
```

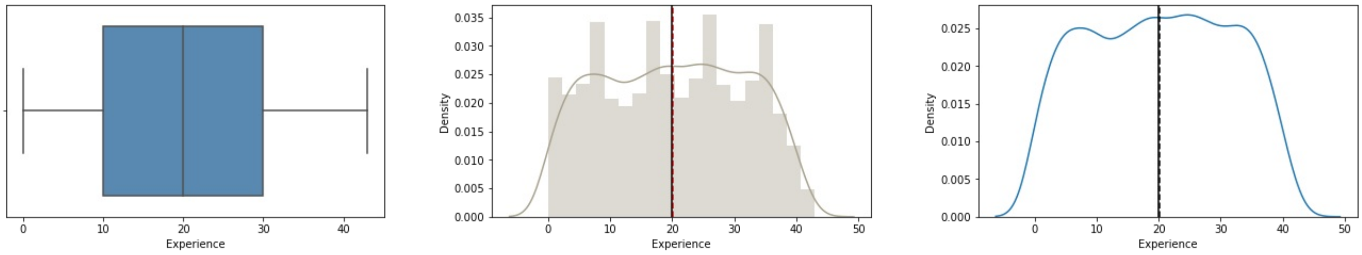


Observation:

1. There seems to be no outliers present
2. Data is evenly distributed to the right and left of the mean.

Observations on Experience

In [21]:
```
histogram_boxplot(data['Experience'])
```

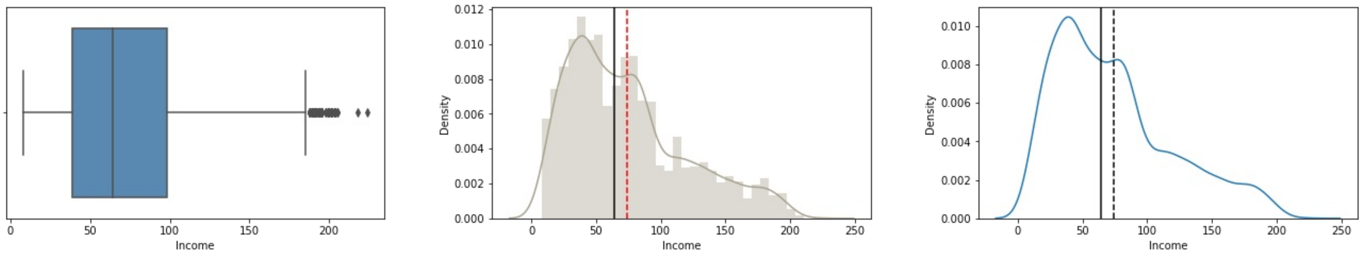


Observation:

1. Data is most eveenly distributed to the right and left of the mean.
2. Mean appears to be at 20.
3. No outliers visible.

Observations on Income

In [22]:
```
histogram_boxplot(data['Income'])
```

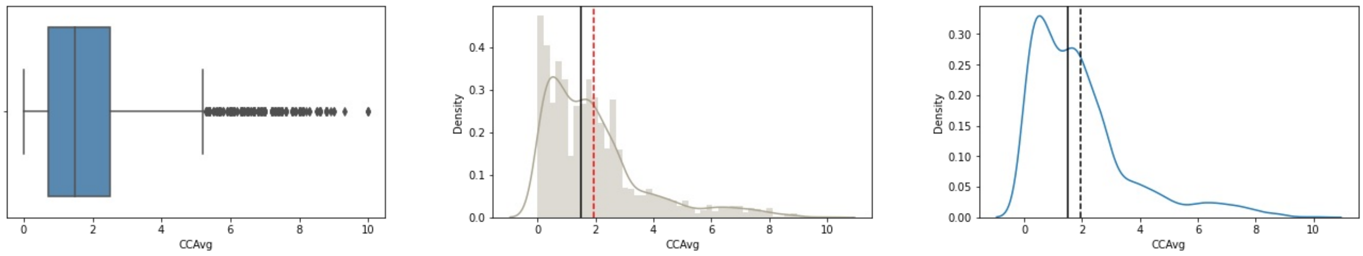


Observation:

1. No outliers visible.
2. As the income increases the density decreases.
3. THe density is the highest around 50 income.

Observations on CCAvg

In [23]:
```
histogram_boxplot(data['CCAvg'])
```

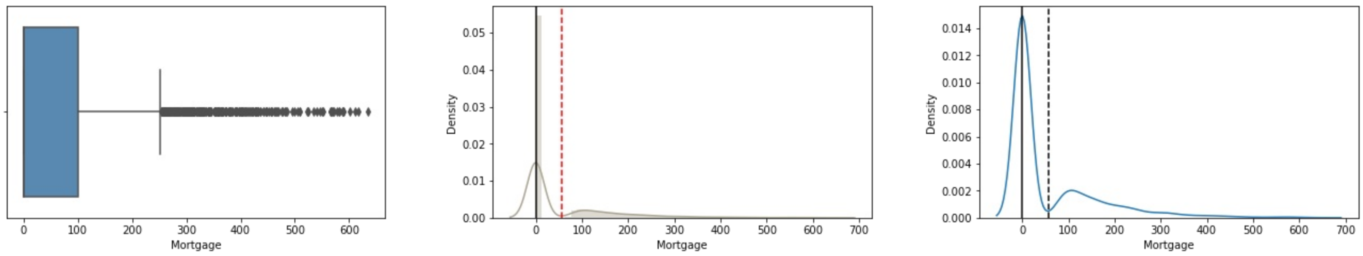


Observation:

1. No outliers visible.
2. The desnity decreases mostly till 4 CCAvg
3. THere is a slight increase in the desity at 5 CCAvg.
4. Lots of outliers visible

Observations on Mortgage

In [24]:
```
histogram_boxplot(data['Mortgage'])
```



Observation:

1. Mostly rightly skewed data.
2. The min and the 25th percentile seems to be equal.
3. Lots of outliers visible

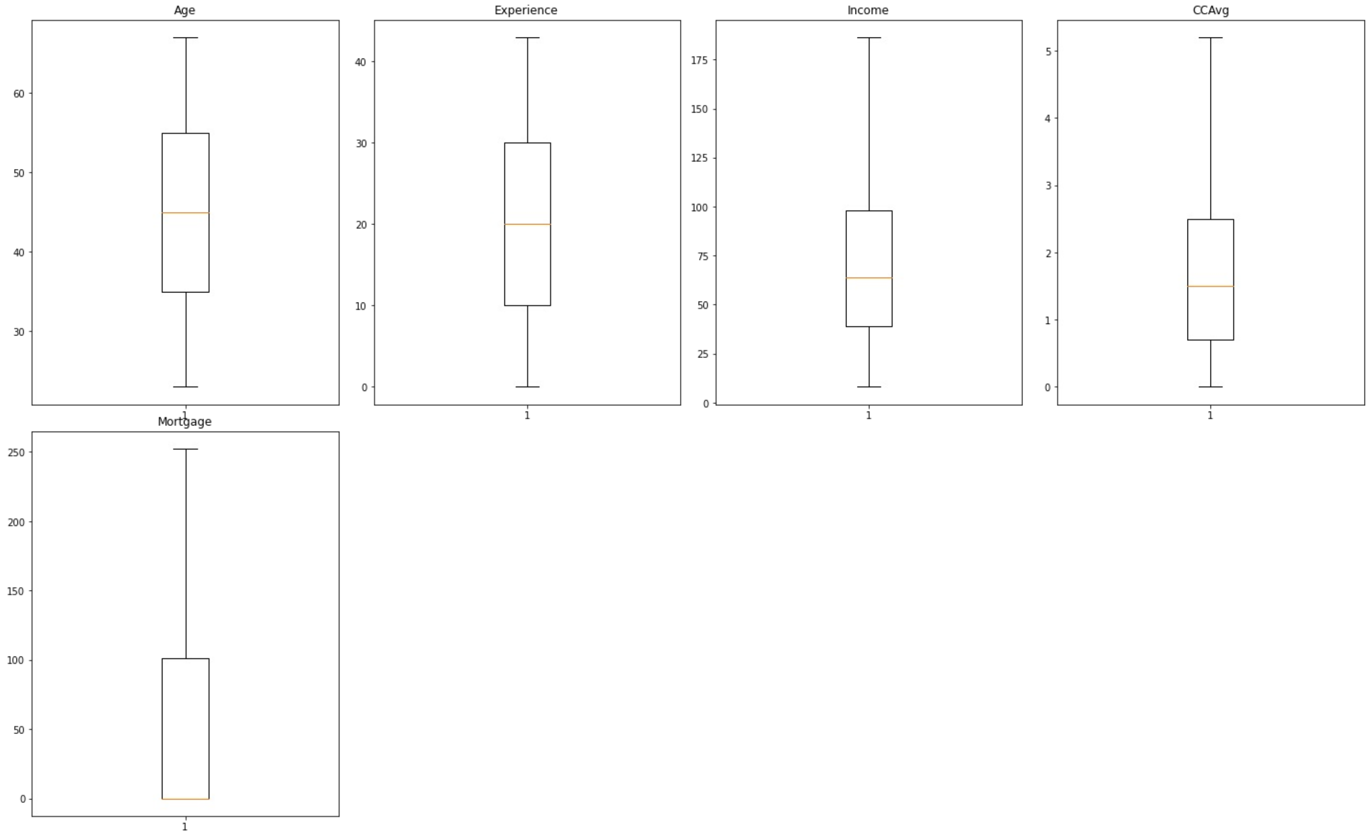
Outlier treatments

In [25]:
```
# Let's treat outliers by flooring and capping
def treat_outliers(df, col):
    """
    treats outliers in a variable
    col: str, name of the numerical variable
    df: dataframe
    col: name of the column
    """
    Q1 = df[col].quantile(0.25)  # 25th quantile
    Q3 = df[col].quantile(0.75)  # 75th quantile
    IQR = Q3 - Q1
    Lower_Whisker = Q1 - 1.5 * IQR
    Upper_Whisker = Q3 + 1.5 * IQR

    # all the values smaller than Lower_Whisker will be assigned the value of Lower_Whisker
    # all the values greater than Upper_Whisker will be assigned the value of Upper_Whisker
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker)

    return df


def treat_outliers_all(df, col_list):
    """
    treat outlier in all numerical variables
```

```
    col_list: list of numerical variables
    df: data frame
    """
    for c in col_list:
        df = treat_outliers(df, c)

    return df
```

In [26]:
```
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()#numberic columns
numerical_col = data.select_dtypes(include=np.number).columns.tolist()#converting numerical cols
data = treat_outliers_all(data, numerical_col) #treating outliers
```

In [27]:
```
plt.figure(figsize=(20, 30))

for i, variable in enumerate(numeric_columns): #boxplots subplots
    plt.subplot(5, 4, i + 1)
    plt.boxplot(data[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



Observation:

1. As it can be seen above, all outliers has been treated and no outliers remain.

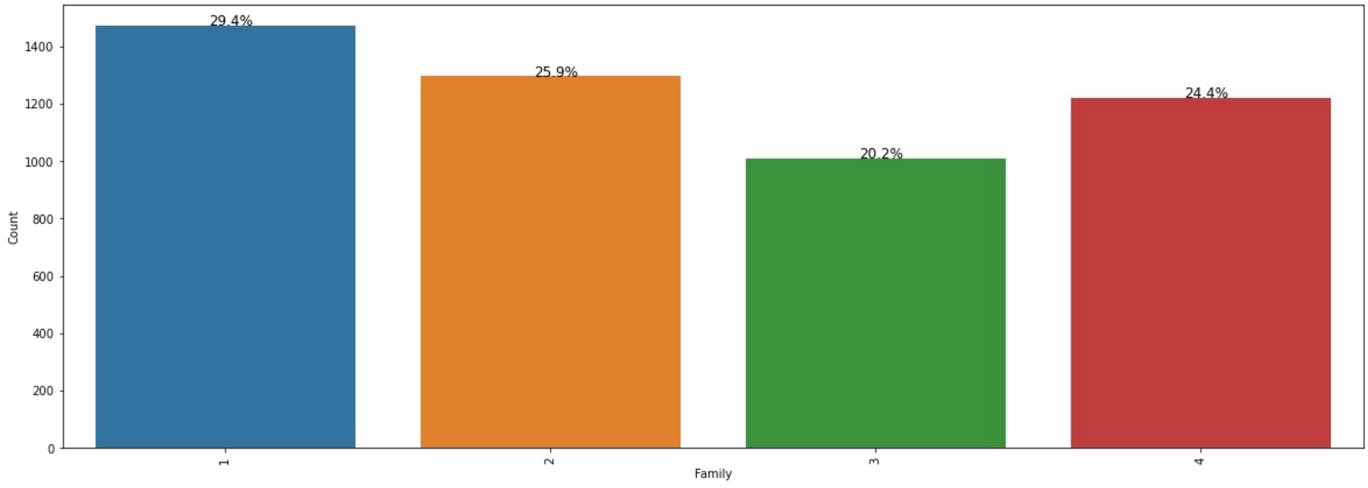
Categorical Variables

In [28]:
```
def bar_perc(plot, feature):
    '''
    plot
    feature: 1-d categorical feature array
    '''
    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total) # percentage of each class of the category
        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
```

Observations on Family

In [29]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data['Family']) #count plot for Name
plt.xlabel('Family')
plt.xticks(rotation=90)
```

```
plt.ylabel('Count')
bar_perc(ax,data['Family'])
```

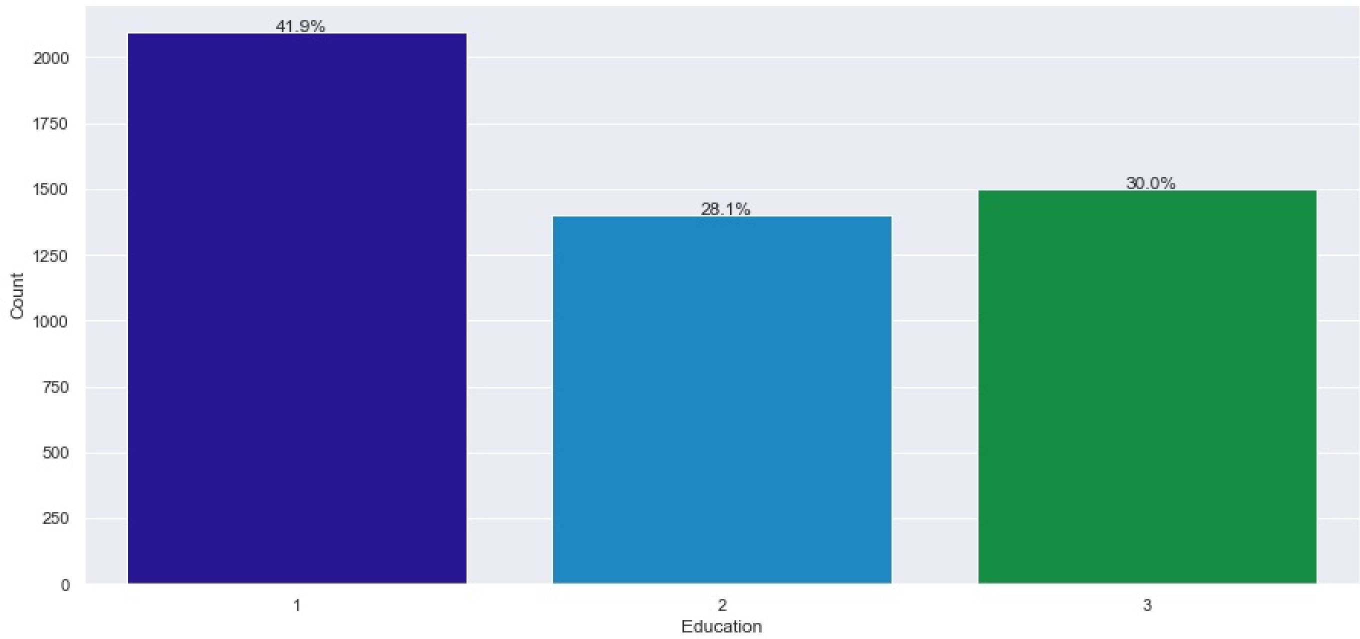


Observations:

1. 1 Member households are most common
2. Least common is 3 member households at 20.2%

Observations on Education

In [98]:
```
plt.figure(figsize=(15,7))
ax = sns.countplot(data['Education']) #count plot for Location
plt.xlabel('Education')
plt.ylabel('Count')
bar_perc(ax,data['Education'])
```

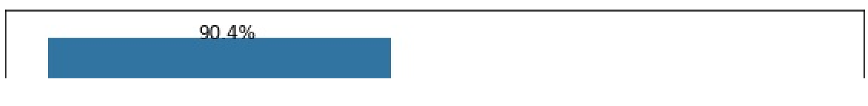


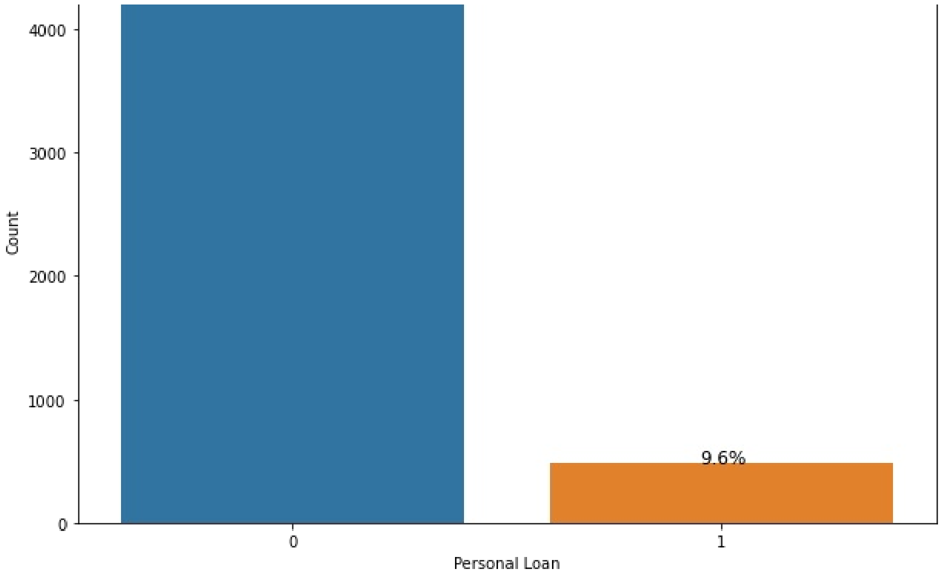
Observation:

1. Undergraduate are most common with 41.9%
2. Least Common is Masters degree with 28.1%

Observation on Personal Loan

In [31]:
```
plt.figure(figsize=(10,7))
ax = sns.countplot(data['Personal_Loan'])
plt.xlabel('Personal Loan')
plt.ylabel('Count')
bar_perc(ax,data['Personal_Loan'])
```



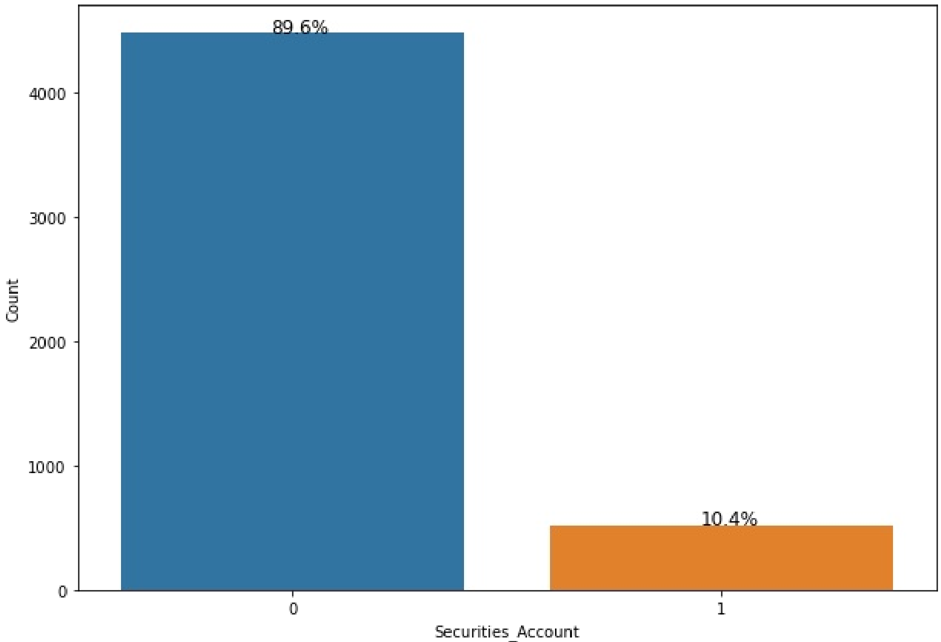


Observation:

1. Most observations 90.4% haven't taken a loan
2. 9.6% has only taken a loan

Observation on Securities Account

In [32]:
```
plt.figure(figsize=(10,7))
ax = sns.countplot(data['Securities_Account'])
plt.xlabel('Securities_Account')
plt.ylabel('Count')
bar_perc(ax,data['Securities_Account'])
```

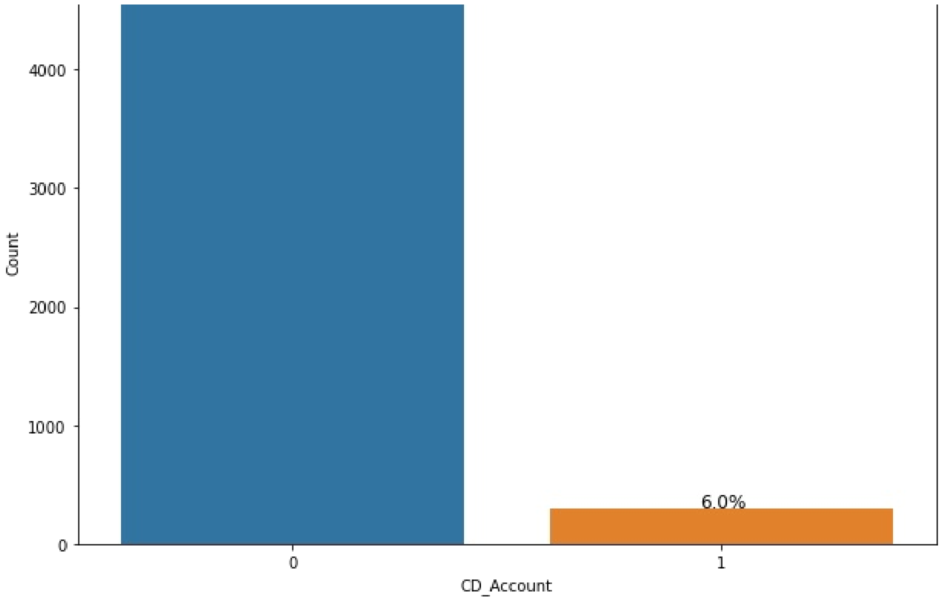


Observation:

1. Most clients do not have a securities account at almost 90%
2. 10% of the cleints have securities account

Observation on CD_Account

In [33]:
```
plt.figure(figsize=(10,7))
ax = sns.countplot(data['CD_Account'])
plt.xlabel('CD_Account')
plt.ylabel('Count')
bar_perc(ax,data['CD_Account'])
```

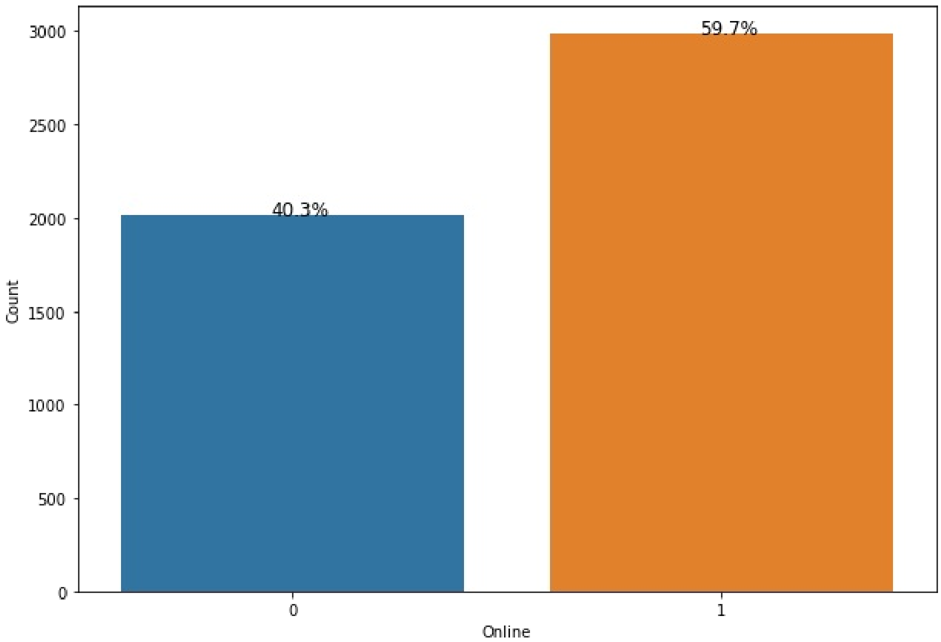


Observation:

1. Most customers do not have a CD_account with 94%
2. 6% have CD_Account

Observation on Online

In [34]:
```
plt.figure(figsize=(10,7))
ax = sns.countplot(data['Online'])
plt.xlabel('Online')
plt.ylabel('Count')
bar_perc(ax,data['Online'])
```

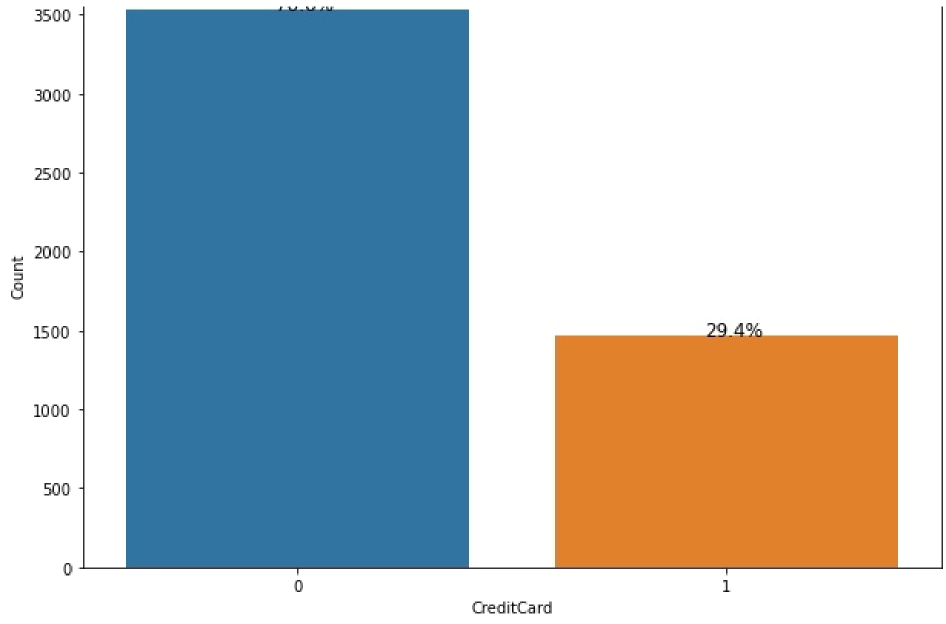


Observation:

1. 60% of customers use online interenet facilities provided by the bank
2. 40% do not use any interenet facilities.

Observation on CreditCard

In [35]:
```
plt.figure(figsize=(10,7))
ax = sns.countplot(data['CreditCard'])
plt.xlabel('CreditCard')
plt.ylabel('Count')
bar_perc(ax,data['CreditCard'])
```

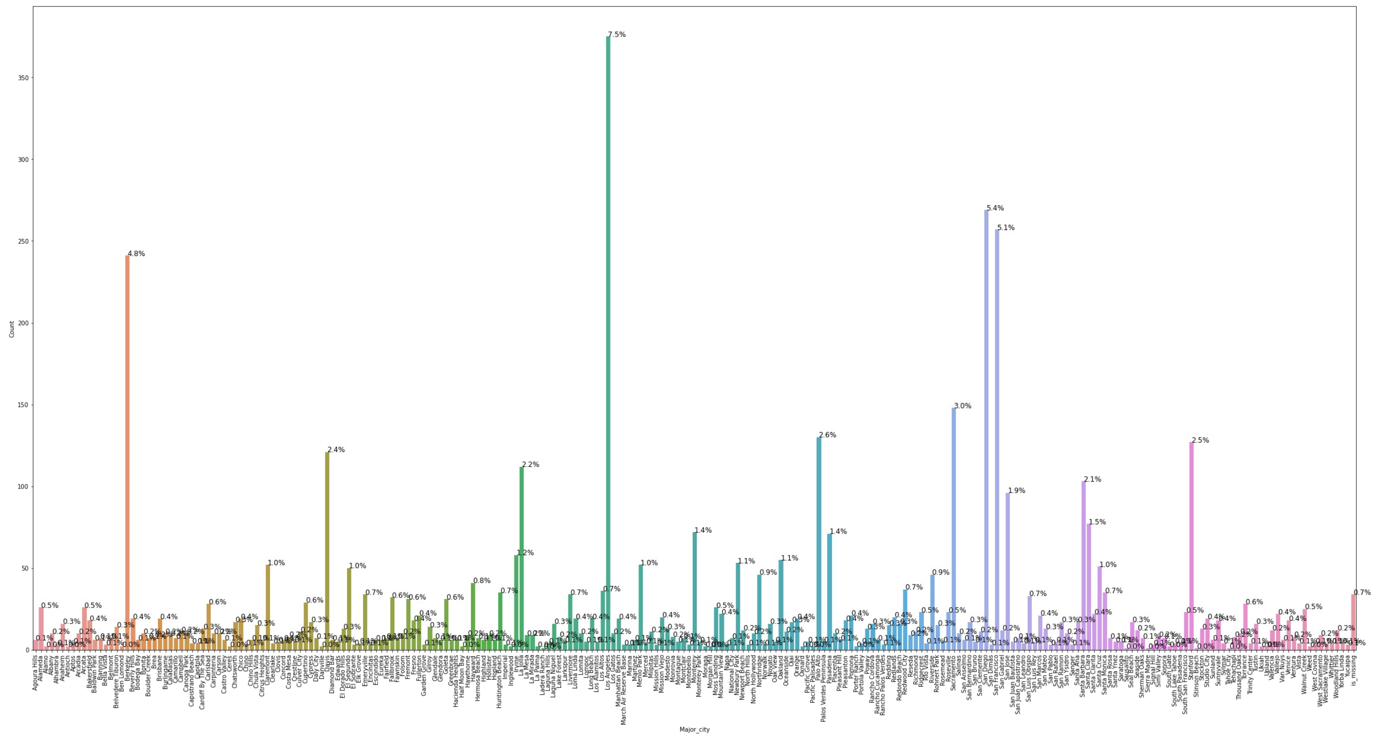


Observation:

1. 70% of customers do not use credit cards
2. 30% of customers do use credit cards

Observation on MajorCity

In [36]:
```
plt.figure(figsize=(40,20))
ax = sns.countplot(data['Major_city'])
plt.xlabel('Major_city')
plt.xticks(rotation= 90)
plt.ylabel('Count')
bar_perc(ax,data['Major_city'])
```



Observation:

1. Highest is berkley with 4.8%

Bivariate Analysis

In [37]:
```
data.corr() #correlation of data
```

Out[37]:

	Age	Experience	Income	CCAvg	Mortgage
Age	1.000000	0.994198	-0.054988	-0.052032	-0.012033
Experience	0.994198	1.000000	-0.046429	-0.050544	-0.010807
Income	-0.054988	-0.046429	1.000000	0.637869	0.135018
CCAvg	-0.052032	-0.050544	0.637869	1.000000	0.068329
Mortgage	-0.012033	-0.010807	0.135018	0.068329	1.000000

In [38]:

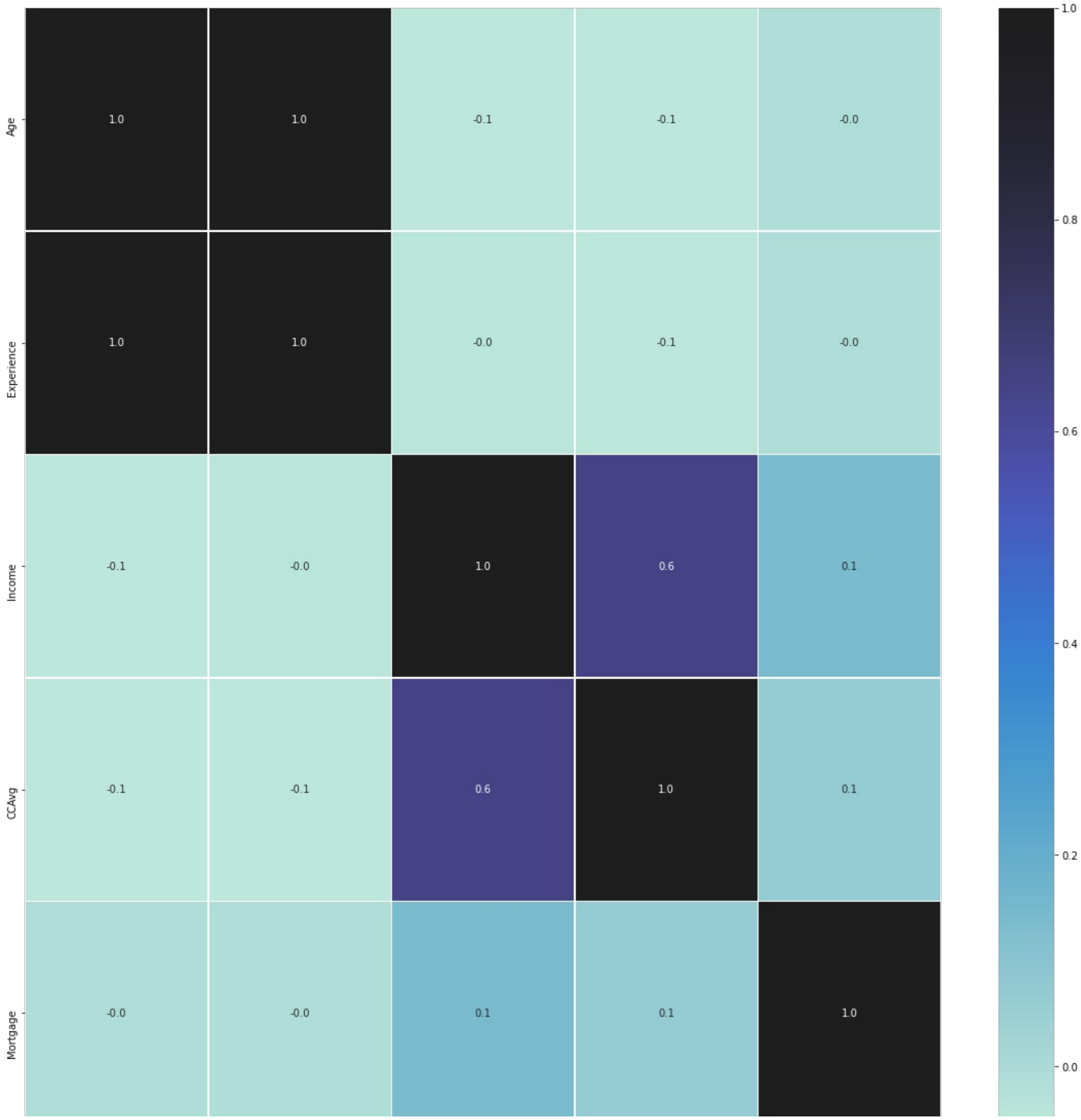
```
data.cov() #covariance of data
```

Out[38]:

	Age	Experience	Income	CCAvg	Mortgage
Age	131.404166	130.383204	-28.759880	-0.866176	-11.448996
Experience	130.383204	130.884673	-24.235388	-0.839746	-10.262119
Income	-28.759880	-24.235388	2081.742966	42.264531	511.343041
CCAvg	-0.866176	-0.839746	42.264531	2.108928	8.236490
Mortgage	-11.448996	-10.262119	511.343041	8.236490	6889.896601

In [39]:

```
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(), annot=True, linewidths=.5, fmt= '.1f', center = 1 )  # heatmap
plt.show()
```

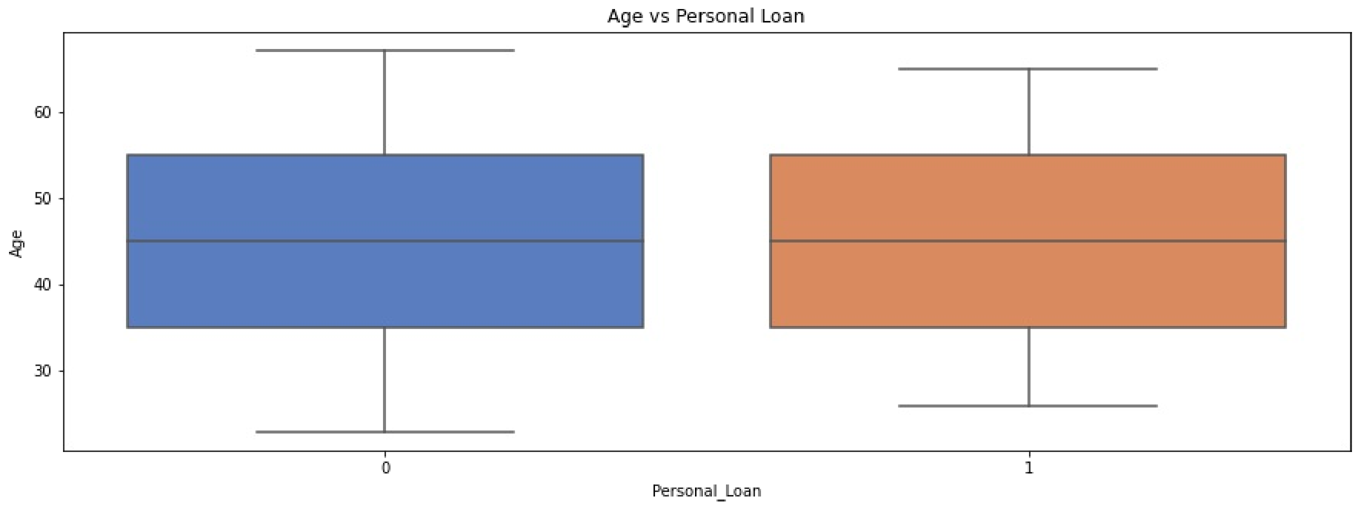


Observation:

1. Near perfect correlation between Age and Experience
2. Income and CCAvg have high correaltion between the variables.
3. All other variables have low correlation between each other

Numerical Vs Categorical

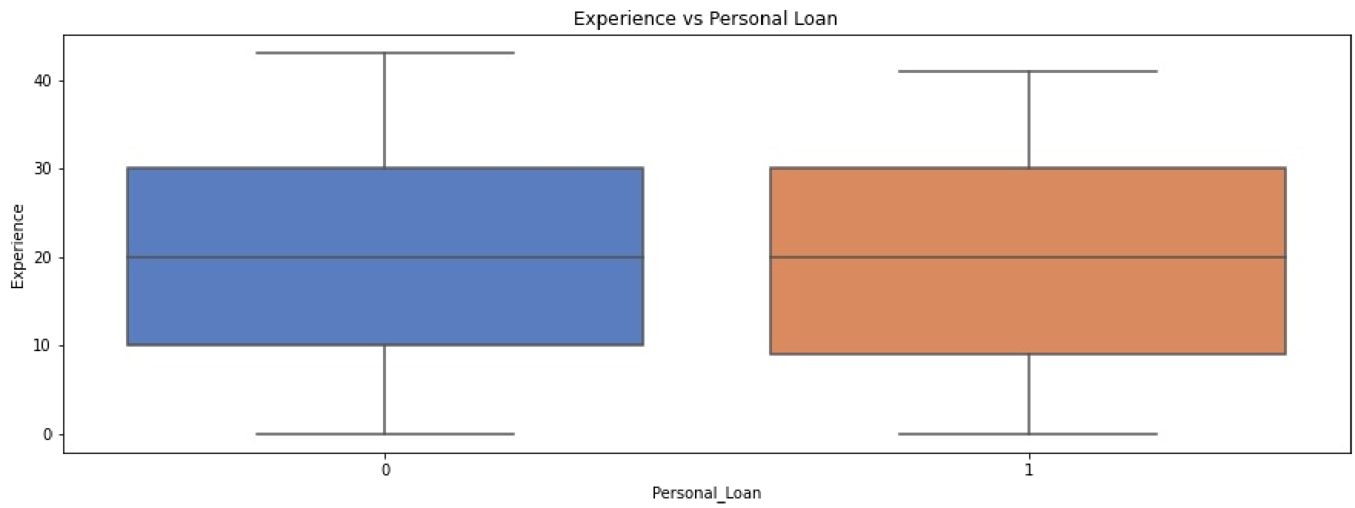
In [42]:
```
plt.figure(figsize=(15,5))  # setting the figure size
plt.title('Age vs Personal Loan')
ax = sns.boxplot(x='Personal_Loan', y='Age', data=data, palette='muted')#barplot of location vs price
```



Observations:

1. Personal Loan appear to be equal in all age cateogries
2. Age does not appear to be a strong indicator of when clients take personal loan

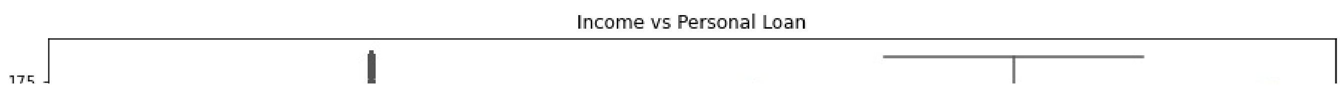
In [43]:
```
plt.figure(figsize=(15,5))  # setting the figure size
plt.title('Experience vs Personal Loan')
ax = sns.boxplot(x='Personal_Loan', y='Experience', data=data, palette='muted')
```

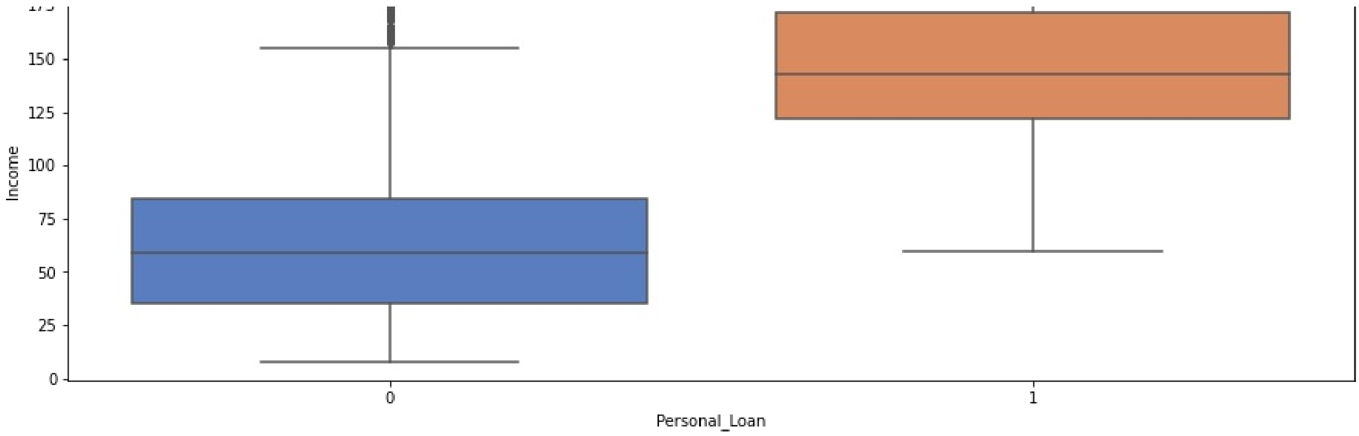


Observations:

1. There does not appear to be much difference between personal loan and expereince

In [45]:
```
plt.figure(figsize=(15,5))  # setting the figure size
plt.title('Income vs Personal Loan')
ax = sns.boxplot(x='Personal_Loan', y='Income', data=data, palette='muted')
```

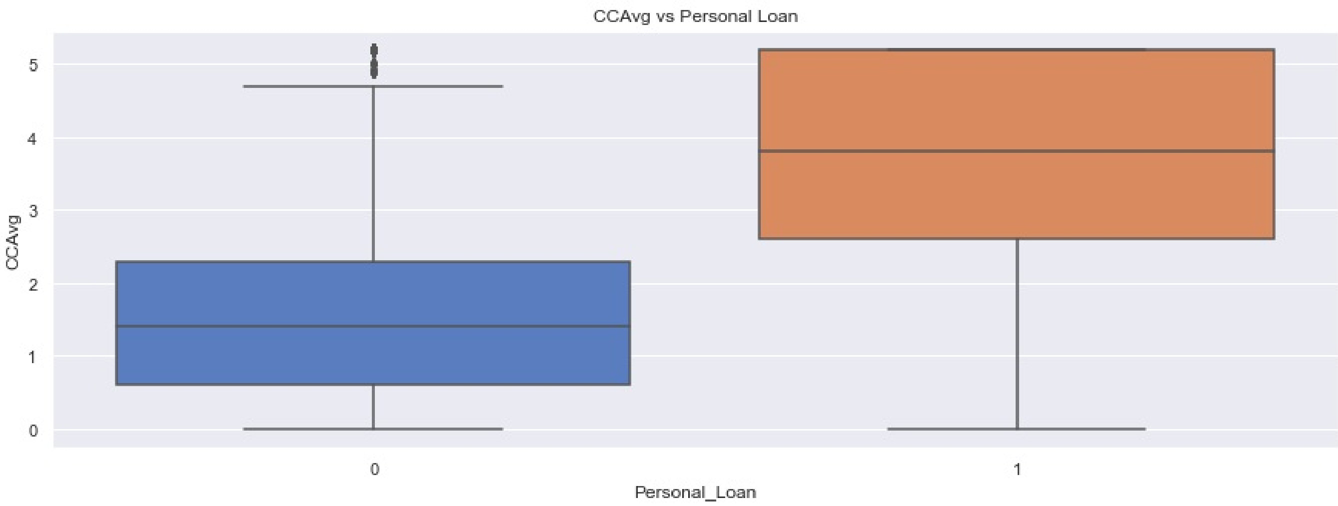




Observation:

1. CLients with higher income on average do take personal loan
2. The 75th percentile of clients that do not take loan is about the average of clients that take loan with higher income

In [96]:
```
plt.figure(figsize=(15,5))  # setting the figure size
plt.title('CCAvg vs Personal Loan')
ax = sns.boxplot(x='Personal_Loan', y='CCAvg', data=data, palette='muted')
```



Observation:

1. On average clinets who do not take personal loan have lower CCAvg, while clients who do take personal loan have higher CCAvg.
2. The 75th percentile of clients who take personal loan have higher CCAvg than clients who do not take personal loan

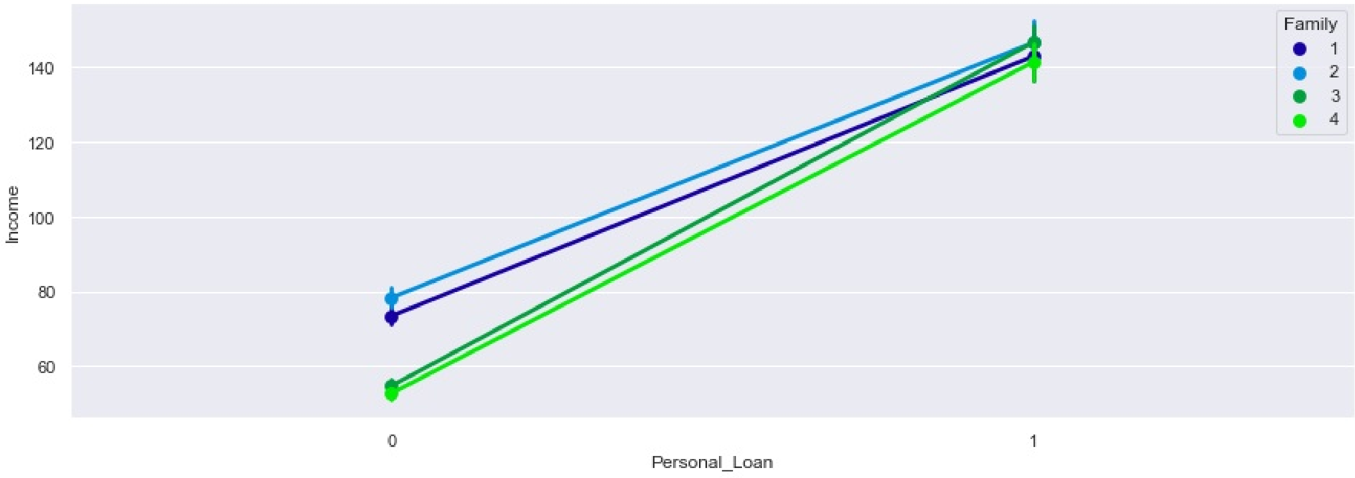
In [97]:
```
plt.figure(figsize=(15,5))  # setting the figure size
plt.title('Mortgage vs Personal Loan')
ax = sns.boxplot(x='Personal_Loan', y='Mortgage', data=data, palette='muted')
```

Observation:

1. There appear to be some outliers in Mortgage with clients who do not take personal loan
2. There are more clients who take Personal Loan who have higher mortgage.

Observation on Income, Family and Personal Loan

In [67]:
```
plt.figure(figsize=(15,5))

sns.pointplot(x="Personal_Loan", y='Income', hue = 'Family',  data=data)
plt.show()
```

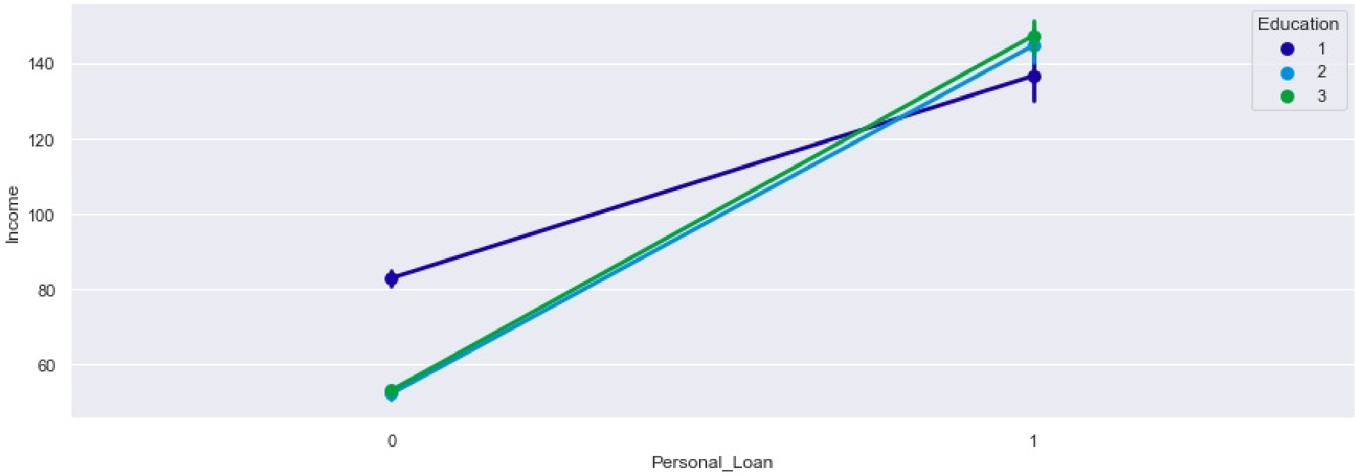


Observation:

1. Family with higher income take personal loans as compared to family wiht lower income.

Observation on Education, Personal Loan and Income

In [66]:
```
plt.figure(figsize=(15,5))

sns.pointplot(x="Personal_Loan", y='Income', hue = 'Education',  data=data)
plt.show()
```



Observation:

1. Personal loan are opted by higher income clients regradeless of what education level they have.
2. Level 3 education level are more likely to take Personal loans as comared to level 2 and level 1 education clients

Observation on Securities Account, Personal Loan and Income

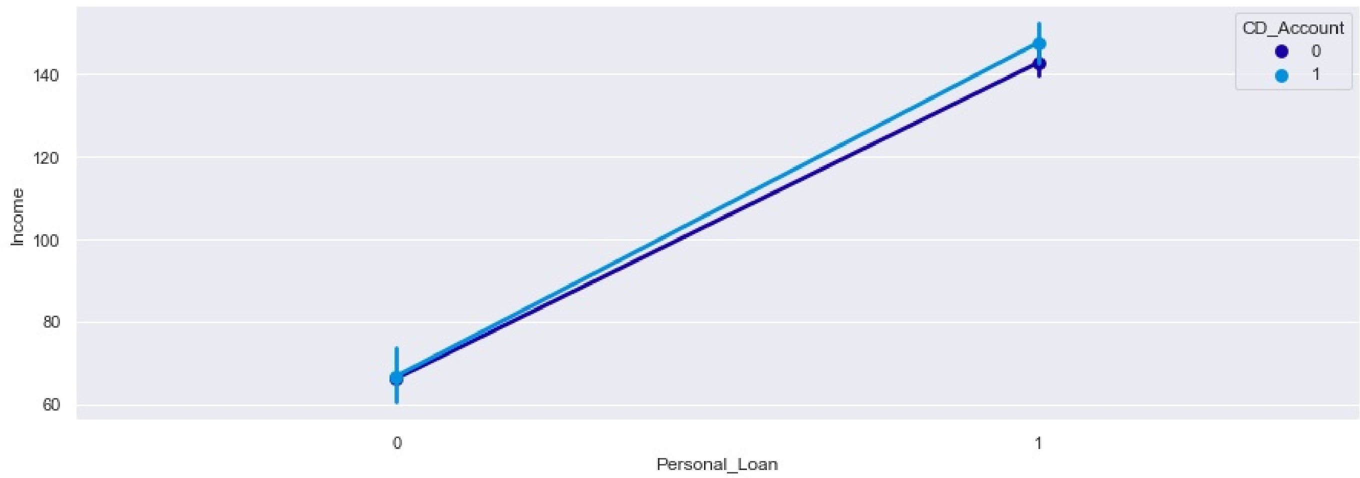
In [68]:
```
plt.figure(figsize=(15,5))

sns.pointplot(x="Personal_Loan", y='Income', hue = 'Securities_Account',  data=data)
plt.show()
```

Observation:

1. Less Clients take personal loan with securities account.
2. Higher income clients are less likely to have a securities account and a personal loan

Observation on CD_Account, Personal Loan and Income

In [69]:
```
plt.figure(figsize=(15,5))

sns.pointplot(x="Personal_Loan", y='Income', hue = 'CD_Account',  data=data)
plt.show()
```

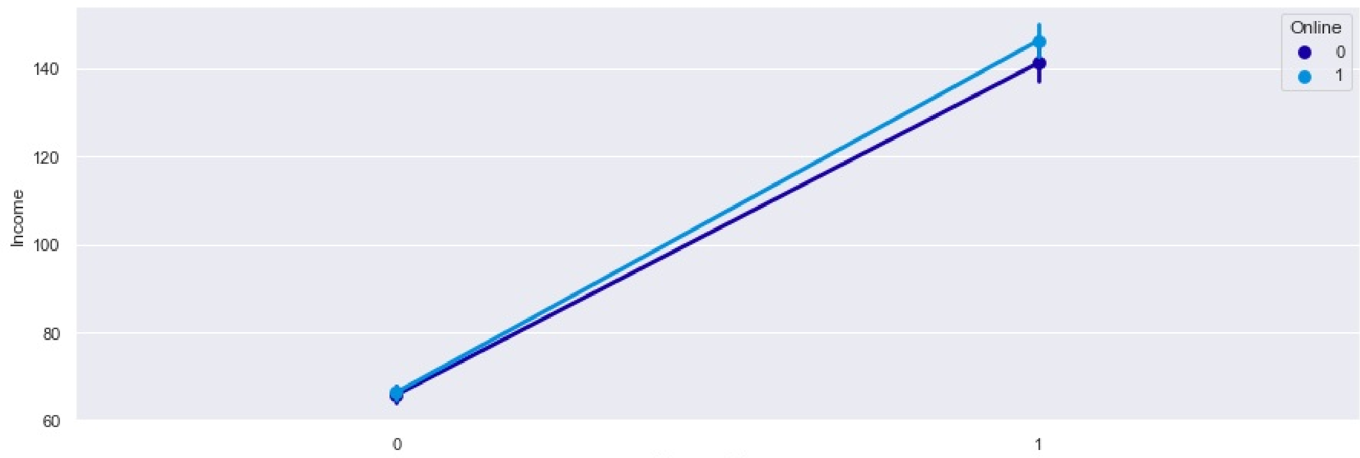


Observation:

1. Clients with higher income do own a CD_Account when taking a personal Loan

Observation on Online, Personal Loan and Income

In [80]:
```
plt.figure(figsize=(15,5))

sns.pointplot(x="Personal_Loan", y='Income', hue = 'Online',  data=data)
plt.show()
```



Personal_Loan

Observation:

1. Clients with higher income do own use the online faciltiies do take a personal Loan

Observation on CreditCard, Personal Loan and Income

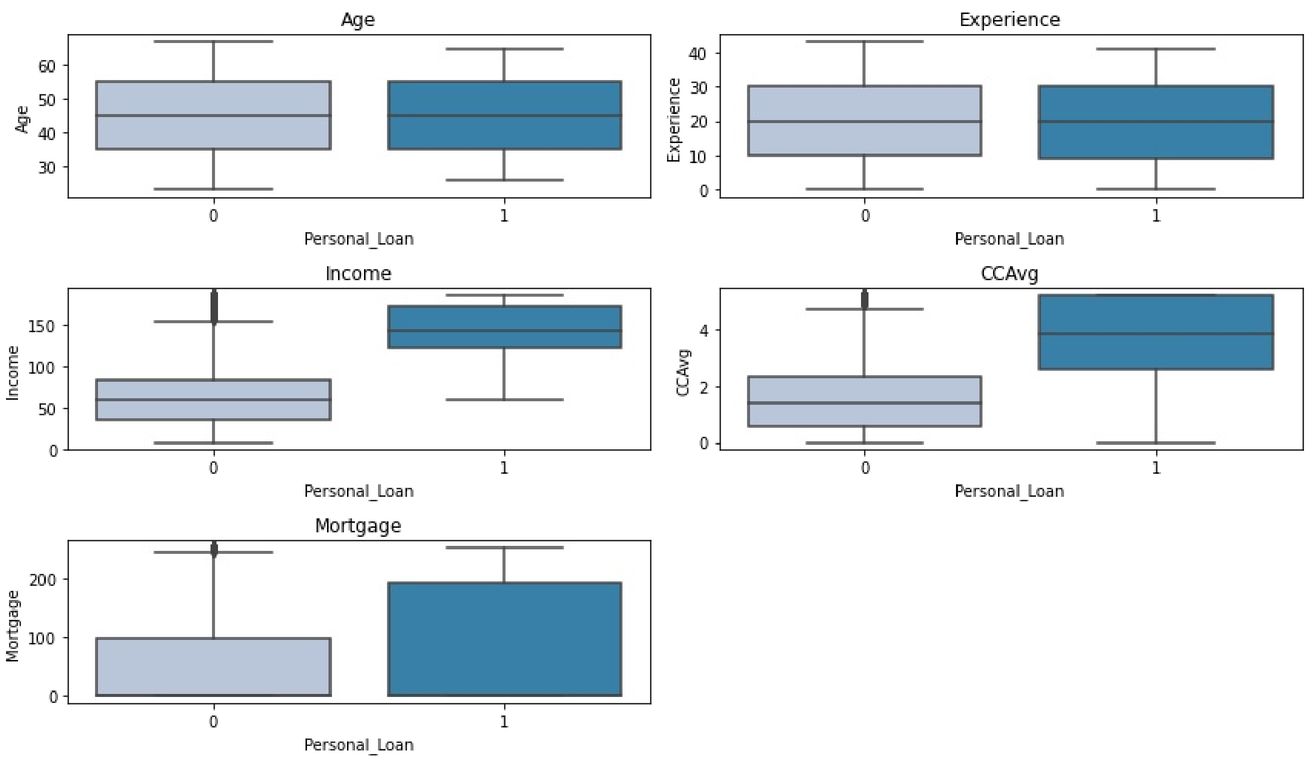
In [71]:
```
plt.figure(figsize=(15,5))

sns.pointplot(x="Personal_Loan", y='Income', hue = 'CreditCard',  data=data)
plt.show()
```

Observation:

1. There isn't much different in clients with higher or lower income when it comes to owning a creditcard and taking personal loans

Barplots for all non-categorical variables against Personal_Loan

In [60]:
```
cols = data[['Age','Experience','Income','CCAvg','Mortgage']].columns.tolist()
plt.figure(figsize=(12,7))

for i, variable in enumerate(cols):
                     plt.subplot(3,2,i+1)
                     sns.boxplot(data["Personal_Loan"],data[variable],palette="PuBu")
                     plt.tight_layout()
                     plt.title(variable)
plt.show()
```



Observation:

1. Income, CCAvg and Mortgage, all have impact on clients that take Personal Loan.
2. The variables Income, CCAvg and MOrtgage have higher values on average when a client decides to take a personal loan
3. Age and experience do not seem to have much impact on Personal Loan.

Data Preparation for Modeling

In [138…

```
import scipy.stats as stats # this library contains a large number of probability distributions as well as a grov
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant

# To build sklearn model
from sklearn.linear_model import LogisticRegression

# To get diferent metric scores
from sklearn import metrics
from sklearn.metrics import f1_score,accuracy_score, recall_score, precision_score, roc_auc_score, roc_curve, cor
```

In [139…

```
def split(*kwargs):
    '''
    Function to split data into X and Y then one hot encode the X variable.
    Returns training and test sets
    *kwargs : Variable to remove from the dataset before splitting into X and Y
    '''
    X = data.drop([*kwargs], axis=1)
    Y = data['Personal_Loan']

    X = pd.get_dummies(X,columns=["Family", 'Education', 'Securities_Account', 'CD_Account', 'Online', 'CreditCar
    X = add_constant(X)

    #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 = 1)
    return X_train,X_test, y_train, y_test
```

In [140…

```
X_train,X_test, y_train, y_test = split('Personal_Loan', 'Major_city')
print(X_test.head())
```

```
      const  Age  Experience  Income  CCAvg  Mortgage  Family_2  Family_3  \
ID
2765    1.0   31           5    84.0    2.9     105.0         0         0
4768    1.0   35           9    45.0    0.9     101.0         0         1
3815    1.0   34           9    35.0    1.3       0.0         0         1
3500    1.0   49          23   114.0    0.3     252.5         0         0
2736    1.0   36          12    70.0    2.6     165.0         0         1

      Family_4  Education_2  Education_3  Securities_Account_1  CD_Account_1  \
ID
2765         0            0            1                     0             0
4768         0            0            0                     1             0
3815         0            0            0                     0             0
3500         0            0            0                     0             0
2736         0            1            0                     0             0

      Online_1  CreditCard_1
ID
2765         0             1
4768         0             0
3815         0             0
3500         1             0
2736         1             0
```

Observation:

1. Major_city was excluded from any of the analysis as caused LinAlgError, as the correlation between the Major_city was highly correlated with other variables causing problems

In [141…

```
def get_metrics_score1(model,train,test,train_y,test_y,threshold=0.5,flag=True,roc=False):
    '''
    Function to calculate different metric scores of the model - Accuracy, Recall, Precision, and F1 score
    model: classifier to predict values of X
    train, test: Independent features
    train_y,test_y: Dependent variable
    threshold: thresold for classifiying the observation as 1
```

```
    flag: If the flag is set to True then only the print statements showing different will be displayed. The defa
    roc: If the roc is set to True then only roc score will be displayed. The default value is set to False.
    '''
    # defining an empty list to store train and test results

    score_list=[]

    pred_train = (model.predict(train)>threshold)
    pred_test = (model.predict(test)>threshold)

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

    train_acc = accuracy_score(pred_train,train_y)
    test_acc = accuracy_score(pred_test,test_y)

    train_recall = recall_score(train_y,pred_train)
    test_recall = recall_score(test_y,pred_test)

    train_precision = precision_score(train_y,pred_train)
    test_precision = precision_score(test_y,pred_test)

    train_f1 = f1_score(train_y,pred_train)
    test_f1 = f1_score(test_y,pred_test)


    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test_precision,train_f1,test_f


    if flag == True:
        print("Accuracy on training set : ",accuracy_score(pred_train,train_y))
        print("Accuracy on test set : ",accuracy_score(pred_test,test_y))
        print("Recall on training set : ",recall_score(train_y,pred_train))
        print("Recall on test set : ",recall_score(test_y,pred_test))
        print("Precision on training set : ",precision_score(train_y,pred_train))
        print("Precision on test set : ",precision_score(test_y,pred_test))
        print("F1 on training set : ",f1_score(train_y,pred_train))
        print("F1 on test set : ",f1_score(test_y,pred_test))

    if roc == True:
        print("ROC-AUC Score on training set : ",roc_auc_score(train_y,pred_train))
        print("ROC-AUC Score on test set : ",roc_auc_score(test_y,pred_test))

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

In [142…

```
def get_metrics_score2(model,train,test,train_y,test_y,flag=True,roc=False):
    '''
    Function to calculate different metric scores of the model - Accuracy, Recall, Precision, and F1 score
    model: classifier to predict values of X
    train, test: Independent features
    train_y,test_y: Dependent variable
    flag: If the flag is set to True then only the print statements shwoing different will be displayed. The defa
    roc: If the roc is set to True then only roc score will be displayed. The default value is set to False.
    '''
    # defining an empty list to store train and test results

    score_list=[]

    pred_train = model.predict(train)
    pred_test = model.predict(test)

    train_acc = accuracy_score(pred_train,train_y)
    test_acc = accuracy_score(pred_test,test_y)

    train_recall = recall_score(train_y,pred_train)
    test_recall = recall_score(test_y,pred_test)

    train_precision = precision_score(train_y,pred_train)
    test_precision = precision_score(test_y,pred_test)

    train_f1 = f1_score(train_y,pred_train)
    test_f1 = f1_score(test_y,pred_test)


    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test_precision,train_f1,test_f

     # If the flag is set to True then only the following print statements will be dispayed. The default value is
    if flag == True:
        print("Accuracy on training set : ",accuracy_score(pred_train,train_y))
        print("Accuracy on test set : ",accuracy_score(pred_test,test_y))
        print("Recall on training set : ",recall_score(train_y,pred_train))
        print("Recall on test set : ",recall_score(test_y,pred_test))
        print("Precision on training set : ",precision_score(train_y,pred_train))
        print("Precision on test set : ",precision_score(test_y,pred_test))
        print("F1 on training set : ",f1_score(train_y,pred_train))
        print("F1 on test set : ",f1_score(test_y,pred_test))

    if roc == True:
```

```
        print("ROC-AUC Score on training set : ",roc_auc_score(train_y,pred_train))
        print("ROC-AUC Score on test set : ",roc_auc_score(test_y,pred_test))

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

In [143…

```
def make_confusion_matrix(model,test_X,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    test_X: test set
    y_actual : ground truth

    '''
    y_predict = model.predict(test_X)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[1,0])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - >50K","Actual - <=50K"]],
                  columns = [i for i in ['Predicted - >50K','Predicted - <=50k']])
    group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [144…

```
lr = LogisticRegression(solver='newton-cg',random_state=1,fit_intercept=False)
model  = lr.fit(X_train,y_train)

# Let's check model performances for this model
scores_LR = get_metrics_score2(model,X_train,X_test,y_train,y_test)
```

```
Accuracy on training set :  0.964
Accuracy on test set :  0.9546666666666667
Recall on training set :  0.7099697885196374
Recall on test set :  0.6174496644295302
Precision on training set :  0.8867924528301887
Precision on test set :  0.8932038834951457
F1 on training set :  0.7885906040268457
F1 on test set :  0.7301587301587301
```

In [145…

```
logit = sm.Logit(y_train, X_train)
lg = logit.fit(warn_convergence =False)

# Let's check model performances for this model
scores_LR = get_metrics_score1(lg,X_train,X_test,y_train,y_test)
```

```
Optimization terminated successfully.
         Current function value: 0.098908
         Iterations 10
Accuracy on training set :  0.9665714285714285
Accuracy on test set :  0.9613333333333334
Recall on training set :  0.7371601208459214
Recall on test set :  0.6644295302013423
Precision on training set :  0.8905109489051095
Precision on test set :  0.9252336448598131
F1 on training set :  0.8066115702479338
F1 on test set :  0.7734375000000001
```

In [146…

```
lg.summary()
```

Out[146…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3500
Model:	Logit	**Df Residuals:**	3485
Method:	MLE	**Df Model:**	14
Date:	Fri, 18 Jun 2021	**Pseudo R-squ.:**	0.6840
Time:	02:16:39	**Log-Likelihood:**	-346.18
converged:	True	**LL-Null:**	-1095.5
Covariance Type:	nonrobust	**LLR p-value:**	9.650e-312

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-14.4607	2.312	-6.254	0.000	-18.992	-9.929
Age	-0.0162	0.083	-0.194	0.846	-0.180	0.147
Experience	0.0245	0.083	0.294	0.769	-0.139	0.188

Income	0.0662	0.004	15.694	0.000	0.058	0.074
CCAvg	0.5316	0.080	6.645	0.000	0.375	0.688
Mortgage	0.0013	0.001	1.296	0.195	-0.001	0.003
Family_2	0.0655	0.296	0.221	0.825	-0.515	0.646
Family_3	2.7453	0.337	8.152	0.000	2.085	3.405
Family_4	1.7806	0.325	5.481	0.000	1.144	2.417
Education_2	4.2435	0.364	11.648	0.000	3.529	4.958
Education_3	4.5111	0.367	12.308	0.000	3.793	5.229
Securities_Account_1	-1.0191	0.423	-2.407	0.016	-1.849	-0.189
CD_Account_1	3.6459	0.458	7.964	0.000	2.749	4.543
Online_1	-0.5915	0.214	-2.759	0.006	-1.012	-0.171
CreditCard_1	-0.9789	0.280	-3.498	0.000	-1.527	-0.430

Possibly complete quasi-separation: A fraction 0.19 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

But first we will have to remove multicollinearity from the data to get reliable coefficients and p-values.

1. There are different ways of detecting (or testing) multi-collinearity, one such way is the Variation Inflation Factor.
2. Additional Information on VIF Variance Inflation factor: Variance inflation factors measure the inflation in the variances of the regression coefficients estimates due to collinearities that exist among the predictors. It is a measure of how much the variance of the estimated regression coefficient βk is “inflated”by the existence of correlation among the predictor variables in the model.
3. General Rule of thumb: If VIF is 1 then there is no correlation among the kth predictor and the remaining predictor variables, and hence the variance of βk is not inflated at all. Whereas if VIF exceeds 5, we say there is moderate VIF and if it is 10 or exceeding 10, it shows signs of high multi-collinearity. But the purpose of the analysis should dictate which threshold to use

Multicollinearity

In [147…
```
# changing datatype of colums to numeric for checking vif
X_train_num = X_train.astype(float).copy()
```

In [148…
```
vif_series1 = pd.Series([variance_inflation_factor(X_train_num.values,i) for i in range(X_train_num.shape[1])],ir
print('Series before feature selection: \n\n{}\n'.format(vif_series1))
```

```
Series before feature selection:

const                   466.511039
Age                      95.850764
Experience               95.756287
Income                    1.831372
CCAvg                     1.707041
Mortgage                  1.020406
Family_2                  1.401950
Family_3                  1.384443
Family_4                  1.426069
Education_2               1.301434
Education_3               1.340159
Securities_Account_1      1.147680
CD_Account_1              1.363271
Online_1                  1.040982
CreditCard_1              1.111249
dtype: float64
```

Observation:

1. The high correlation between these variables has been highlighted in the VIF values as well.
2. Only Age and Experience show high multicollinearity

In [149…
```
X_train_num1 = X_train_num.drop('Age',axis=1)
vif_series2 = pd.Series([variance_inflation_factor(X_train_num1.values,i) for i in range(X_train_num1.shape[1])],
print('Series before feature selection: \n\n{}\n'.format(vif_series2))
```

```
Series before feature selection:
```

```
const                    15.205511
Experience                1.012893
Income                    1.825870
CCAvg                     1.701098
Mortgage                  1.020357
Family_2                  1.401880
Family_3                  1.380378
Family_4                  1.426022
Education_2               1.288092
Education_3               1.257142
Securities_Account_1      1.147226
CD_Account_1              1.362293
Online_1                  1.040816
CreditCard_1              1.111230
dtype: float64
```

Observation:

1. Dropping Age has reduced the high VIF values in Experience.

Let's create a model with all the features except Age

In [150…
```
X_train.drop(['Age'],axis=1,inplace=True)
X_test.drop(['Age'],axis=1,inplace=True)
```

In [151…
```
logit1 = sm.Logit(y_train, X_train.astype(float))
lg1 = logit1.fit(warn_convergence =False)

# Let's check model performances for this model
scores_LR = get_metrics_score1(lg1,X_train,X_test,y_train,y_test)
```

```
Optimization terminated successfully.
         Current function value: 0.098913
         Iterations 10
Accuracy on training set :  0.9662857142857143
Accuracy on test set :  0.9613333333333334
Recall on training set :  0.7311178247734139
Recall on test set :  0.6644295302013423
Precision on training set :  0.8929889298892989
Precision on test set :  0.9252336448598131
F1 on training set :  0.8039867109634551
F1 on test set :  0.7734375000000001
```

Droping Experience

In [153…
```
X_train2,X_test2,y_train,y_test = split('Personal_Loan','Experience')
```

In [154…
```
X_train2.drop(['Major_city'],axis=1,inplace=True)
X_test2.drop(['Major_city'],axis=1,inplace=True)
```

In [155…
```
X_train_num3 = X_train2.astype(float).copy()
vif_series4 = pd.Series([variance_inflation_factor(X_train_num3.values,i) for i in range(X_train_num3.shape[1])],
print('Series before feature selection: \n\n{}\n'.format(vif_series4))
```

```
Series before feature selection:

const                    28.464443
Age                       1.013893
Income                    1.826586
CCAvg                     1.700369
Mortgage                  1.020371
Family_2                  1.401877
Family_3                  1.380454
Family_4                  1.425860
Education_2               1.287354
Education_3               1.257770
Securities_Account_1      1.147170
CD_Account_1              1.362048
Online_1                  1.040829
CreditCard_1              1.111222
dtype: float64
```

In [156…
```
logit3 = sm.Logit(y_train, X_train2.astype(float))
```

```
In [156…   logit3 = sm.Logit(y_train, X_train2.astype(float))
lg3 = logit3.fit(warn_convergence =False)

# Let's check model performances for this model
scores_LR = get_metrics_score1(lg3,X_train2,X_test2,y_train,y_test)
```

```
Optimization terminated successfully.
         Current function value: 0.098920
         Iterations 10
Accuracy on training set :  0.9665714285714285
Accuracy on test set :  0.9613333333333334
Recall on training set :  0.7311178247734139
Recall on test set :  0.6644295302013423
Precision on training set :  0.8962962962962963
Precision on test set :  0.9252336448598131
F1 on training set :  0.805324459234609
F1 on test set :  0.7734375000000001
```

Observation:

1. Changing Age and Education, a change in the model performance is not observed, so either model can be used for further testing.

Summary of final model

In [159…
```
lg3.summary()
```

Out[159…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3500
Model:	Logit	**Df Residuals:**	3486
Method:	MLE	**Df Model:**	13
Date:	Fri, 18 Jun 2021	**Pseudo R-squ.:**	0.6839
Time:	02:18:24	**Log-Likelihood:**	-346.22
converged:	True	**LL-Null:**	-1095.5
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-15.0822	0.945	-15.959	0.000	-16.935	-13.230
Age	0.0082	0.009	0.923	0.356	-0.009	0.026
Income	0.0663	0.004	15.802	0.000	0.058	0.075
CCAvg	0.5307	0.080	6.647	0.000	0.374	0.687
Mortgage	0.0013	0.001	1.288	0.198	-0.001	0.003
Family_2	0.0655	0.296	0.221	0.825	-0.515	0.647
Family_3	2.7451	0.337	8.150	0.000	2.085	3.405
Family_4	1.7812	0.325	5.482	0.000	1.144	2.418
Education_2	4.2395	0.364	11.642	0.000	3.526	4.953
Education_3	4.4977	0.364	12.370	0.000	3.785	5.210
Securities_Account_1	-1.0168	0.424	-2.399	0.016	-1.848	-0.186
CD_Account_1	3.6483	0.458	7.967	0.000	2.751	4.546
Online_1	-0.5905	0.214	-2.756	0.006	-1.010	-0.171
CreditCard_1	-0.9776	0.280	-3.495	0.000	-1.526	-0.429

Possibly complete quasi-separation: A fraction 0.19 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 [163…
```
X_train4 = X_train2.drop(['Age','Mortgage'], axis = 1)
X_test4 = X_test2.drop(['Age', 'Mortgage'], axis = 1)

logit4 = sm.Logit(y_train, X_train4.astype(float))
lg4 = logit4.fit(warn_convergence =False)

print(lg4.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.099258
```

```
Iterations 10
                           Logit Regression Results
==============================================================================
Dep. Variable:           Personal_Loan   No. Observations:                 3500
Model:                           Logit   Df Residuals:                     3488
Method:                            MLE   Df Model:                           11
Date:                 Fri, 18 Jun 2021   Pseudo R-squ.:                  0.6829
Time:                         02:24:37   Log-Likelihood:                -347.40
converged:                        True   LL-Null:                       -1095.5
Covariance Type:             nonrobust   LLR p-value:                     0.000
=======================================================================================
                          coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                 -14.6094      0.817    -17.882      0.000     -16.211     -13.008
Income                  0.0664      0.004     15.869      0.000       0.058       0.075
CCAvg                   0.5182      0.079      6.531      0.000       0.363       0.674
Family_2                0.0830      0.296      0.280      0.779      -0.497       0.663
Family_3                2.7636      0.337      8.193      0.000       2.102       3.425
Family_4                1.7845      0.325      5.483      0.000       1.147       2.422
Education_2             4.2119      0.363     11.613      0.000       3.501       4.923
Education_3             4.4714      0.362     12.359      0.000       3.762       5.180
Securities_Account_1   -1.0098      0.425     -2.375      0.018      -1.843      -0.176
CD_Account_1            3.6761      0.458      8.023      0.000       2.778       4.574
Online_1               -0.5891      0.214     -2.757      0.006      -1.008      -0.170
CreditCard_1           -0.9725      0.278     -3.497      0.000      -1.518      -0.427
=======================================================================================

Possibly complete quasi-separation: A fraction 0.19 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:

1. Mortgage and Age that had high P values have been dropped.
2. Only high P-value is Family_2 which can be ingnored as not all the categorical values of Family are high.

Metrics of final model 'lg4'

In [164…

```
scores_LR = get_metrics_score1(lg4,X_train4,X_test4,y_train,y_test)
```

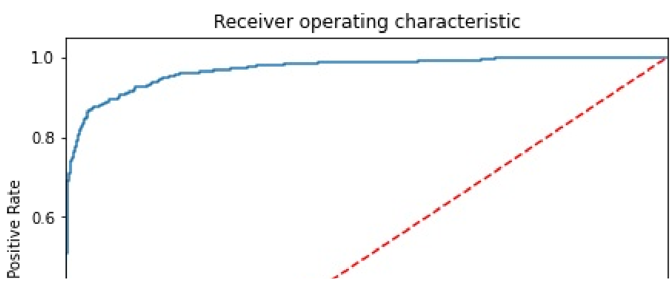
```
Accuracy on training set :  0.9691428571428572
Accuracy on test set :  0.9586666666666667
Recall on training set :  0.7371601208459214
Recall on test set :  0.6442953020134228
Precision on training set :  0.9207547169811321
Precision on test set :  0.9142857142857143
F1 on training set :  0.8187919463087249
F1 on test set :  0.7559055118110237
```

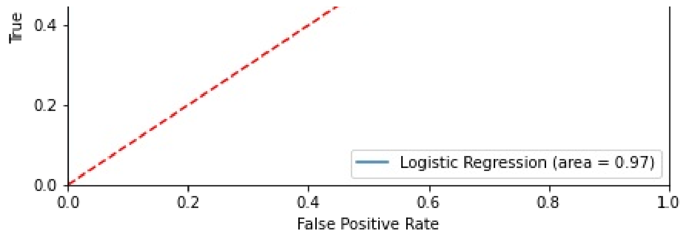
ROC-AUC

- ROC-AUC on training set

In [166…

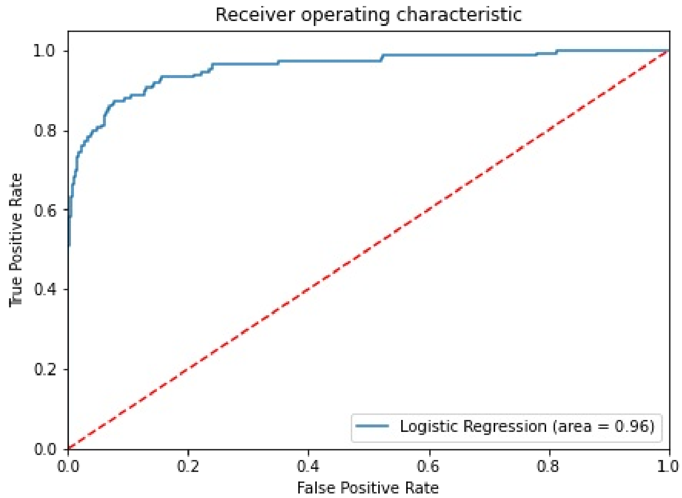
```
logit_roc_auc_train = roc_auc_score(y_train, lg4.predict(X_train4))
fpr, tpr, thresholds = roc_curve(y_train, lg4.predict(X_train4))
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```





- ROC-AUC on test set

In [167...
```
logit_roc_auc_test = roc_auc_score(y_test, lg4.predict(X_test4)) #roc_auc test
fpr, tpr, thresholds = roc_curve(y_test, lg4.predict(X_test4))
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')
plt.legend(loc="lower right")
plt.show()
```



- Logistic Regression model is giving a generalized performance on training and test set.

Converting coefficients to odds

- The coefficients of the logistic regression model are in terms of log(odd), to find the odds we have to take the exponential of the coefficients.
- Therefore, **odds = exp(b)**
- The percentage change in odds is given as **odds = (exp(b) - 1) * 100**

- Odds from coefficients

In [168...
```
odds = np.exp(lg4.params) # converting coefficients to odds
pd.set_option('display.max_columns',None)  # removing limit from number of columns to display
pd.DataFrame(odds, X_train4.columns, columns=['odds']).T # adding the odds to a dataframe
```

Out[168...

	const	Income	CCAvg	Family_2	Family_3	Family_4	Education_2	Education_3	Securities_Account_1	CD_Account_1	Online_1
odds	4.520806e-07	1.068674	1.678945	1.086525	15.856514	5.956709	67.485613	87.47987	0.364301	39.491291	0.554852

- Percentage change in odds

In [169...
```
perc_change_odds = (np.exp(lg4.params)-1)*100 # finding the percentage change
pd.set_option('display.max_columns',None) # removing limit from number of columns to display
pd.DataFrame(perc_change_odds, X_train4.columns, columns=['change_odds%']).T # adding the change_odds% to a datai
```

Out[169...

	const	Income	CCAvg	Family_2	Family_3	Family_4	Education_2	Education_3	Securities_Account_1	CD_Accou
change_odds%	-99.999955	6.867374	67.894538	8.652495	1485.651446	495.670882	6648.561295	8647.986976	-63.569905	3849.12

Coefficent Interpertations:

1. Having all other variables constant, 1 unit change in Income will increase the chances of taking a loan.
2. CCAvg, Family size, Education, 1 unit change will bring about positive change and increase chances of taking a loan.
3. HAving a securities account will decrease the changes of taking a loan by 63%.
4. Having an online account also decreases the chances of taking a loan by 44.5%.
5. Owning a credit card also decreases the chance of taking a loan by 62%.

In [170...

```
# 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, lg4.predict(X_test4))

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

```
0.0976608563542546
```

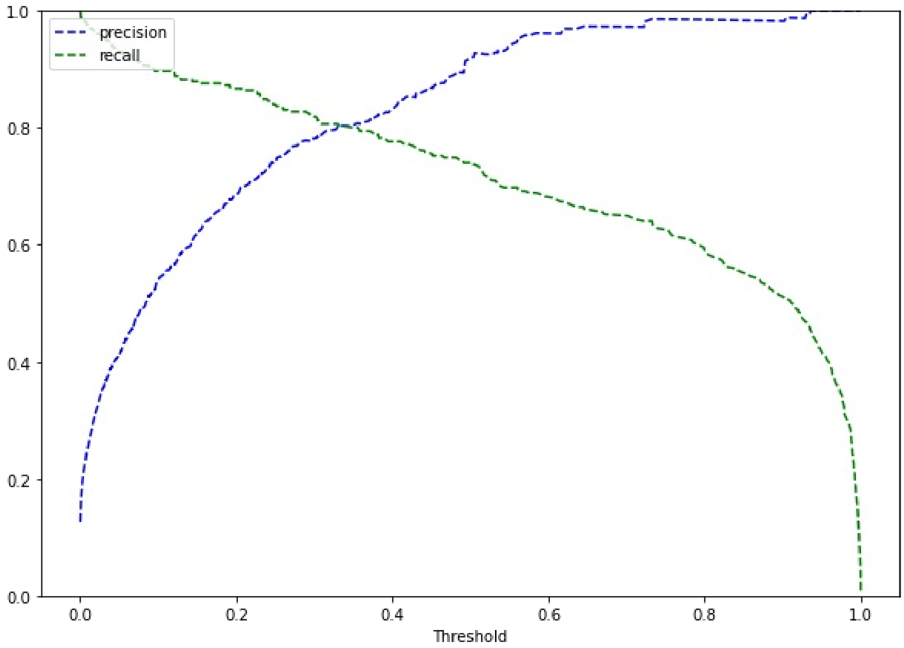
In [171...

```
scores_LR = get_metrics_score1(lg4,X_train4,X_test4,y_train,y_test,threshold=optimal_threshold_auc_roc,roc=True)
```

```
Accuracy on training set :  0.9165714285714286
Accuracy on test set :  0.918
Recall on training set :  0.8972809667673716
Recall on test set :  0.8657718120805369
Precision on training set :  0.5351351351351351
Precision on test set :  0.5560344827586207
F1 on training set :  0.6704288939051918
F1 on test set :  0.6771653543307086
ROC-AUC Score on training set :  0.9079336357976966
ROC-AUC Score on test set :  0.8947659948633624
```

In [172...

```
y_scores=lg4.predict(X_train4)
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 [173...

```
optimal_threshold_curve = 0.38
```

```
optimal_threshold_curve = 0.58

scores_LR = get_metrics_score1(lg4,X_train4,X_test4,y_train,y_test,threshold=optimal_threshold_curve,roc=True)
```

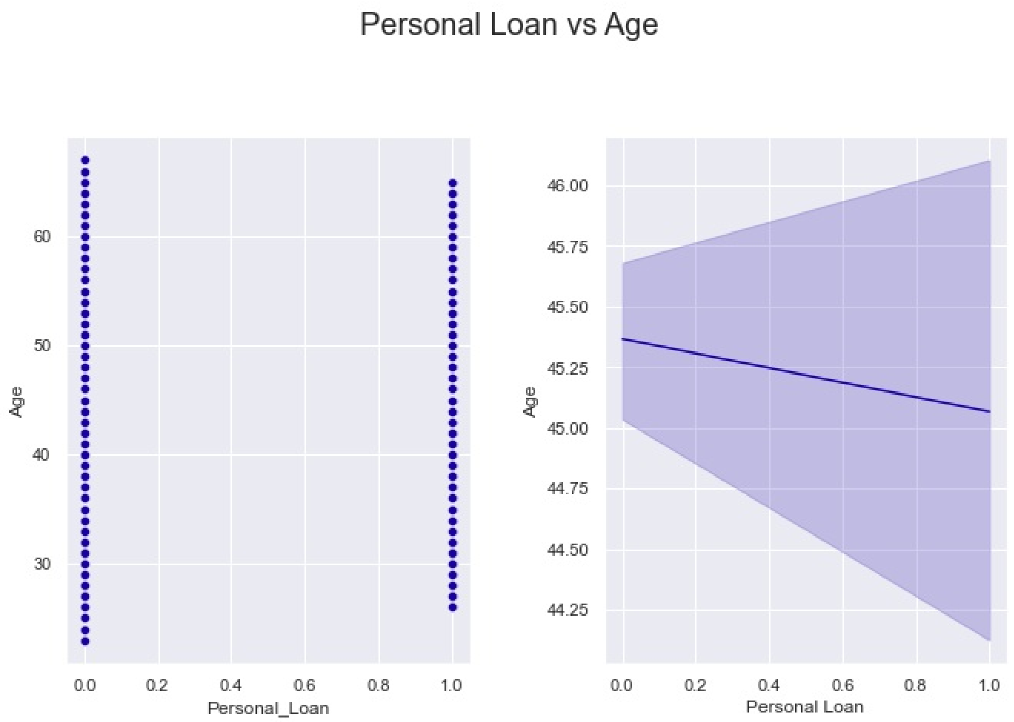
```
Accuracy on training set :  0.9637142857142857
Accuracy on test set :  0.9573333333333334
Recall on training set :  0.7885196374622356
Recall on test set :  0.7114093959731543
Precision on training set :  0.8207547169811321
Precision on test set :  0.8346456692913385
F1 on training set :  0.8043143297380587
F1 on test set :  0.7681159420289855
ROC-AUC Score on training set :  0.8852664454272365
ROC-AUC Score on test set :  0.8479326772611886
```

Observations on Personal Loan vs variables not considered for the final model in Logistic regression

Personal Loan vs Age

In [91]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,7)) #creating subplots
ax1 = sns.scatterplot(x = 'Personal_Loan', ax = ax1,  y = 'Age', data = data) #scatterplot fo age vs price
ax2 = sns.lineplot(x = 'Personal_Loan', ax = ax2, y = 'Age', data = data) #lineplot of age vs price
plt.suptitle('Personal Loan vs Age',fontsize=20)
plt.xlabel('Personal Loan')
fig.tight_layout(pad=3.0)
plt.ylabel('Age')
```

Out[91]:
```
Text(353.23863636363626, 0.5, 'Age')
```



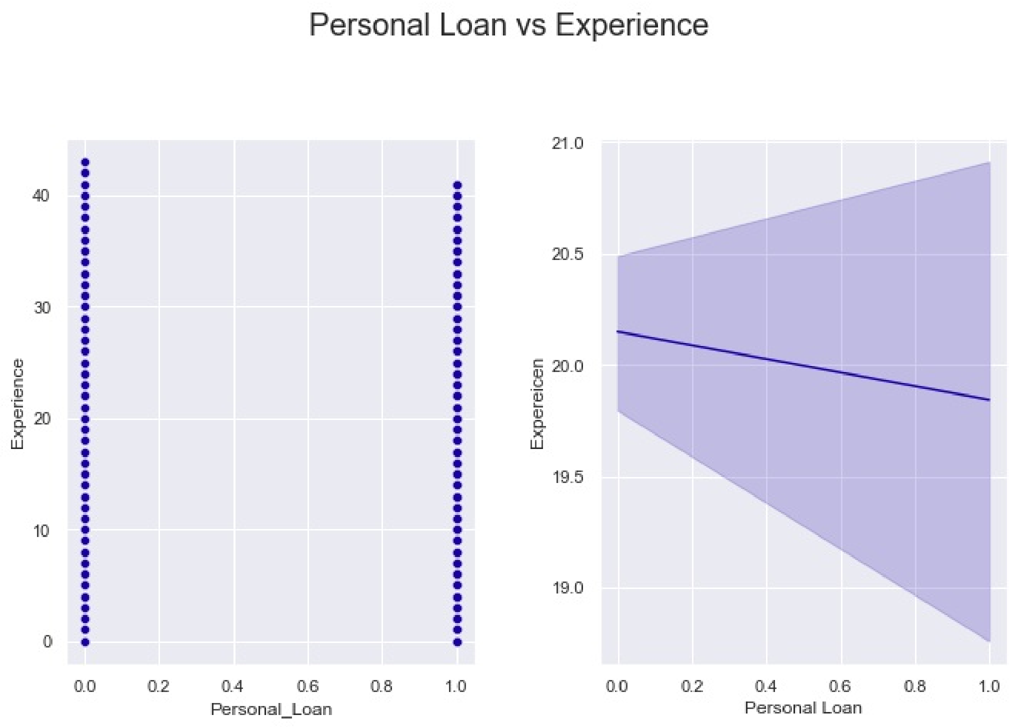
Observation:

1. Age decreased for cleints that did take personal loan.
2. On average clients who are 45 and a half years took no laons
3. On average cleitns who just turned 45 were more likey to to take a loan

Personal Loan vs Experience

In [93]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,7)) #creating subplots
ax1 = sns.scatterplot(x = 'Personal_Loan', ax = ax1,  y = 'Experience', data = data) #scatterplot fo age vs price
ax2 = sns.lineplot(x = 'Personal_Loan', ax = ax2, y = 'Experience', data = data) #lineplot of age vs price
plt.suptitle('Personal Loan vs Experience',fontsize=20)
plt.xlabel('Personal Loan')
fig.tight_layout(pad=3.0)
plt.ylabel('Expereicen')
```

Out[93]: Text(359.36363636363626, 0.5, 'Expereicen')



Observation:

1. Clients who had lower experience took more personal loan
2. CLients who had higher experience took less personal loans

Personal Loan vs Mortgage

In [95]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,7)) #creating subplots
ax1 = sns.scatterplot(x = 'Personal_Loan', ax = ax1,  y = 'Mortgage', data = data) #scatterplot fo age vs price
ax2 = sns.lineplot(x = 'Personal_Loan', ax = ax2, y = 'Mortgage', data = data) #lineplot of age vs price
plt.suptitle('Personal Loan vs Mortgage',fontsize=20)
plt.xlabel('Personal Loan')
fig.tight_layout(pad=3.0)
plt.ylabel('Mortgage')
```

Out[95]: Text(368.61363636363626, 0.5, 'Mortgage')

Observation:

1. Clients who had lower Mortgage did not take personal loans
2. Clients who on average had 75 mortgage were more likely to take personal loans

Data Preperation for Decision trees

In [100…

```
data = data.drop(['Major_city'],axis=1)
dummy_data = pd.get_dummies(data, columns=["Family", 'Education', 'Securities_Account', 'CD_Account', 'Online', '
dummy_data.head()
```

Out[100…

ID	Age	Experience	Income	CCAvg	Mortgage	Personal_Loan	Family_2	Family_3	Family_4	Education_2	Education_3	Securities_Account_1
1	25	1	49.0	1.6	0.0	0	0	0	1	0	0	1
2	45	19	34.0	1.5	0.0	0	0	1	0	0	0	1
3	39	15	11.0	1.0	0.0	0	0	0	0	0	0	0
4	35	9	100.0	2.7	0.0	0	0	0	0	1	0	0
5	35	8	45.0	1.0	0.0	0	0	0	1	1	0	0

Model Building - Approach

1. Data preparation
2. Partition the data into train and test set.
3. Built a CART model on the train data.
4. Tune the model and prune the tree, if required.
5. Test the data on test set.

In [101…

```
column_names = list(dummy_data.columns)
column_names.remove('Personal_Loan')                    # Keep only names of features by removing the name of ta
feature_names = column_names
print(feature_names)
```

```
['Age', 'Experience', 'Income', 'CCAvg', 'Mortgage', 'Family_2', 'Family_3', 'Family_4', 'Education_2', 'Educatio
n_3', 'Securities_Account_1', 'CD_Account_1', 'Online_1', 'CreditCard_1']
```

Split Data

In [102…

```
X = dummy_data.drop('Personal_Loan',axis=1)                                                   # Features
y = dummy_data['Personal_Loan'].astype('int64')                                               # Labels (Target Vari
# converting target to integers - since some functions might not work with bool type
```

In [103…

```
# Splitting data into training and test set:
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3, random_state=1)
print(X_train.shape, X_test.shape)
```

```
(3500, 14) (1500, 14)
```

Build Decision Tree Model

- We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split.
- If the frequency of class A is 10% and the frequency of class B is 90%, then class B will become the dominant class and the decision tree will become biased toward the dominant classes.
- In this case, 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.

In [109…

```
from sklearn import tree
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='gini',class_weight={0:0.15,1:0.85},random_state=1)
```

In [110…

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

Out[110… DecisionTreeClassifier(class_weight={0: 0.15, 1: 0.85}, random_state=1)

In [111…
```
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                  columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [188…
```
y_train.value_counts(1)
```

Out[188…
```
0    0.905429
1    0.094571
Name: Personal_Loan, dtype: float64
```

We only have 9% of positive classes, so if our model marks each sample as negative, then also we'll get 90% accuracy, hence accuracy is not a good metric to evaluate here.

In [189…
```
##  Function to calculate recall score
def get_recall_score(model):
    '''
    model : classifier to predict values of X

    '''
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)
    print("Recall on training set : ",metrics.recall_score(y_train,pred_train))
    print("Recall on test set : ",metrics.recall_score(y_test,pred_test))
```

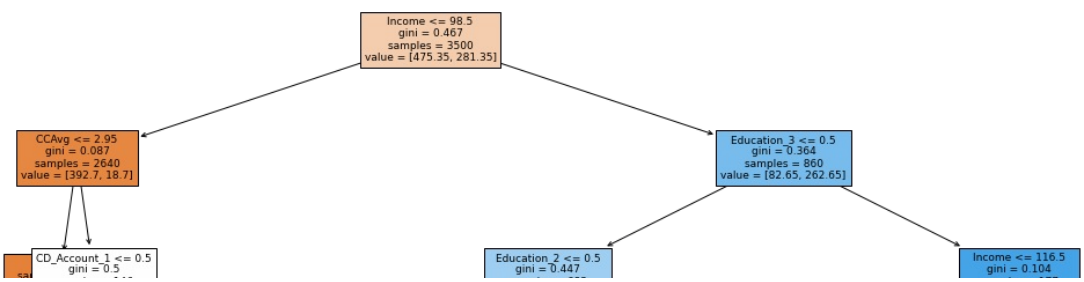
In [190…
```
get_recall_score(model)
```

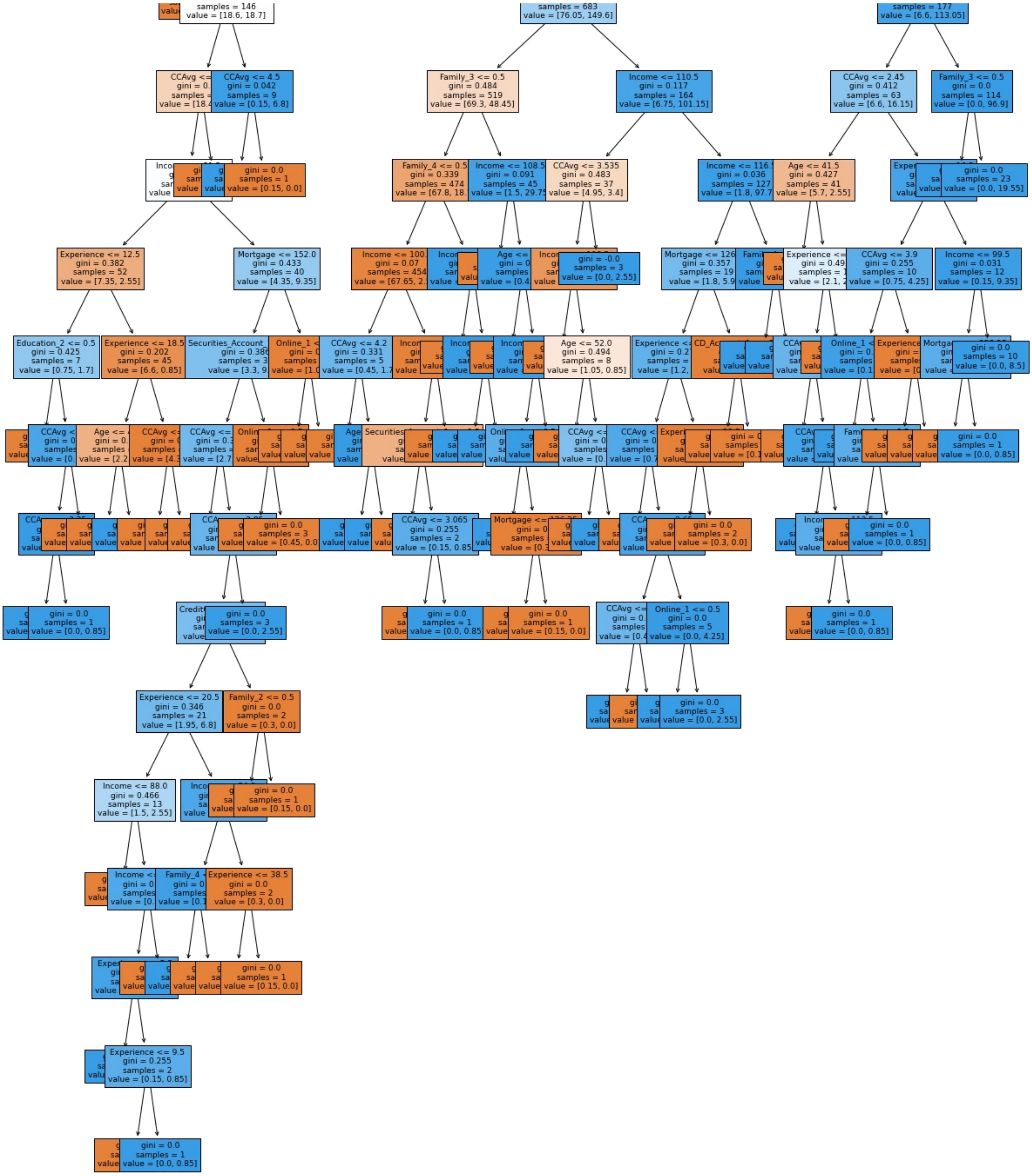
```
Recall on training set :  1.0
Recall on test set :  0.8926174496644296
```

There isn't a huge disparity in performance of model on training set and test set, which suggests that the model is good, but there is slight overfitting.

Visualizing the Decision Tree

In [194…
```
plt.figure(figsize=(20,30))
out = tree.plot_tree(model,feature_names=feature_names,filled=True,fontsize=9,node_ids=False,class_names=None,)
#below code will add arrows to the decision tree split if they are missing
for o in out:
     arrow = o.arrow_patch
     if arrow is not None:
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```





In [195…

```
# 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: [374.10, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- CD_Account_1 <= 0.50
|   |   |   |--- CCAvg <= 3.95
|   |   |   |   |--- Income <= 81.50
|   |   |   |   |   |--- Experience <= 12.50
|   |   |   |   |   |   |--- Education_2 <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.60, 0.00] class: 0
|   |   |   |   |   |   |--- Education_2 >  0.50
|   |   |   |   |   |   |   |--- CCAvg <= 3.50
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.25
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |   |--- CCAvg >  3.25
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
```

```
|   |   |   |   |   |   |   |--- CCAvg >  3.50
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |--- Experience >  12.50
|   |   |   |   |   |   |--- Experience <= 18.50
|   |   |   |   |   |   |   |--- Age <= 42.50
|   |   |   |   |   |   |   |   |--- weights: [2.25, 0.00] class: 0
|   |   |   |   |   |   |   |--- Age >  42.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |--- Experience >  18.50
|   |   |   |   |   |   |   |--- CCAvg <= 3.10
|   |   |   |   |   |   |   |   |--- weights: [1.05, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  3.10
|   |   |   |   |   |   |   |   |--- weights: [3.30, 0.00] class: 0
|   |   |   |   |--- Income >  81.50
|   |   |   |   |   |--- Mortgage <= 152.00
|   |   |   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.85
|   |   |   |   |   |   |   |   |   |--- CreditCard_1 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- Experience <= 20.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 5
|   |   |   |   |   |   |   |   |   |   |--- Experience >  20.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 3
|   |   |   |   |   |   |   |   |   |--- CreditCard_1 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- Family_2 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- Family_2 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  3.85
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.55] class: 1
|   |   |   |   |   |   |--- Securities_Account_1 >  0.50
|   |   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |--- Mortgage >  152.00
|   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.75, 0.00] class: 0
|   |   |   |--- CCAvg >  3.95
|   |   |   |   |--- weights: [6.75, 0.00] class: 0
|   |   |--- CD_Account_1 >  0.50
|   |   |   |--- CCAvg <= 4.50
|   |   |   |   |--- weights: [0.00, 6.80] class: 1
|   |   |   |--- CCAvg >  4.50
|   |   |   |   |--- weights: [0.15, 0.00] class: 0
|--- Income >  98.50
|   |--- Education_3 <= 0.50
|   |   |--- Education_2 <= 0.50
|   |   |   |--- Family_3 <= 0.50
|   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |--- Income <= 100.00
|   |   |   |   |   |   |--- CCAvg <= 4.20
|   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  4.20
|   |   |   |   |   |   |   |--- Age <= 54.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |--- Age >  54.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |--- Income >  100.00
|   |   |   |   |   |   |--- Income <= 103.50
|   |   |   |   |   |   |   |--- Securities_Account_1 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [2.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Securities_Account_1 >  0.50
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.06
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  3.06
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |--- Income >  103.50
|   |   |   |   |   |   |   |--- weights: [64.95, 0.00] class: 0
|   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |--- Income <= 102.00
|   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |--- Income >  102.00
|   |   |   |   |   |   |--- Income <= 106.00
|   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |--- Income >  106.00
|   |   |   |   |   |   |   |--- weights: [0.00, 15.30] class: 1
|   |   |   |--- Family_3 >  0.50
|   |   |   |   |--- Income <= 108.50
```

```
|   |   |   |   |   |--- weights: [1.05, 0.00] class: 0
|   |   |   |   |--- Income >  108.50
|   |   |   |   |   |--- Age <= 26.00
|   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |--- Age >  26.00
|   |   |   |   |   |   |--- Income <= 118.00
|   |   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |   |--- Mortgage <= 126.25
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Mortgage >  126.25
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |--- Income >  118.00
|   |   |   |   |   |   |   |--- weights: [0.00, 28.05] class: 1
|   |   |--- Education_2 >  0.50
|   |   |   |--- Income <= 110.50
|   |   |   |   |--- CCAvg <= 3.54
|   |   |   |   |   |--- Income <= 106.50
|   |   |   |   |   |   |--- weights: [3.90, 0.00] class: 0
|   |   |   |   |   |--- Income >  106.50
|   |   |   |   |   |   |--- Age <= 52.00
|   |   |   |   |   |   |   |--- weights: [0.75, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  52.00
|   |   |   |   |   |   |   |--- CCAvg <= 1.85
|   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  1.85
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |--- CCAvg >  3.54
|   |   |   |   |   |--- weights: [0.00, 2.55] class: 1
|   |   |   |--- Income >  110.50
|   |   |   |   |--- Income <= 116.50
|   |   |   |   |   |--- Mortgage <= 126.25
|   |   |   |   |   |   |--- Experience <= 35.50
|   |   |   |   |   |   |   |--- CCAvg <= 1.20
|   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  1.20
|   |   |   |   |   |   |   |   |--- CCAvg <= 2.65
|   |   |   |   |   |   |   |   |   |--- CCAvg <= 1.75
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |   |   |--- CCAvg >  1.75
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  2.65
|   |   |   |   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.55] class: 1
|   |   |   |   |   |   |--- Experience >  35.50
|   |   |   |   |   |   |   |--- Experience <= 38.50
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  38.50
|   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |--- Mortgage >  126.25
|   |   |   |   |   |   |--- CD_Account_1 <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |   |--- CD_Account_1 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |--- Income >  116.50
|   |   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |   |--- weights: [0.00, 70.55] class: 1
|   |   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |   |--- weights: [0.00, 21.25] class: 1
|   |--- Education_3 >  0.50
|   |   |--- Income <= 116.50
|   |   |   |--- CCAvg <= 2.45
|   |   |   |   |--- Age <= 41.50
|   |   |   |   |   |--- weights: [3.60, 0.00] class: 0
|   |   |   |   |--- Age >  41.50
|   |   |   |   |   |--- Experience <= 31.50
|   |   |   |   |   |   |--- CCAvg <= 0.35
|   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  0.35
|   |   |   |   |   |   |   |--- CCAvg <= 1.25
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |--- CCAvg >  1.25
|   |   |   |   |   |   |   |   |--- Income <= 113.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Income >  113.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |--- Experience >  31.50
|   |   |   |   |   |   |--- weights: [1.50, 0.00] class: 0
|   |   |   |--- CCAvg >  2.45
|   |   |   |   |--- Experience <= 16.50
```

```
|   |   |   |   |   |--- CCAvg <= 3.90
|   |   |   |   |   |   |--- Online_1 <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 3.40] class: 1
|   |   |   |   |   |   |--- Online_1 >  0.50
|   |   |   |   |   |   |   |--- Family_3 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |   |--- Family_3 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |--- CCAvg >  3.90
|   |   |   |   |   |   |--- Experience <= 7.00
|   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |--- Experience >  7.00
|   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |--- Experience >  16.50
|   |   |   |   |   |--- Income <= 99.50
|   |   |   |   |   |   |--- Mortgage <= 223.25
|   |   |   |   |   |   |   |--- weights: [0.15, 0.00] class: 0
|   |   |   |   |   |   |--- Mortgage >  223.25
|   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |--- Income >  99.50
|   |   |   |   |   |   |--- weights: [0.00, 8.50] class: 1
|   |   |--- Income >  116.50
|   |   |   |--- Family_3 <= 0.50
|   |   |   |   |--- weights: [0.00, 77.35] class: 1
|   |   |   |--- Family_3 >  0.50
|   |   |   |   |--- weights: [0.00, 19.55] class: 1
```

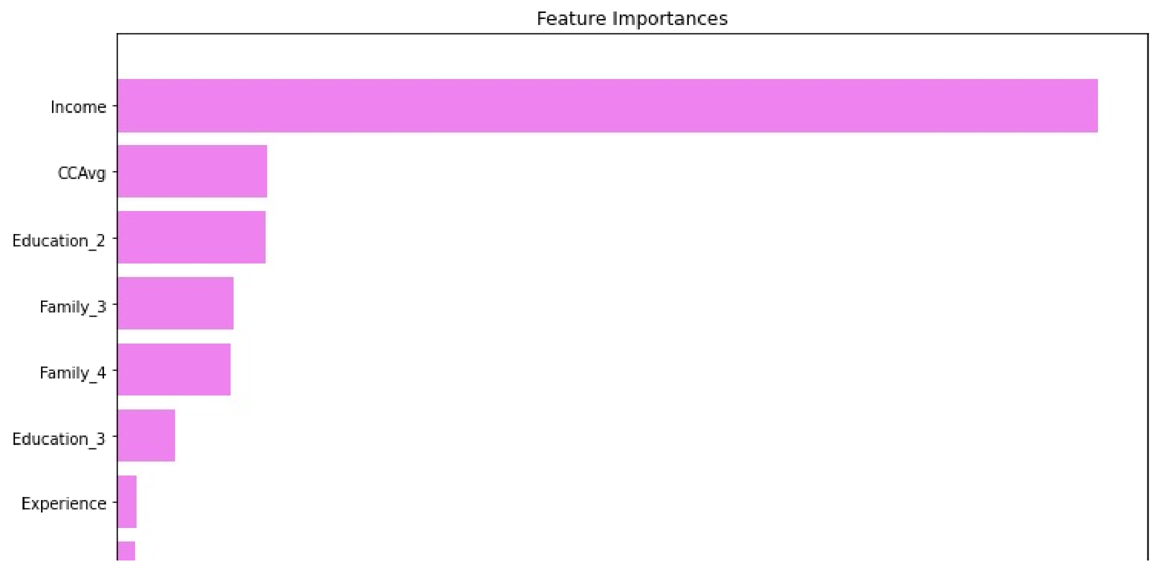
In [196…

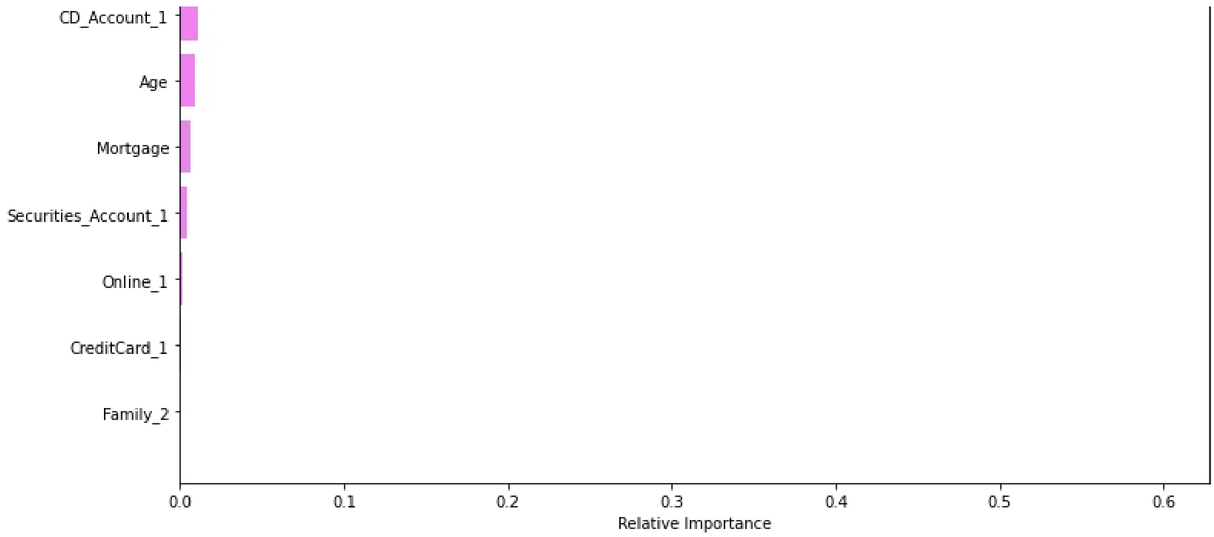
```
# importance of features in the tree building ( The importance of a feature is computed as the
#(normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance )

print (pd.DataFrame(model.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by = 'Imp
```

```
                                Imp
Income               5.978824e-01
CCAvg                9.132013e-02
Education_2          9.029869e-02
Family_3             7.104576e-02
Family_4             6.900636e-02
Education_3          3.514783e-02
Experience           1.198951e-02
CD_Account_1         1.099955e-02
Age                  9.174049e-03
Mortgage             5.887150e-03
Securities_Account_1 4.716203e-03
Online_1             1.541163e-03
CreditCard_1         9.911646e-04
Family_2             4.993910e-18
```

In [197…

```
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='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```





According to the decision tree model, Income is the most important variable for predicting the Revenue. Followed by CCAvg and Education and Family Size.

The tree above is very complex and difficult to interpret.

Reducing over fitting

Using GridSearch for Hyperparameter tuning of our tree model

- Hyperparameter tuning is also tricky in the sense that there is no direct way to calculate how a change in the hyperparameter value will reduce the loss of your model, so we usually resort to experimentation. i.e we'll use Grid search
- 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.

In [200…

```
from sklearn.model_selection import GridSearchCV
```

In [201…

```
# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1,class_weight = {0:.15,1:.85})

# Grid of parameters to choose from
parameters = {
            'max_depth': np.arange(1,10),
            'criterion': ['entropy','gini'],
            'splitter': ['best','random'],
            'min_impurity_decrease': [0.000001,0.00001,0.0001],
            'max_features': ['log2','sqrt']
             }

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

# Run the grid search
grid_obj = GridSearchCV(estimator, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

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

# Fit the best algorithm to the data.
estimator.fit(X_train, y_train)
```

Out[201…

```
DecisionTreeClassifier(class_weight={0: 0.15, 1: 0.85}, criterion='entropy',
                       max_depth=3, max_features='log2',
                       min_impurity_decrease=1e-06, random_state=1)
```

In [206…

```
get_recall_score(estimator)
```

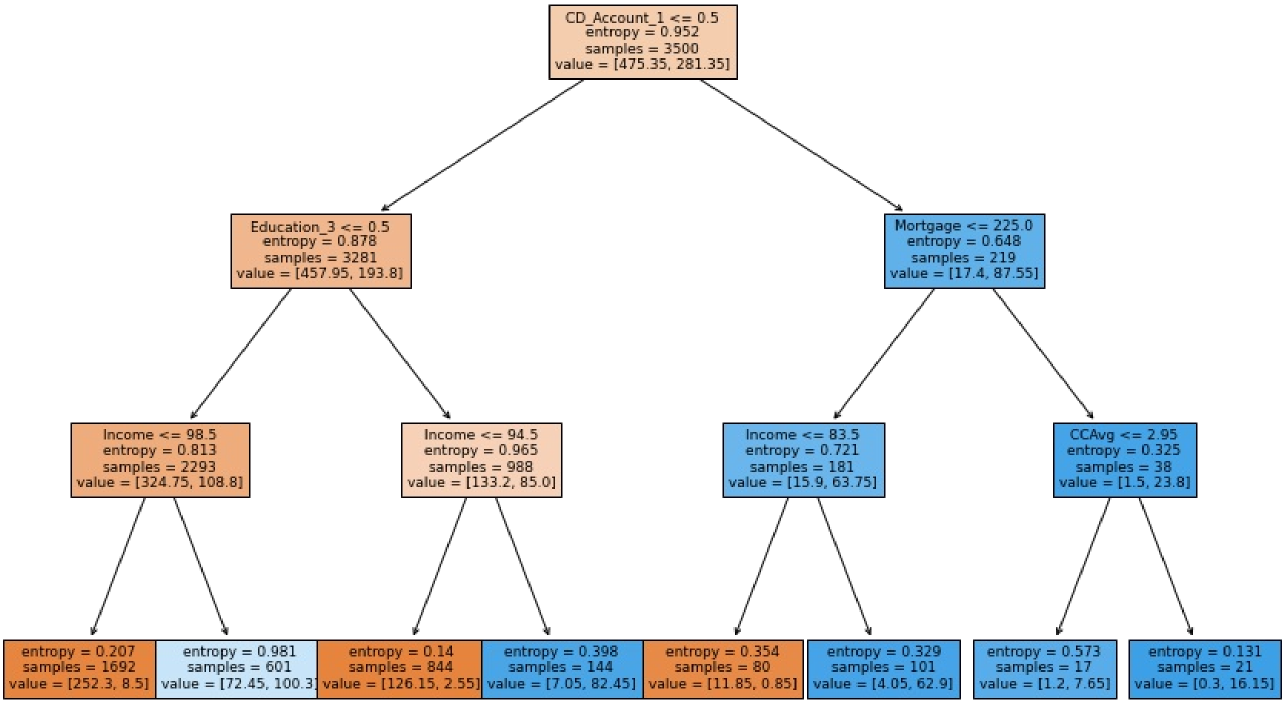
```
Recall on training set :  0.9577039274924471
Recall on test set :  0.9328859060402684
```

Recall has improved for both train and test set after hyperparameter tuning and we have a generalized model.

Visualizing the Decision Tree

In [207…

```
plt.figure(figsize=(15,10))
out = tree.plot_tree(estimator,feature_names=feature_names,filled=True,fontsize=9,node_ids=False,class_names=None
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```



In [208…

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

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

```
|--- CD_Account_1 <= 0.50
|   |--- Education_3 <= 0.50
|   |   |--- Income <= 98.50
|   |   |   |--- weights: [252.30, 8.50] class: 0
|   |   |--- Income >  98.50
|   |   |   |--- weights: [72.45, 100.30] class: 1
|   |--- Education_3 >  0.50
|   |   |--- Income <= 94.50
|   |   |   |--- weights: [126.15, 2.55] class: 0
|   |   |--- Income >  94.50
|   |   |   |--- weights: [7.05, 82.45] class: 1
|--- CD_Account_1 >  0.50
|   |--- Mortgage <= 225.00
|   |   |--- Income <= 83.50
|   |   |   |--- weights: [11.85, 0.85] class: 0
|   |   |--- Income >  83.50
|   |   |   |--- weights: [4.05, 62.90] class: 1
|   |--- Mortgage >  225.00
|   |   |--- CCAvg <= 2.95
|   |   |   |--- weights: [1.20, 7.65] class: 1
|   |   |--- CCAvg >  2.95
|   |   |   |--- weights: [0.30, 16.15] class: 1
```

In [209…

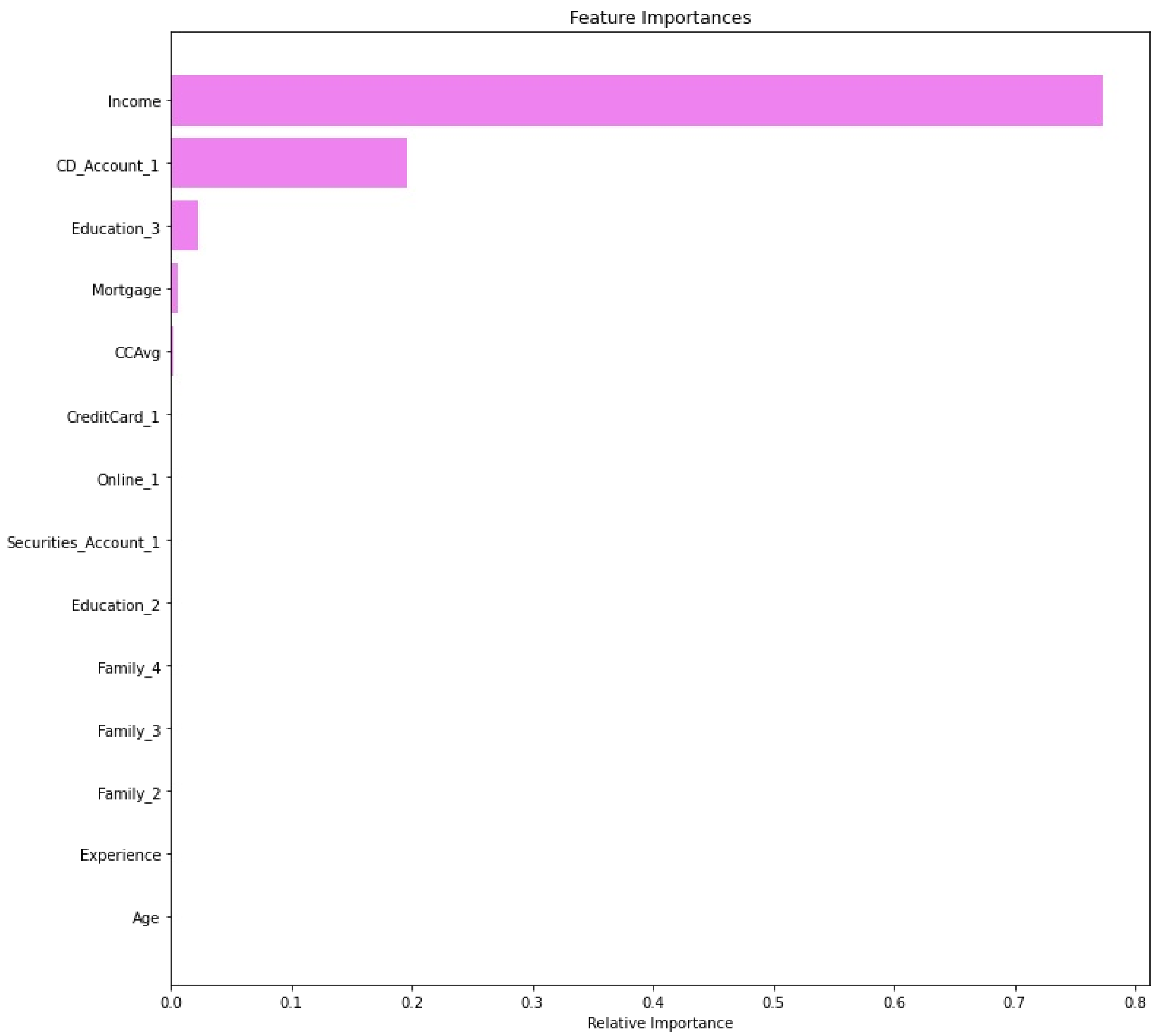
```
# importance of features in the tree building ( The importance of a feature is computed as the
#(normalized) total reduction of the 'criterion' brought by that feature. It is also known as the Gini importance
```

```
print (pd.DataFrame(estimator.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by =

#Here we will see that importance of features has increased
```

```
                          Imp
Income               0.773069
CD_Account_1         0.195777
Education_3          0.023006
Mortgage             0.005746
CCAvg                0.002402
Age                  0.000000
Experience           0.000000
Family_2             0.000000
Family_3             0.000000
Family_4             0.000000
Education_2          0.000000
Securities_Account_1 0.000000
Online_1             0.000000
CreditCard_1         0.000000
```

In [210…

```
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 [211…

```
clf = DecisionTreeClassifier(random_state=1,class_weight = {0:0.15,1:0.85})
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

In [212…

```
pd.DataFrame(path)
```

Out[212…

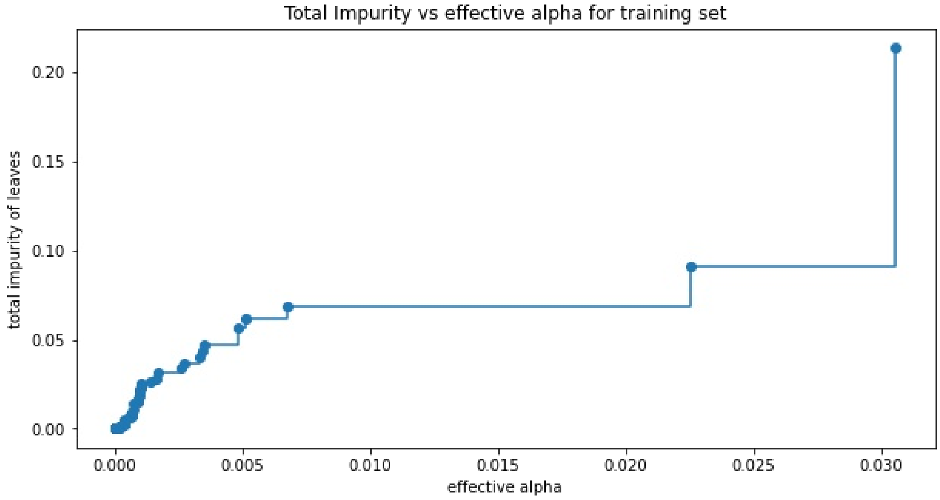
ccp_alphas	impurities

0	0.000000e+00	-9.810318e-15
1	1.320471e-19	-9.810186e-15
2	7.482671e-19	-9.809437e-15
3	7.482671e-19	-9.808689e-15
4	1.760629e-18	-9.806928e-15
5	2.332833e-18	-9.804596e-15
6	2.494224e-18	-9.802101e-15
7	2.905037e-18	-9.799196e-15
8	3.521257e-18	-9.795675e-15
9	4.665666e-18	-9.791009e-15
10	4.665666e-18	-9.786344e-15
11	5.854090e-18	-9.780490e-15
12	7.658734e-18	-9.772831e-15
13	9.478050e-18	-9.763353e-15
14	8.081285e-17	-9.682540e-15
15	2.985586e-16	-9.383981e-15
16	1.872164e-04	3.744328e-04
17	1.872164e-04	7.488657e-04
18	1.914713e-04	1.131808e-03
19	1.950992e-04	1.522007e-03
20	3.350189e-04	1.857026e-03
21	3.369896e-04	2.194015e-03
22	3.643130e-04	2.558328e-03
23	3.829427e-04	2.941271e-03
24	3.879017e-04	3.329173e-03
25	3.905508e-04	4.500825e-03
26	3.928099e-04	4.893635e-03
27	5.528735e-04	5.999382e-03
28	5.860688e-04	6.585451e-03
29	6.546462e-04	7.240097e-03
30	6.554717e-04	7.895569e-03
31	6.758139e-04	8.571383e-03
32	6.925559e-04	9.263938e-03
33	7.289811e-04	1.072190e-02
34	7.504611e-04	1.447421e-02
35	8.789656e-04	1.535317e-02
36	9.093369e-04	1.626251e-02
37	9.095010e-04	1.717201e-02
38	9.404360e-04	1.811245e-02
39	9.407728e-04	1.999399e-02
40	9.951370e-04	2.198427e-02
41	1.011155e-03	2.299542e-02
42	1.013173e-03	2.400859e-02
43	1.018946e-03	2.502754e-02
44	1.399934e-03	2.642747e-02
45	1.638043e-03	2.806552e-02
46	1.686407e-03	3.143833e-02
47	2.602631e-03	3.404096e-02
48	2.742431e-03	3.678339e-02
49	3.335999e-03	4.011939e-02
50	3.409906e-03	4.352930e-02
51	3.527226e-03	4.705652e-02
52	4.797122e-03	5.665076e-02
53	5.138280e-03	6.178904e-02
54	6.725814e-03	6.851486e-02

55	2.253222e-02	9.104708e-02
56	3.057320e-02	2.133399e-01
57	2.537957e-01	4.671356e-01

In [213…

```
fig, ax = plt.subplots(figsize=(10,5))
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. The last value in ccp_alphas is the alpha value that prunes the whole tree, leaving the tree, clfs[-1], with one node.

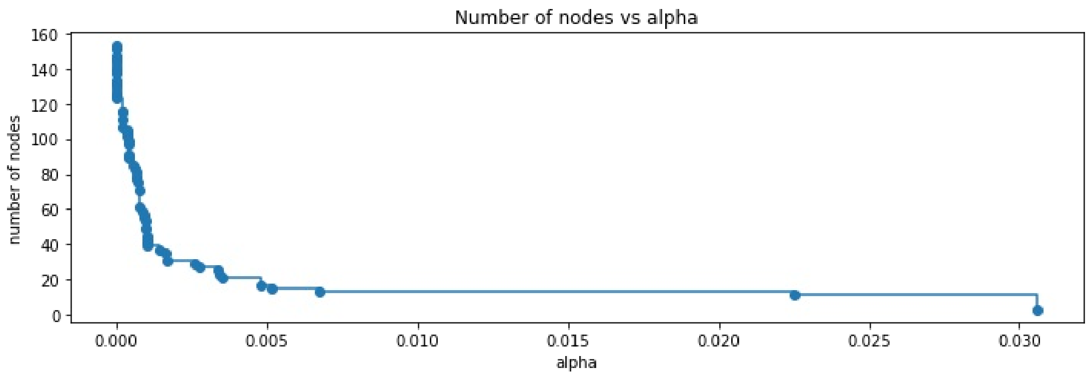
In [215…

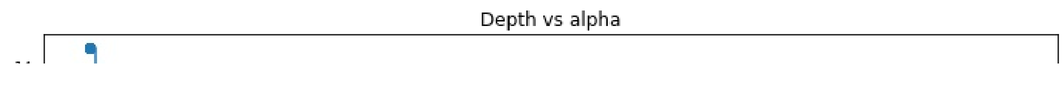
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha,class_weight = {0:0.15,1:0.85})
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

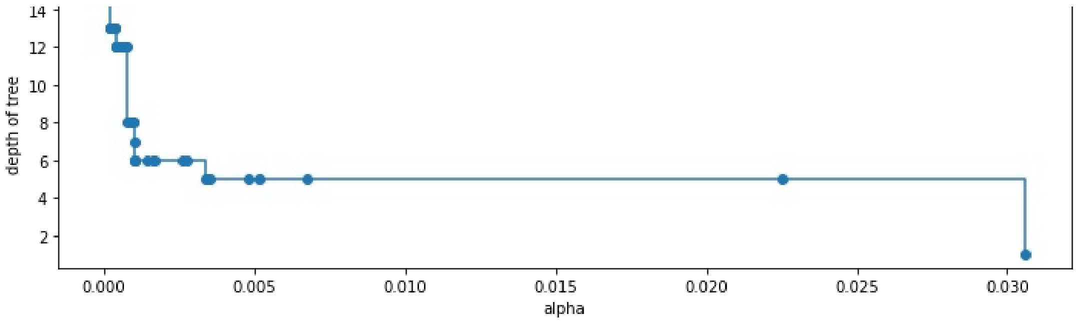
```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.25379571489481
```

In [216…

```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1,figsize=(10,7))
ax[0].plot(ccp_alphas, node_counts, marker='o', drawstyle="steps-post")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker='o', drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```





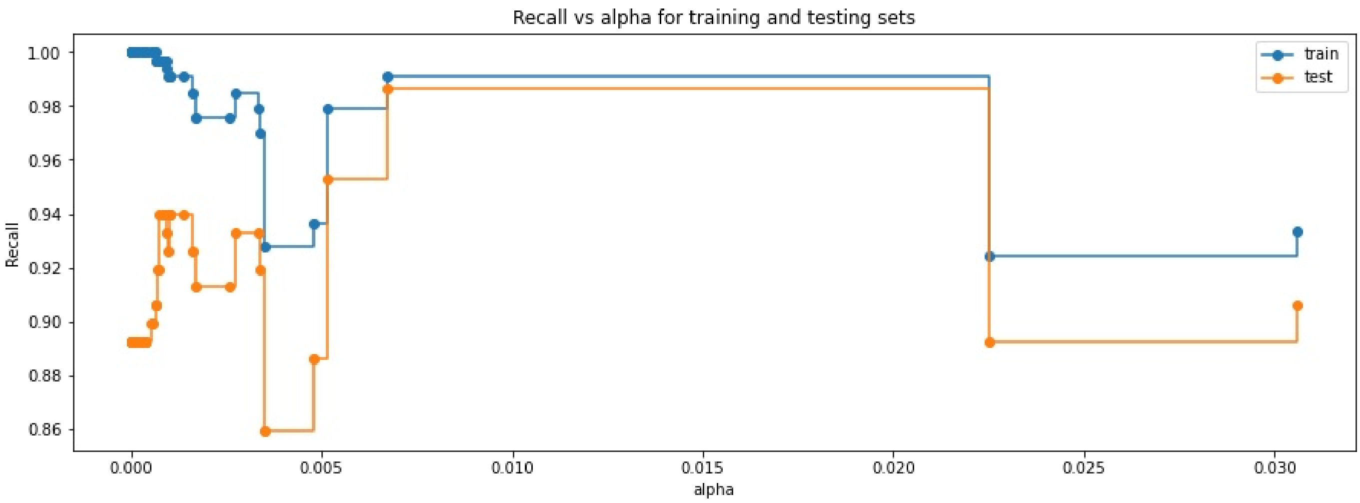


In [217…
```
recall_train=[]
for clf in clfs:
    pred_train3=clf.predict(X_train)
    values_train=metrics.recall_score(y_train,pred_train3)
    recall_train.append(values_train)
```

In [218…
```
recall_test=[]
for clf in clfs:
    pred_test3=clf.predict(X_test)
    values_test=metrics.recall_score(y_test,pred_test3)
    recall_test.append(values_test)
```

In [219…
```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
```

In [220…
```
fig, ax = plt.subplots(figsize=(15,5))
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()
```



Maximum value of Recall is at 0.030 alpha, but if we choose decision tree will only have a root node and we would lose the buisness rules, instead we can choose alpha 0.0067 retaining information and getting higher recall.

In [222…
```
# creating the model where we get highest train and test recall
index_best_model = np.argmax(recall_test)
best_model = clfs[index_best_model]
print(best_model)
```

```
DecisionTreeClassifier(ccp_alpha=0.006725813690407138,
                       class_weight={0: 0.15, 1: 0.85}, random_state=1)
```

In [223…
```
best_model.fit(X_train, y_train)
```

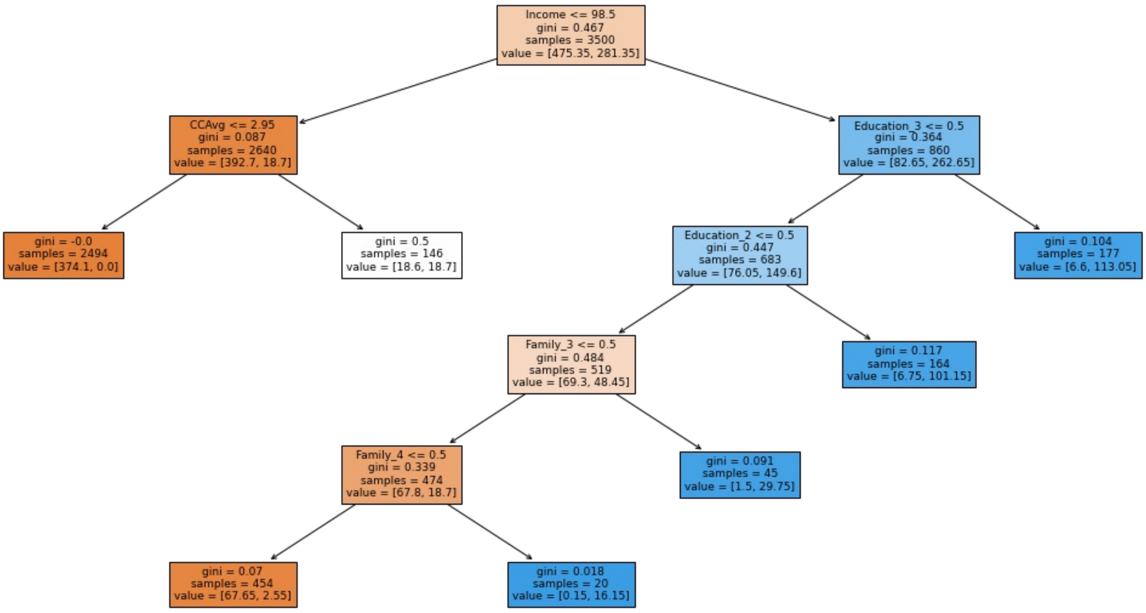
Out[223…
```
DecisionTreeClassifier(ccp_alpha=0.006725813690407138,
                       class_weight={0: 0.15, 1: 0.85}, random_state=1)
```

In [225…
```
get_recall_score(best_model)
```

```
Recall on training set :  0.9909365558912386
Recall on test set :  0.9865771812080537
```

Visualizing the Decision Tree

In [229…
```
plt.figure(figsize=(20,10))

out = tree.plot_tree(best_model,feature_names=feature_names,filled=True,fontsize=9,node_ids=False,class_names=Nor
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```



- This model might be giving the highest recall but a buisness would not be able to use it to actually target the potential customers.

Creating model with 0.0067 ccp_alpha

In [233…
```
best_model2 = DecisionTreeClassifier(ccp_alpha=0.0067,
                       class_weight={0: 0.15, 1: 0.85}, random_state=1)
best_model2.fit(X_train, y_train)
```

Out[233…
```
DecisionTreeClassifier(ccp_alpha=0.0067, class_weight={0: 0.15, 1: 0.85},
                       random_state=1)
```

In [235…
```
get_recall_score(best_model2)
```

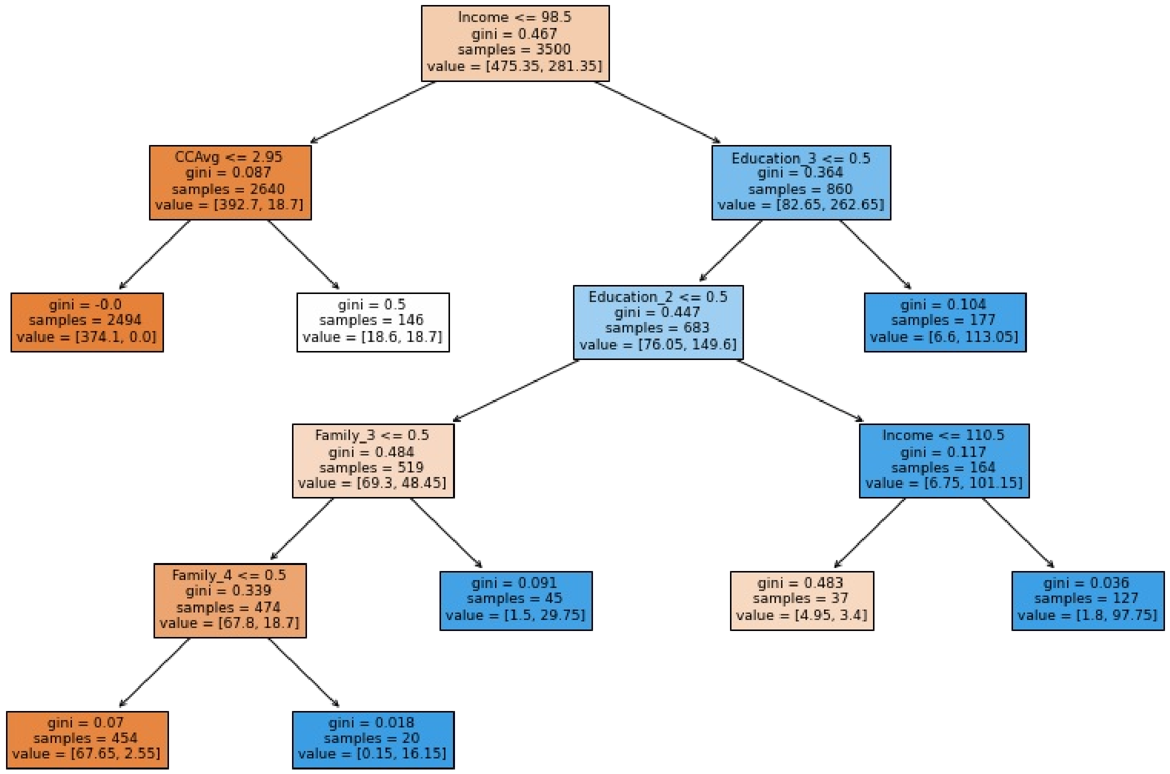
```
Recall on training set :  0.9788519637462235
Recall on test set :  0.9530201342281879
```

The results have improved from the initial model and we have got higher recall than the hyperparameter tuned model and generalized decision tree - having comparable performance on training and test set.

Visualizing the Decision Tree

In [236…
```
plt.figure(figsize=(15,10))

out = tree.plot_tree(best_model2,feature_names=feature_names,filled=True,fontsize=9,node_ids=False,class_names=No
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
```

```
        arrow.set_edgecolor('black')
        arrow.set_linewidth(1)
plt.show()
```



In [237…

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

print(tree.export_text(best_model2,feature_names=feature_names,show_weights=True))
```

```
|--- Income <= 98.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [374.10, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- weights: [18.60, 18.70] class: 1
|--- Income >  98.50
|   |--- Education_3 <= 0.50
|   |   |--- Education_2 <= 0.50
|   |   |   |--- Family_3 <= 0.50
|   |   |   |   |--- Family_4 <= 0.50
|   |   |   |   |   |--- weights: [67.65, 2.55] class: 0
|   |   |   |   |--- Family_4 >  0.50
|   |   |   |   |   |--- weights: [0.15, 16.15] class: 1
|   |   |   |--- Family_3 >  0.50
|   |   |   |   |--- weights: [1.50, 29.75] class: 1
|   |   |--- Education_2 >  0.50
|   |   |   |--- Income <= 110.50
|   |   |   |   |--- weights: [4.95, 3.40] class: 0
|   |   |   |--- Income >  110.50
|   |   |   |   |--- weights: [1.80, 97.75] class: 1
|   |--- Education_3 >  0.50
|   |   |--- weights: [6.60, 113.05] class: 1
```

In [238…

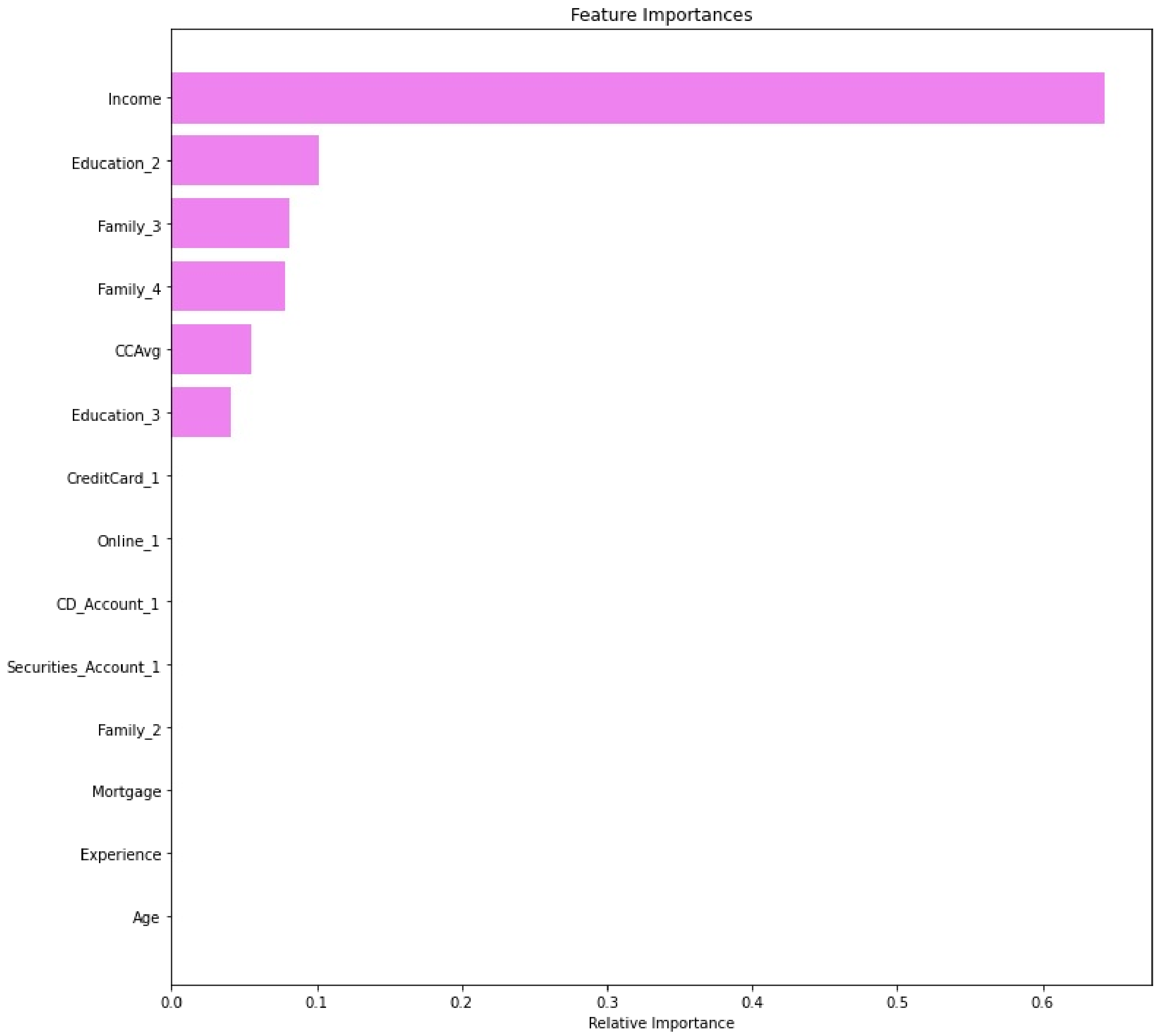
```
# importance of features in the tree building ( The importance of a feature is computed as the
#(normalized) total reduction of the 'criterion' brought by that feature. It is also known as the Gini importance

print (pd.DataFrame(best_model2.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by
```

```
                    Imp
Income         0.642713
Education_2    0.101569
Family_3       0.081044
Family_4       0.078581
CCAvg          0.055588
Education_3    0.040506
Age            0.000000
Experience     0.000000
Mortgage       0.000000
```

```
Family_2             0.000000
Securities_Account_1 0.000000
CD_Account_1         0.000000
Online_1             0.000000
CreditCard_1         0.000000
```

In [239…

```
importances = best_model2.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()
```



- Income and Education, along with Family size are the top three important features to predict customer sessions contributing to Personal Loan.

Comparing all the decision tree models

In [241…

```
comparison_frame = pd.DataFrame({'Model':['Initial decision tree model','Decision treee with hyperparameter tunir
                                          'Decision tree with post-pruning'], 'Train_Recall':[1,0.95,0.97], 'Test
comparison_frame
```

Out[241…

	Model	Train_Recall	Test_Recall
0	Initial decision tree model	1.00	0.89
1	Decision treee with hyperparameter tuning	0.95	0.93
2	Decision tree with post-pruning	0.97	0.95

Decision tree model with post pruning has given the best recall score on data.

Conclusion

1. We analyzed the chances a client is likely to opt for personal loan, using different techniques like basic EDA analysis, Logistic regression, and by and using Decision Tree Classifier to build a predictive model for the same.
2. The model built can be used to predict if a customer is going to opt for a personal loan or not based on differnet given variables.
3. We visualized different trees and their confusion matrix to get a better understanding of the model. Easy interpretation is one of the key benefits of Decision Trees. Many variables were excluded from the analysis as ZipCode, Age, experience and Mortgage from Logistic regression as these variables did not correlate well with Personal Loan and did not show enough evidence to conclude that these variables were important.
4. We verified the fact that how much less data preparation is needed for Decision Trees and such a simple model gave good results even with outliers and imbalanced classes which shows the robustness of Decision Trees.
5. From all the analysis it was found out that Income, Education, Family and CCAvg were of higher improtance in deciding if a client would opt for Personal loan or not.
6. We established the importance of hyper-parameters/ pruning to reduce overfitting.The final model with post pruning had higher test recall 95%.

Recommendation

1. For customer segments Clients with higher income (above 140,000 dollars), ccavg (Over 3), and family member size of 2 and 3, with higher education (graduate level experience, 2) are more likely to opt for personal loan.
2. Age, ZipCodes, Experience, and Mortgage did not seem to have an impact in determining if a client would opt for personal loans or not.
3. Given the data set, there were more clients who did not take personal loan, given any of the other variables.
4. Having all other variables constant, 1 unit change in Income will increase the chances of taking a loan. CCAvg, Family size, Education, 1 unit change will bring about positive change and increase chances of taking a loan.Having a securities account will decrease the changes of taking a loan by 63%. Having an online account also decreases the chances of taking a loan by 44.5%.Owning a credit card also decreases the chance of taking a loan by 62%
5. From conduction the Decision tree, the model with the highest recall was considered as the best model. As recall should be used when you want to minimize False negatives, i.e. one wants at least positives should not be predicted as negatives.
6. The best model from decision tree was found with post-prunning with a total recall of 95%, this shows that the data is fit well, without any overfitting and that the model is a good overall fit.
7. Income was shown as the highest priority feature when determining if a client would opt for the personal loan or not.
8. It would be best to advertise towards higher income (above 140,000 dollars), ccavg (Over 3), and family member size of 2 and 3, with higher education (graduate level experience, 2) are more likely to opt for personal loan.
9. From the given data set, a liability customer was not likely to opt for the personal loan, but targeting the above segment of customers will definitely show a transition of customers to opt for personal loans.

In [ ]: