Case Study: AllLife bank using Logistic Regression | Decision Tree

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.
- The bank is interested in expanding this base (asset customers) rapidly to bring in more loan business and in the process, earn more through the interest on loans.
- 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.

Problem:

- The management wants to explore ways of converting its liability customers to personal loan customers (while retaining them as depositors).
- What are the independent variables affecting the convert of customer to Person Loan?
- Is There a segmente of customers, with some patten?

Objective:

Explore the dataset and extract insights from the data.

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.

Data Dictionary:

- ID: Customer ID
- Age: Customer's age in completed years
- Experience: #years of professional experience
- Income: Annual income of the customer (in thousand dollars)
- ZIP Code: Home Address ZIP code.
- Family: the Family size of the customer
- CCAvg: Average spending on credit cards per month (in thousand dollars)

- Education: Education Level. 1: Undergrad; 2: Graduate;3: Advanced/Professional
- Mortgage: Value of house mortgage if any. (in thousand dollars)
- Personal_Loan: Did this customer accept the personal loan offered in the last campaign?
- Securities_Account: Does the customer have securities account with the bank?
- CD_Account: Does the customer have a certificate of deposit (CD) account with the bank?
- Online: Do customers use internet banking facilities?
- CreditCard: Does the customer use a credit card issued by any other Bank (excluding All life Bank)?

Loading libraries

In [1]:
```
# this will help in making the Python code more structured automatically (goo
%load_ext nb_black
```

In [2]:
```
# silence unnecessary warnings
import warnings

warnings.filterwarnings("ignore")

# Libraries to help with read, manipulation and visualization data
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# To build sklearn model
import statsmodels.stats.api as sms
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

# Logistic Regression assumptions
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# To get diferent metric scores
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    roc_auc_score,
    roc_curve,
    confusion_matrix,
    precision_recall_curve,
)

# Removes the limit from the number of displayed columns and rows.
# This is so I can see the entire dataframe when I  print it
pd.set_option("display.max_columns", None)
```

```
# pd.set_option('display.max_rows', None)
pd.set_option("display.max_rows", 500)
```

Loading and exploring the data

Loading the data into python to explore and understand it.

In [3]:
```
# Load the data into pandas dataframe
df = pd.read_csv("Loan_Modelling.csv")
```

Overview of the data

In [4]:
```
print(
    f"There are \033[1;4m{df.shape[0]}\033[m rows and \033[1;4m{df.shape[1]}\
)  # f-string

# Look at 10 random rows
# Setting the random seed via np.random.seed to get the same random results e
np.random.seed(1)
df.sample(n=10)
```

There are **5000** rows and **14** columns.

Out[4]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Person
2764	2765	31	5	84	91320	1	2.9	3	105	
4767	4768	35	9	45	90639	3	0.9	1	101	
3814	3815	34	9	35	94304	3	1.3	1	0	
3499	3500	49	23	114	94550	1	0.3	1	286	
2735	2736	36	12	70	92131	3	2.6	2	165	
3922	3923	31	4	20	95616	4	1.5	2	0	
2701	2702	50	26	55	94305	1	1.6	2	0	
1179	1180	36	11	98	90291	3	1.2	3	0	
932	933	51	27	112	94720	3	1.8	2	0	
792	793	41	16	98	93117	1	4.0	3	0	

`ID.` is just an index for the data entry. In all likelihood, this column will not be a significant factor in determining if customer will by or not a personal loan. Therefore, we will not drop this variable just yet. Let us see if there is any relationship with the price when we do bivariate analysis.

`ZIPCode` Let us check how many unique values we have. If they are too many, we gonna need some processing to extract important information, binning and convert as categorical variable.

`Personal Loan` is our Target Variables and the variable which we need to predict.

In [5]:
```
# Analyzing the % that dependent variable is distributed on data set.
df["Personal_Loan"].value_counts() / len(df["Personal_Loan"])
```

Out[5]:
```
0    0.904
1    0.096
Name: Personal_Loan, dtype: float64
```

Observation:

We have an imbalanced data set, with 90.5% of customers as Won't by a personal loan and only 9.5% that Will by a personal loan.

Checking for duplicates in the data.

In [6]:
```
df[df.duplicated()].count()
```

Out[6]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIPCode               0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal_Loan         0
Securities_Account    0
CD_Account            0
Online                0
CreditCard            0
dtype: int64
```

- There are no duplicated values in the data that needs to be removed.

Check the data types of the columns for the dataset.

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIPCode             5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64  
 8   Mortgage            5000 non-null   int64  
 9   Personal_Loan       5000 non-null   int64  
 10  Securities_Account  5000 non-null   int64  
 11  CD_Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

Observations:

- Personal_Loan is the dependent variable - type integer.

- As expected, `ZIPCode` is numerical when it should ideally be categorical. To be able to get summary statistics for this column, We will have to process it first.
- Family, Education, Securities_Account, CD_Account, Online and CreditCard are ideally categorical variables.

Processing Columns

1. ZIP Code

It is a numeric value but it's also essentially a category. We gonna treat values mapping zip codes to different locations.

In [8]:
```
# Checking how many unique values on column ZIPCode
print(
    f"There are \033[1;4m{df['ZIPCode'].nunique()}\033[m unique values of Zip
)
```

```
There are 467 unique values of Zip Code on the data set.
```

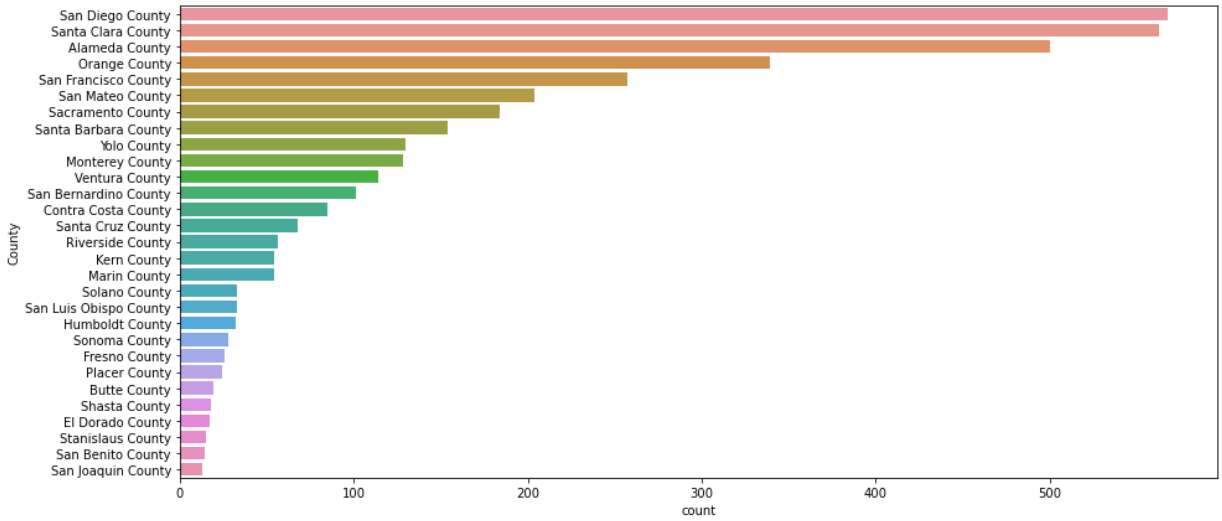
In [9]:
```
# Import librarie to get City, County and State value for any zipcode.
from uszipcode import SearchEngine

search = SearchEngine()

# Create a new column for corresponding city,county,state by zip code.
city = []
county = []
state = []

for zip in df["ZIPCode"]:
    zipcode = search.by_zipcode(zip)
    city.append(zipcode.major_city)
    county.append(zipcode.county)
    state.append(zipcode.state)


# Adding column to the dataset, with new list of values.
df["City"] = city
df["County"] = county
df["State"] = state
```

In [10]:
```
print(
    f"There are:\n{df['City'].nunique()} unique city; \n {df['County'].nuniqu
    f"\n  {df['State'].nunique()} unique state."
)
```

```
There are:
244 unique city;
 38 unique county;
  1 unique state.
```

In [11]:
```
plt.figure(figsize=(15, 7))
sns.countplot(y="County", data=df, order=df["County"].value_counts().index[1:
```

Out[11]:
```
<AxesSubplot:xlabel='count', ylabel='County'>
```



Observation:

It still a lot of unique values for county, lets try group it by North and South.

Check for missing values

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIPCode               0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal_Loan         0
Securities_Account    0
CD_Account            0
Online                0
CreditCard            0
City                 34
County               34
State                34
dtype: int64
```

- There are 34 missing values in the data, after applying ZipCode

In [13]:
```
# Let us check the null values in our dataset.
df[df.City.isnull()].head(5)
```

Out[13]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_
106	107	43	17	69	92717	4	2.9	1	0	
172	173	38	13	171	92717	2	7.8	1	0	
184	185	52	26	63	92717	2	1.5	2	0	
321	322	44	20	101	92717	3	4.4	2	82	
366	367	50	24	35	92717	1	0.3	3	0	

In [14]:
```
# Replacing null values with 'Unknown'
df.County = df.County.fillna("Unknown")
```

In [15]:
```
# Creating a new dataframe with only the 2 columns that we are going to analy
df_none = df.filter(["County", "ZIPCode"], axis=1)
```

In [16]:
```
# Lets check Age by experience greater or equal to 0
df_none[df_none["County"] == "Unknown"].sort_values("ZIPCode").value_counts()
```

Out[16]:
```
County   ZIPCode
Unknown  92717      22
         96651       6
         92634       5
         93077       1
dtype: int64
```

Observations:

- Checking the ZIPCode on Google to see where does it belongs to and seems to be all out EUA;
- It seems to be a mistake, we gonna keep it as `Unknown` ,

In [17]:
```
# Create a list of Southern California county .
Southern_Ca = [
    "Kings County",
    "Tulare County",
    "Inyo County",
    "Imperial County",
    "Los Angeles County",
    "Orange County",
    "Riverside County",
    "San Bernardino County",
    "San Diego County",
    "Santa Barbara County",
    "Ventura County",
    "Kern County",
    "San County",
    "Luis Obispo County",
]

# Create a new column for corresponding north or south.
north_south = []

for county in df["County"]:
    if county in Southern_Ca:
        north_south.append("South")
    elif county == "Unknown":
        north_south.append("Unknown")
    else:
        north_south.append("North")

# Adding column to the dataset, with new list of values.
df["north_south"] = north_south
```

In [18]:
```
# Grouping by county and north-south to make sure values are correct.
df.groupby(by=["County", "north_south"]).size().sort_values(ascending=False)
```

Out[18]:
```
County                 north_south
Los Angeles County     South          1095
San Diego County       South           568
Santa Clara County     North           563
Alameda County         North           500
Orange County          South           339
San Francisco County   North           257
San Mateo County       North           204
Sacramento County      North           184
Santa Barbara County   South           154
Yolo County            North           130
Monterey County        North           128
Ventura County         South           114
San Bernardino County  South           101
Contra Costa County    North            85
Santa Cruz County      North            68
Riverside County       South            56
Kern County            South            54
Marin County           North            54
Unknown                Unknown          34
San Luis Obispo County North            33
Solano County          North            33
Humboldt County        North            32
Sonoma County          North            28
Fresno County          North            26
Placer County          North            24
Butte County           North            19
Shasta County          North            18
El Dorado County       North            17
Stanislaus County      North            15
San Benito County      North            14
San Joaquin County     North            13
Mendocino County       North             8
Tuolumne County        North             7
Siskiyou County        North             7
Lake County            North             4
Merced County          North             4
Trinity County         North             4
Imperial County        South             3
Napa County            North             3
dtype: int64
```

In [19]:
```
df["north_south"].value_counts()
```

Out[19]:
```
South      2484
North      2482
Unknown      34
Name: north_south, dtype: int64
```

In [20]:
```
df.groupby(by=["Personal_Loan", "north_south"]).size().sort_values(ascending=
```

Out[20]:
```
Personal_Loan  north_south
0              South          2246
               North          2243
1              North           239
               South           238
0              Unknown          31
1              Unknown           3
dtype: int64
```

Observation:

1. Customers seems to be balanced distribuited between North and South of California, meaning that it not gonna be a significante variable

In [21]:
```
# To get the same random results of 10 sample of data every time
np.random.seed(1)
df.sample(n=10)
```

Out[21]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Person
2764	2765	31	5	84	91320	1	2.9	3	105	
4767	4768	35	9	45	90639	3	0.9	1	101	
3814	3815	34	9	35	94304	3	1.3	1	0	
3499	3500	49	23	114	94550	1	0.3	1	286	
2735	2736	36	12	70	92131	3	2.6	2	165	
3922	3923	31	4	20	95616	4	1.5	2	0	
2701	2702	50	26	55	94305	1	1.6	2	0	
1179	1180	36	11	98	90291	3	1.2	3	0	
932	933	51	27	112	94720	3	1.8	2	0	
792	793	41	16	98	93117	1	4.0	3	0	

In [22]:
```
# Let's check data information, non-null values and Dtype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 18 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIPCode             5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64  
 8   Mortgage            5000 non-null   int64  
 9   Personal_Loan       5000 non-null   int64  
 10  Securities_Account  5000 non-null   int64  
 11  CD_Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
 14  City                4966 non-null   object 
 15  County              5000 non-null   object 
 16  State               4966 non-null   object 
```

```
 17  north_south         5000 non-null   object
dtypes: float64(1), int64(13), object(4)
memory usage: 703.2+ KB
```

In [23]:
```
# Converting north_south to categorical
df["north_south"] = df["north_south"].astype("category")
```

In [24]:
```
# Drop the redundant columns.
df.drop(columns=["ZIPCode", "City", "County", "State"], inplace=True)
```

In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   ID                  5000 non-null   int64   
 1   Age                 5000 non-null   int64   
 2   Experience          5000 non-null   int64   
 3   Income              5000 non-null   int64   
 4   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   
 13  north_south         5000 non-null   category
dtypes: category(1), float64(1), int64(12)
memory usage: 512.9 KB
```

- There are NO missing values in the data.

Exploratory Data Analysis

Give a statistical summary for the dataset.

In [26]:
```
# Basic summary stats - Numeric variables
df.describe().T
```

Out[26]:

	count	mean	std	min	25%	50%	75%	max
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750.25	5000.0
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55.00	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30.00	43.0
Income	5000.0	73.774200	46.033729	8.0	39.00	64.0	98.00	224.0
Family	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.70	1.5	2.50	10.0
Education	5000.0	1.881000	0.839869	1.0	1.00	2.0	3.00	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101.00	635.0
Personal_Loan	5000.0	0.096000	0.294621	0.0	0.00	0.0	0.00	1.0

	count	mean	std	min	25%	50%	75%	max
Securities_Account	5000.0	0.104400	0.305809	0.0	0.00	0.0	0.00	1.0
CD_Account	5000.0	0.060400	0.238250	0.0	0.00	0.0	0.00	1.0
Online	5000.0	0.596800	0.490589	0.0	0.00	1.0	1.00	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1.00	1.0

In [27]:
```
# Basic summary stats - Numeric variables
df.describe(include=["category"])
```

Out[27]:

	north_south
count	5000
unique	3
top	South
freq	2484

`Age` values in a high range. We should check a few of the extreme values to get a sense of the data.

`Experience` min and max values warrant a quick check, also there is negative values that we need to treat it on Data cleaning.

`Income` min is to low for a year income, we need to check it and also max seems to be to high.

`north_south` seems to be balanced.

`Family`, `Education` are Ordinal Categorical we gonna keep it as numerical and procide with `Label Encoding` considering that there is a sense of order on the values.

`Securities_Account`, `CD_Account`, `Online`, `CreditCard` are Binary Categorical, we gonna keep it as numerical and procide with `Label Encoding`.

Data Cleaning

1. Experience by Age

We gonna check the min and max values of `Experience` by `Age`

In [28]:
```
# Creating a new dataframe with only the 3 columns that we are going to analy
df_temp = df.filter(["Age", "Experience", "Income"], axis=1)
```

In [29]:
```
# Lets check Age by experience
df_temp.sort_values("Age")
```

Out[29]:

	Age	Experience	Income
3157	23	-1	13

	Age	Experience	Income
4285	23	-3	149
2618	23	-3	55
4411	23	-2	75
2430	23	-1	73
...	...	...	...
4360	67	43	41
3886	67	43	79
2846	67	43	105
1859	67	41	20
4172	67	42	75

5000 rows × 3 columns

- Min and Max Age makes sense.

In [30]:
```
# Lets check Age by experience greater or equal to 0
df_temp[df_temp["Experience"] > -1].sort_values("Age")
```

Out[30]:

	Age	Experience	Income
155	24	0	60
182	24	0	135
2652	24	0	44
4566	24	0	131
4393	24	0	59
...	...	...	...
4468	67	42	51
3703	67	41	78
4451	67	41	18
3331	67	42	21
3886	67	43	79

4948 rows × 3 columns

Observation:

1. Some customers has negative experience time, which does not make sense, we gonna need to treat it as `missing values` .
2. Young customers has less experience than older customers, which makes sense.

In [31]:
```
# Cheking the number of customers by negative time of Experience
exp_temp = df_temp[df_temp["Experience"] < 0]
exp_temp["Experience"].value_counts()
```

Out[31]:
```
-1    33
-2    15
-3     4
Name: Experience, dtype: int64
```

In [32]:
```
# Cheking the number of customers by negative time of Experience per Age
exp_temp.groupby(by=["Experience", "Age"]).size()
```

Out[32]:
```
Experience  Age
-3          23      2
            24      2
-2          23      4
            24      9
            25      1
            28      1
-1          23      6
            24      6
            25     17
            26      1
            29      3
dtype: int64
```

In [33]:
```
# Creating a new dataframe with only customers with positive experience time
exp2_temp = df_temp[(df_temp["Experience"] > -1) & (df_temp["Age"] < 31)]
exp2_temp.groupby(["Age"])["Experience"].mean().sort_values(
    ascending=True
)  # Filtering Eexperience mean per Age.
```

Out[33]:
```
Age
24    0.000000
25    0.514286
26    0.987013
27    1.923077
28    3.009804
29    3.833333
30    4.860294
Name: Experience, dtype: float64
```

Observation:

1. 12 Customers with `Age 23`, has negative experience time, lets replace it with 0 Experience.
2. 17 Customers with `Age 24`, has negative experience time, and mean experience is 0, lets replace it with 0 Experience.
3. 18 Customers with `Age 25`, has negative experience time, and mean experience is less than 1, lets replace it with 0 Experience.
4. 1 and 3 Customers with `Age26 and 29`, has negative experience time, and mean experience is less 1 and less 4 respectively, lets also replace it with 0 once the number of customer is not expressive.

In [34]:
```
# Replacing experience less than 0 by 0
df.Experience = df.Experience.apply(lambda x: 0 if x < 0 else x)
```

In [35]:
```
df.Experience.describe().T
```

Out[35]:
```
count    5000.000000
mean       20.119600
std        11.440484
min         0.000000
```

```
25%        10.000000
50%        20.000000
75%        30.000000
max        43.000000
Name: Experience, dtype: float64
```

- Min of Experience is 0, all negative values fixed.

2. Income

We gonna check the min and max values of `Income` by `Age` and `Experience`

In [36]:
```
# Lets check Income by experience
df_temp.sort_values("Income")
```

Out[36]:

	Age	Experience	Income
1054	63	38	8
1197	48	23	8
939	56	32	8
2591	31	7	8
2905	64	40	8
...	...	...	...
677	46	21	204
526	26	2	205
2988	46	21	205
4993	45	21	218
3896	48	24	224

5000 rows × 3 columns

In [37]:
```
# Cheking the number of customers by Income less than $12 thousand per year
inc_temp = df_temp[df_temp["Income"] < 12]
inc_temp["Income"].value_counts()
```

Out[37]:
```
11    27
9     26
10    23
8     23
Name: Income, dtype: int64
```

In [38]:
```
# Cheking the number of customers by Income less than $12 thousand per year p
inc_temp.groupby(by=["Income", "Experience"]).size()
```

Out[38]:
```
Income  Experience
8       1             1
        7             1
        8             2
        9             1
        11            1
        15            1
        16            1
        19            1
        23            3
        25            1
```

	26	1
	28	1
	32	1
	35	2
	36	1
	37	1
	38	1
	39	1
	40	1
9	2	1
	4	1
	11	4
	13	1
	14	1
	16	1
	17	2
	18	1
	20	1
	21	1
	23	2
	24	1
	25	1
	28	1
	31	1
	32	1
	33	2
	37	2
	38	1
10	2	1
	3	1
	7	2
	8	1
	9	1
	12	1
	14	1
	15	1
	16	1
	18	2
	23	1
	27	1
	29	1
	33	1
	34	1
	36	3
	37	2
	39	1
11	2	1
	3	1
	4	1
	5	1
	6	1
	7	1
	8	1
	11	1
	15	1
	16	1
	17	1
	19	1
	21	1
	22	1
	23	1
	24	1
	26	2
	28	1
	31	1
	32	1
	34	2
	35	1
	37	2

```
        41           1
dtype: int64
```

Observation:

1. There are all king of customers (experience time / age) with low income.
2. Could be an error, or customer who did not work the whole year or have part-time job.
3. There are no pattern, to explain it.

In [39]:
```
# Cheking the number of customers by Income greather than $185 thousand per y
inc_temp = df_temp[df_temp["Income"] > 185]
inc_temp["Income"].value_counts()
```

Out[39]:
```
195    15
191    13
190    11
188    10
194     8
192     6
193     6
201     5
198     3
199     3
200     3
204     3
202     2
203     2
205     2
189     2
218     1
224     1
Name: Income, dtype: int64
```

In [40]:
```
inc_temp["Income"].value_counts().sum()
```

Out[40]:
```
96
```

In [41]:
```
# Cheking the number of customers by Income greather than $185 thousand per y
inc_temp.groupby(by=["Income", "Experience"]).size()
```

Out[41]:
```
Income  Experience
188     2             1
        5             1
        7             1
        8             1
        9             1
        11            1
        12            1
        24            1
        27            1
        36            1
189     18            1
        24            1
190     1             1
        3             2
        11            2
        12            1
        22            1
        23            1
        25            1
        26            1
        32            1
191     4             1
```

```
             6              1
             9              1
             13             1
             16             1
             20             1
             23             1
             26             2
             32             2
             35             1
             37             1
192          7              1
             25             1
             26             1
             27             1
             28             1
             31             1
193          3              1
             6              1
             11             1
             12             1
             21             1
             36             1
194          4              1
             11             1
             12             1
             20             1
             23             1
             26             2
             30             1
195          0              1
             2              2
             5              1
             9              1
             11             1
             13             1
             24             1
             25             1
             27             2
             30             1
             36             1
             38             1
             41             1
198          9              1
             20             1
             22             1
199          4              1
             20             1
             23             1
200          8              1
             10             1
             20             1
201          3              1
             16             1
             19             2
             20             1
202          15             1
             35             1
203          5              1
             22             1
204          4              1
             18             1
             21             1
205          2              1
             21             1
218          21             1
224          24             1
dtype: int64
```

In [42]:
```
# Cheking the number of customers by Income greather than $185 thousand per y
inc_temp = df_temp[(df_temp["Income"] > 185) & (df_temp["Experience"] < 6)]
inc_temp.groupby(by=["Income", "Experience"]).size()
```

Out[42]:
```
Income  Experience
188     2             1
        5             1
190     1             1
        3             2
191     4             1
193     3             1
194     4             1
195     0             1
        2             2
        5             1
199     4             1
201     3             1
203     5             1
204     4             1
205     2             1
dtype: int64
```

In [43]:
```
inc_temp["Income"].value_counts().sum()
```

Out[43]:
```
17
```

Observation:

1. It is unlikely that a person with less than 6 years of experience have an income greater than 185 thousand per year, so we gonna drop this 17 customers (rows).
2. Customers with more than 5 years experience we can consider that the have a income greater than 185 thousand per year and we gonna treat it as real information (NO outlier treatment).

In [44]:
```
# delete all rows with column 'Income' has value >185 and column 'Experience'
indexNames = df[(df["Income"] > 185) & (df["Experience"] < 6)].index
df.drop(indexNames, inplace=True)
```

In [45]:
```
# checking if the 17 custumers were dropped.
temp = df[(df["Income"] > 185) & (df["Experience"] < 6)]
temp.head()
```

Out[45]:

ID	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal_Loan	Securitie

In [46]:
```
# checking if the 17 custumers were dropped.
df.shape
```

Out[46]:
```
(4983, 14)
```

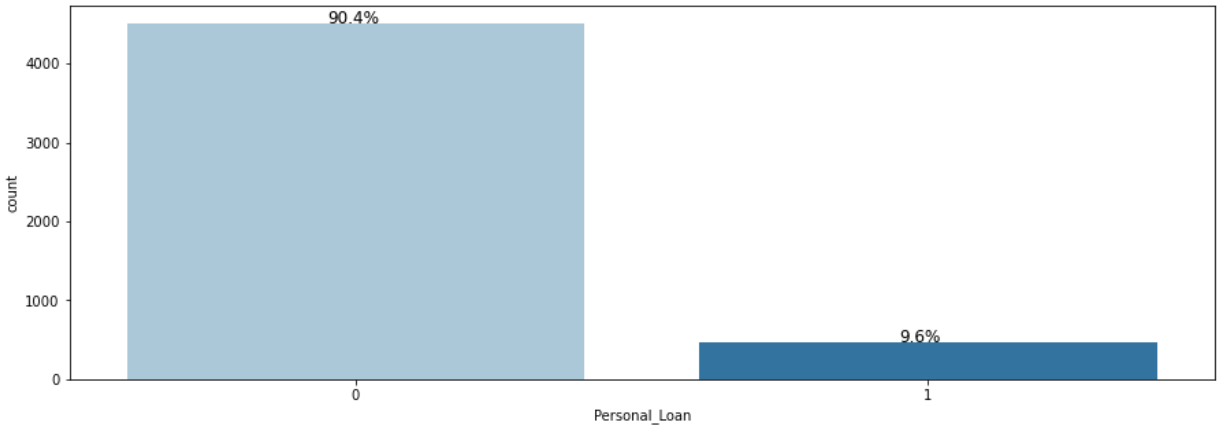
In [47]:
```
# Deliting all temp data frames created
df_temp, exp_temp, exp2_temp, inc_temp, temp = (
    pd.DataFrame(),
    pd.DataFrame(),
    pd.DataFrame(),
    pd.DataFrame(),
```

```
    pd.DataFrame(),
)
lst = [df_temp, exp_temp, exp2_temp, inc_temp, temp]
del lst  # memory release now
```

Data Visualization: Univariate Analysis

In [48]:
```
# Function to create barplots that indicate percentage for each category.


def perc_on_bar(z):
    """
    plot
    feature: categorical feature
    the function won't work if a column is passed in hue parameter
    """

    total = len(df[z])  # length of the column
    plt.figure(figsize=(15, 5))
    # plt.xticks(rotation=45)
    ax = sns.countplot(df[z], palette="Paired")
    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
    plt.show()  # show the plot
```

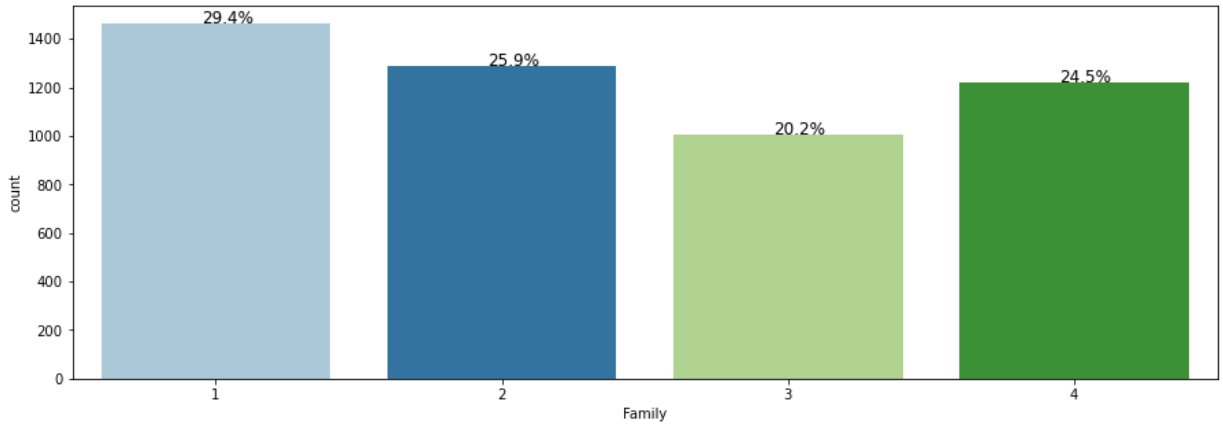
In [49]:
```
perc_on_bar("Personal_Loan")
```



Obsrvation:

- It's a imbalanced data where only 9.6% of customers said yes for a Personal Loan.

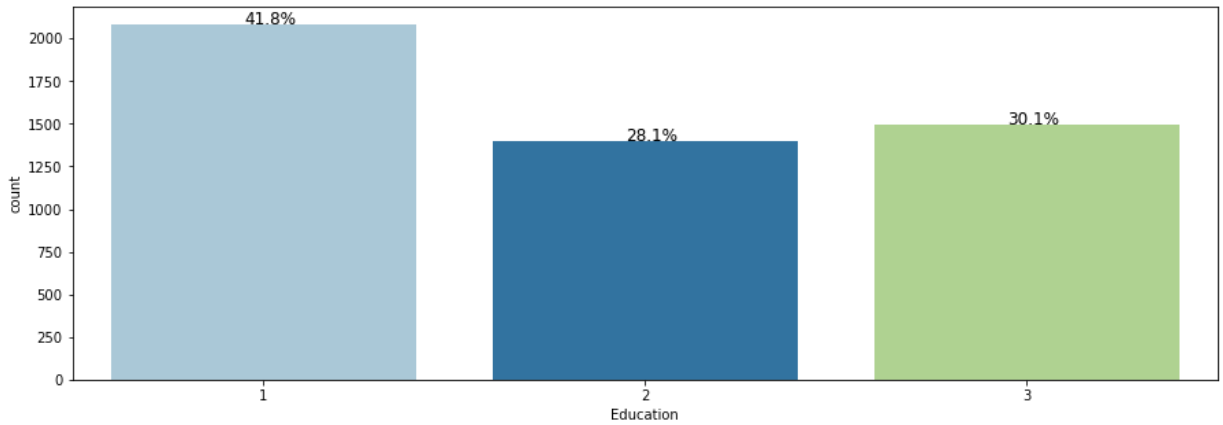
In [50]:
```
perc_on_bar("Family")
```



Obsrvation:

- Data is almost equal distribuited between the Family size.
- The highest value is 29.4% that belongs to family size One.
- The lowest value is 20.2% for family size Three.

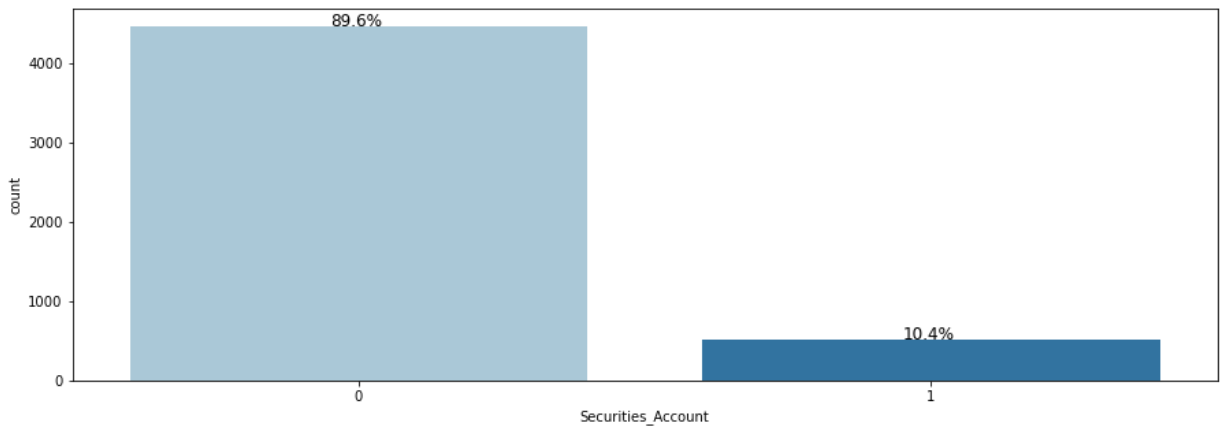
In [51]:
```
perc_on_bar("Education")
```



Obsrvation:

- Undergraduated customers are more than Graduate and Advanced/Professional.

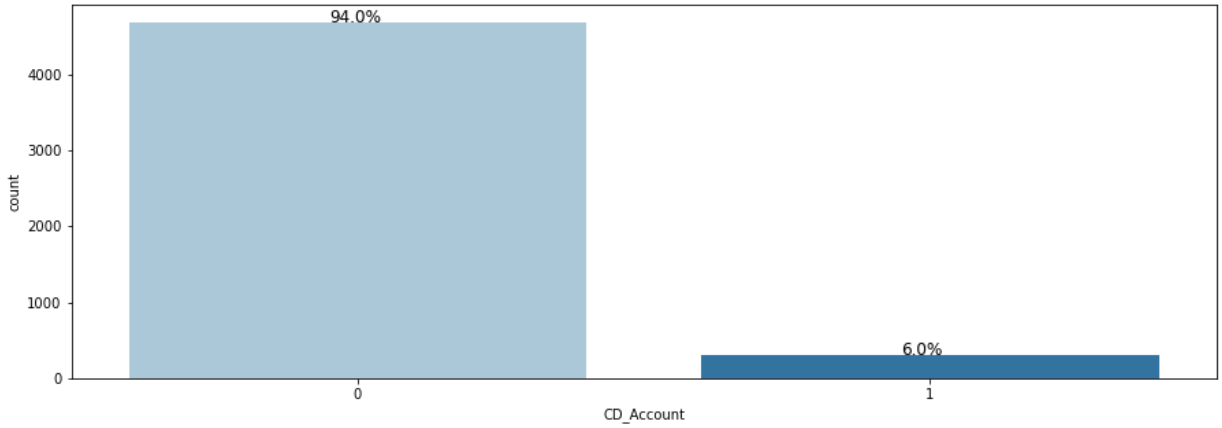
In [52]:
```
perc_on_bar("Securities_Account")
```



Obsrvation:

- Only 10.4% of customers have securities account with the bank.

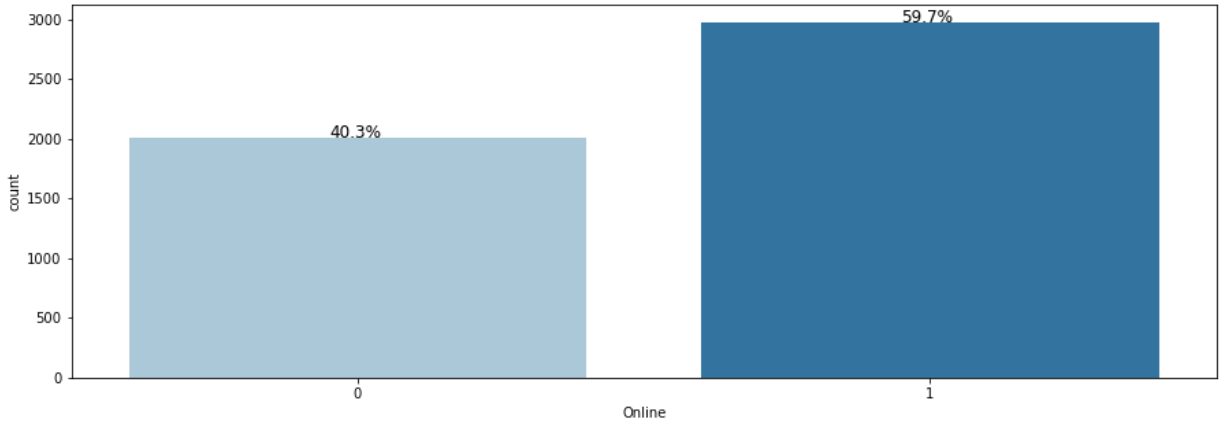
In [53]:
```
perc_on_bar("CD_Account")
```



Observation:

- 94% of customers does not have certificate of deposit (CD) account with the bank.

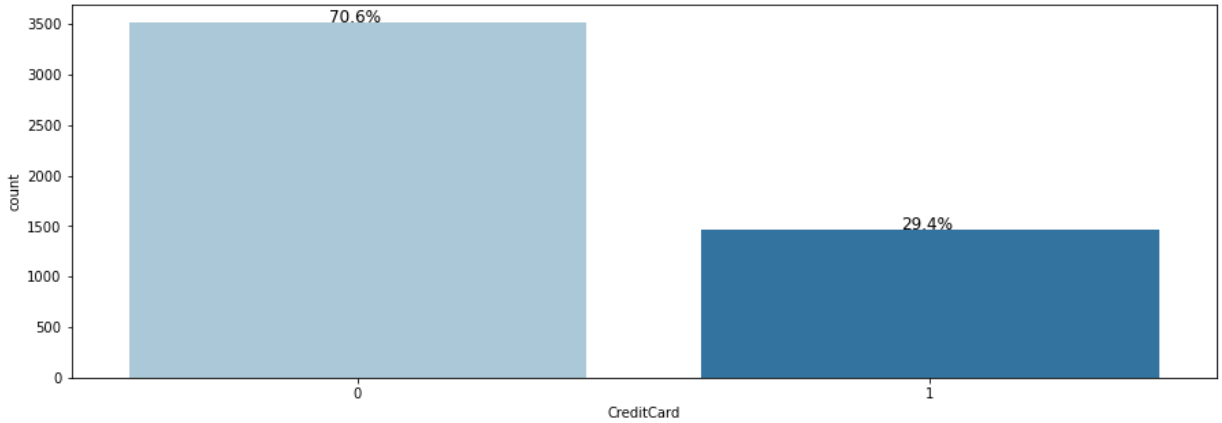
In [54]:
```
perc_on_bar("Online")
```



Observation:

- 59.7% of customers use internet banking facilities.
- It's a good opportunity to use this channel of communication.

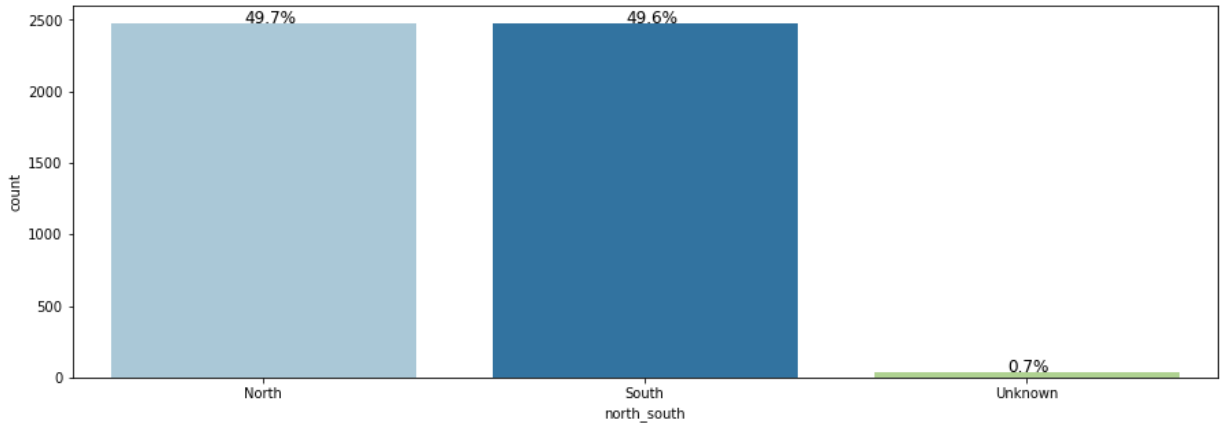
In [55]:
```
perc_on_bar("CreditCard")
```



Observations:

- 70.6% of customers does not use a credit card issued by any other Bank

In [56]:
```
perc_on_bar("north_south")
```

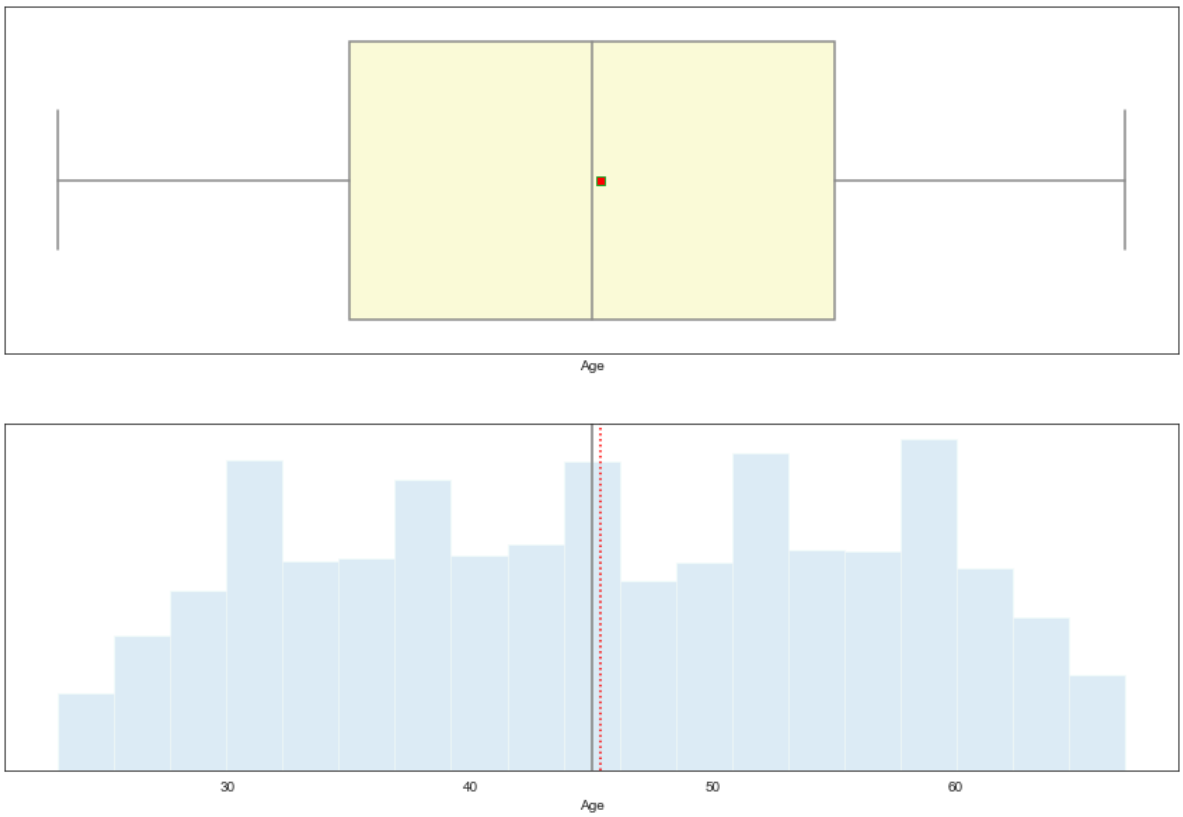


Observations:

- 50.4% of customers lives on North California and 49.6% on South California

In [57]:
```
# A function to create boxplot and histogram for any input numerical


def hist_boxplot(dataframe, figsize=(15, 10), bins=None):
    """
    This function takes the numerical column as the input and returns the box
    dataframe: 1-d feature array
    figsize: size of fig (default (13,8))
    bins: number of bins (default None / auto)
    """

    # Figure aesthetics
    sns.set_style("white")

    # Creating the 2 subplots
    fig, (ax_box, ax_hist) = plt.subplots(nrows=2, sharex=True, figsize=figsi

    # Boxplot will be created and a red square will indicate the mean value o
    sns.boxplot(
        dataframe,
        ax=ax_box,
        showmeans=True,
        meanprops={"marker": "s", "markerfacecolor": "red"},
        color="xkcd:eggshell",
    )

    # For histogram
    sns.distplot(
        dataframe, kde=F, ax=ax_hist, color="lightblue", bins=bins
    ) if bins else sns.distplot(dataframe, kde=False, ax=ax_hist, color="ligh

    # Add mean to the histogram
    ax_hist.axvline(np.mean(dataframe), color="r", linestyle="dotted")
    # Add median to the histogram
    ax_hist.axvline(np.median(dataframe), color="gray", linestyle="solid")
```

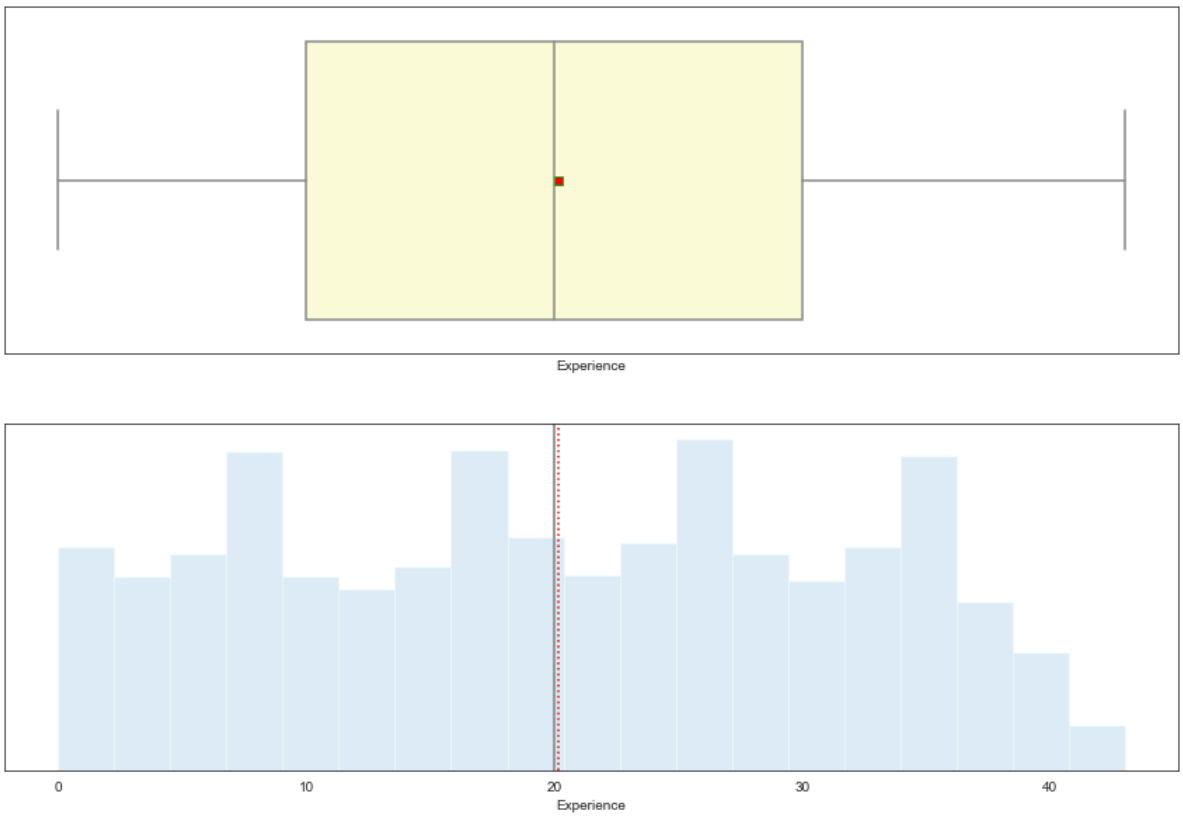
In [58]:
```
hist_boxplot(df.Age)
```



Observations:

- Customers age mean and median is around 45 years old.
- Youngest customers have 23 years old and the olders 67.
- There is no outliers and a good frequence distribution around 25 to 65 years old.

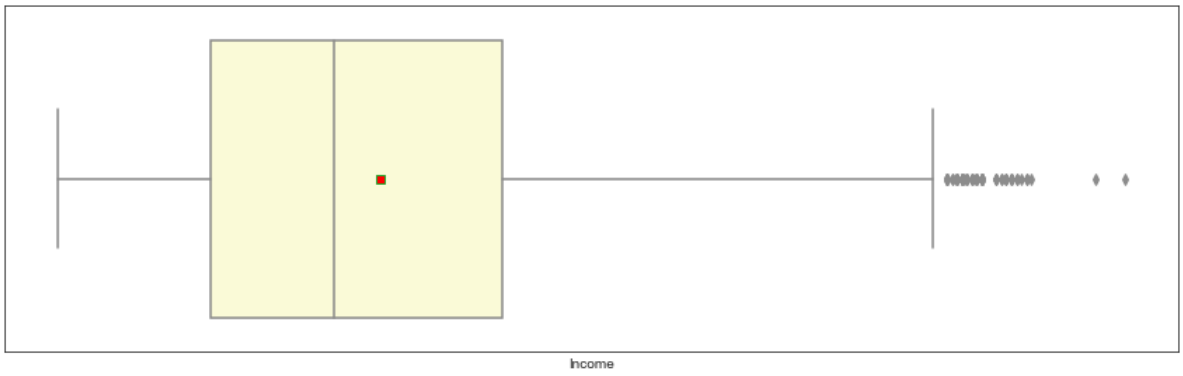
In [59]:
```
hist_boxplot(df.Experience)
```

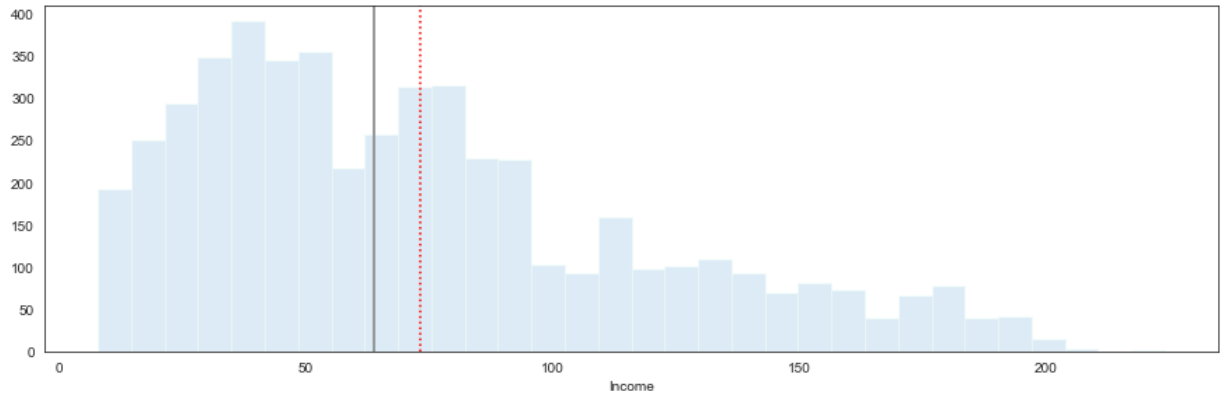


Observations:

- Customers Experience mean and median is around 20 years.
- Min of 0 and max of 43 years.
- There is no outliers and a good frequence distribution around experience.

In [60]:
```
hist_boxplot(df.Income)
```

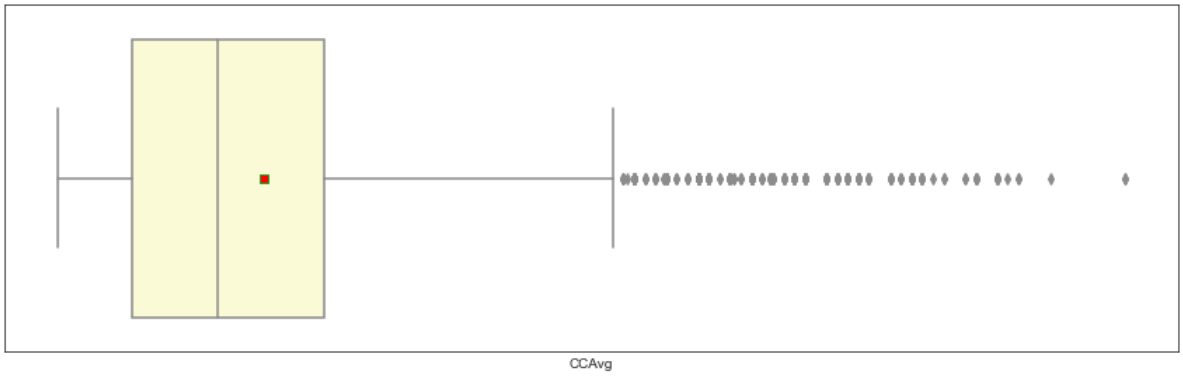


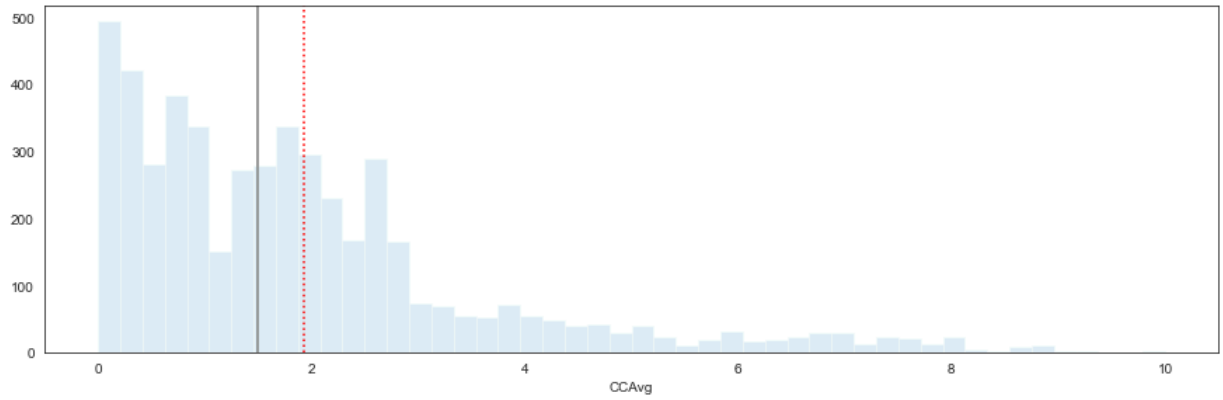


Observations:

- There are few customers with Income greather than $175 thousand/year, but after analysing data it seems to be real data, not an outlier.
- Mim and Max Income around 8 and 224thousand/year
- Income is right skewed

In [61]:
```
hist_boxplot(df.CCAvg)
```

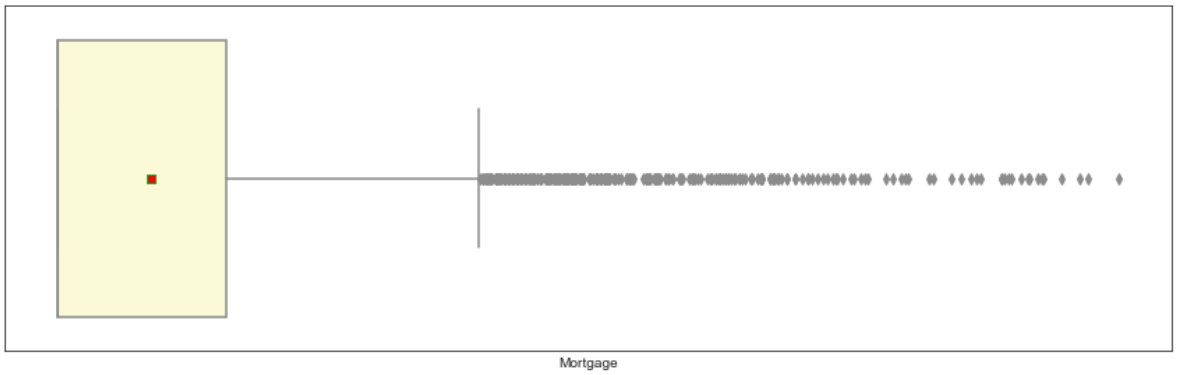


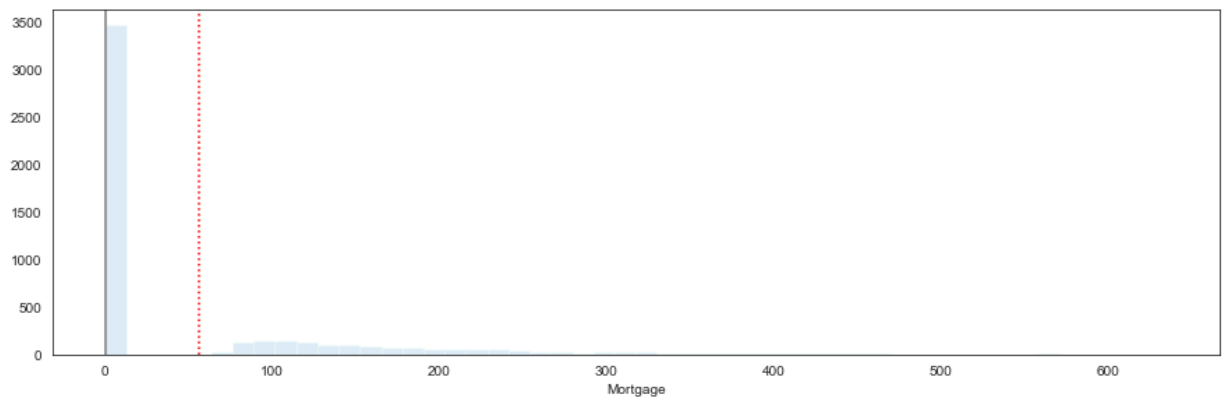


Observations:

- The mean Average spending on credit cards per month (in thousand dollars) is 1.92
- Max is $10 thousand/month which makes sense if it belongs to high income customers (we'll check on multivariate graphs).
- It's right skewed.

In [62]:
```
hist_boxplot(df.Mortgage)
```





In [63]:
```
zero = df[df['Mortgage'] == 0]
zero['Mortgage'].value_counts()
```

Out[63]:
```
0    3450
Name: Mortgage, dtype: int64
```

Observations:

- It's right skewed with mean of 56 thousand dollars and median of 0 (ZZERO)
- 69% of customers does not have a Mortgage
- We'll check on multivariate graphs, if higher Income its correlated with High Mortgage.

Bivariate analysis

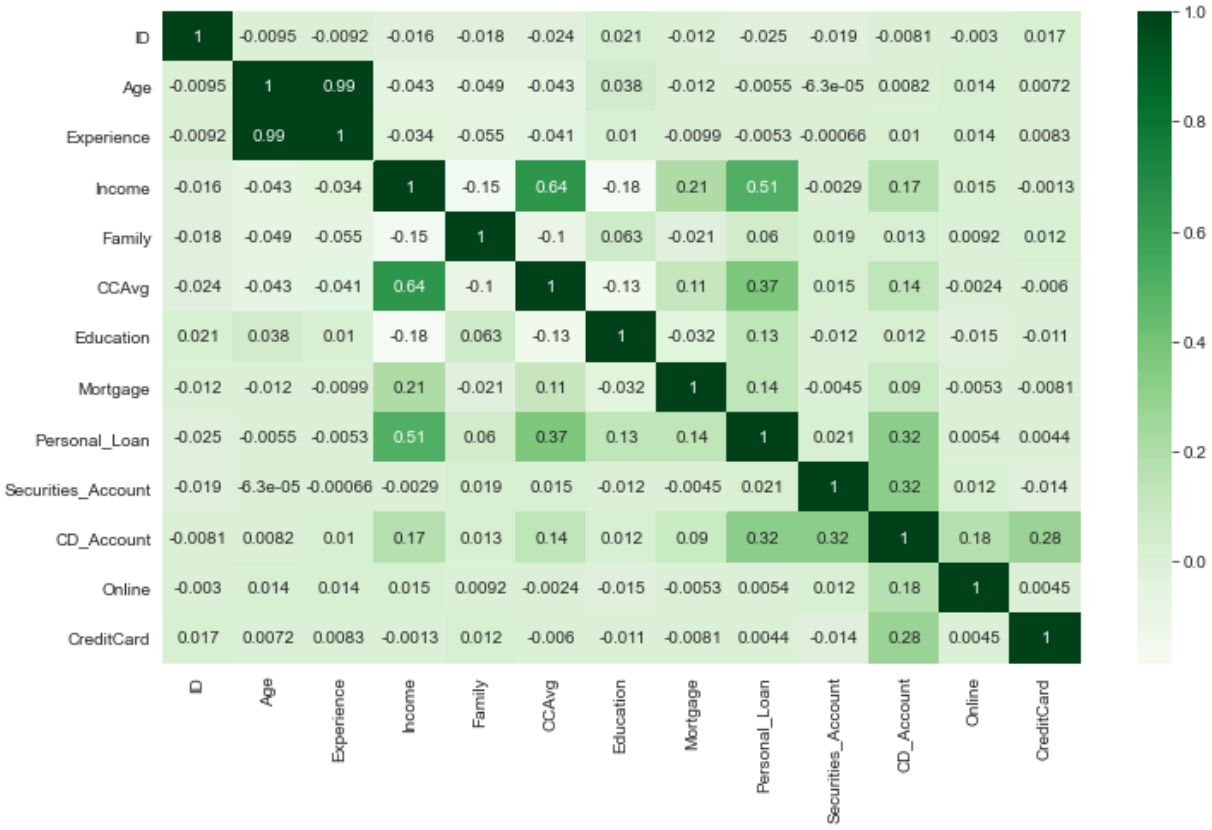
Correlation between numeric Variables

In [64]:
```
df.corr()["Personal_Loan"].sort_values(ascending=False)
```

Out[64]:
```
Personal_Loan         1.000000
Income                0.506201
CCAvg                 0.368107
CD_Account            0.315945
Mortgage              0.143512
Education             0.134795
Family                0.060250
Securities_Account    0.020829
Online                0.005441
CreditCard            0.004436
Experience           -0.005301
Age                  -0.005471
ID                   -0.025070
Name: Personal_Loan, dtype: float64
```

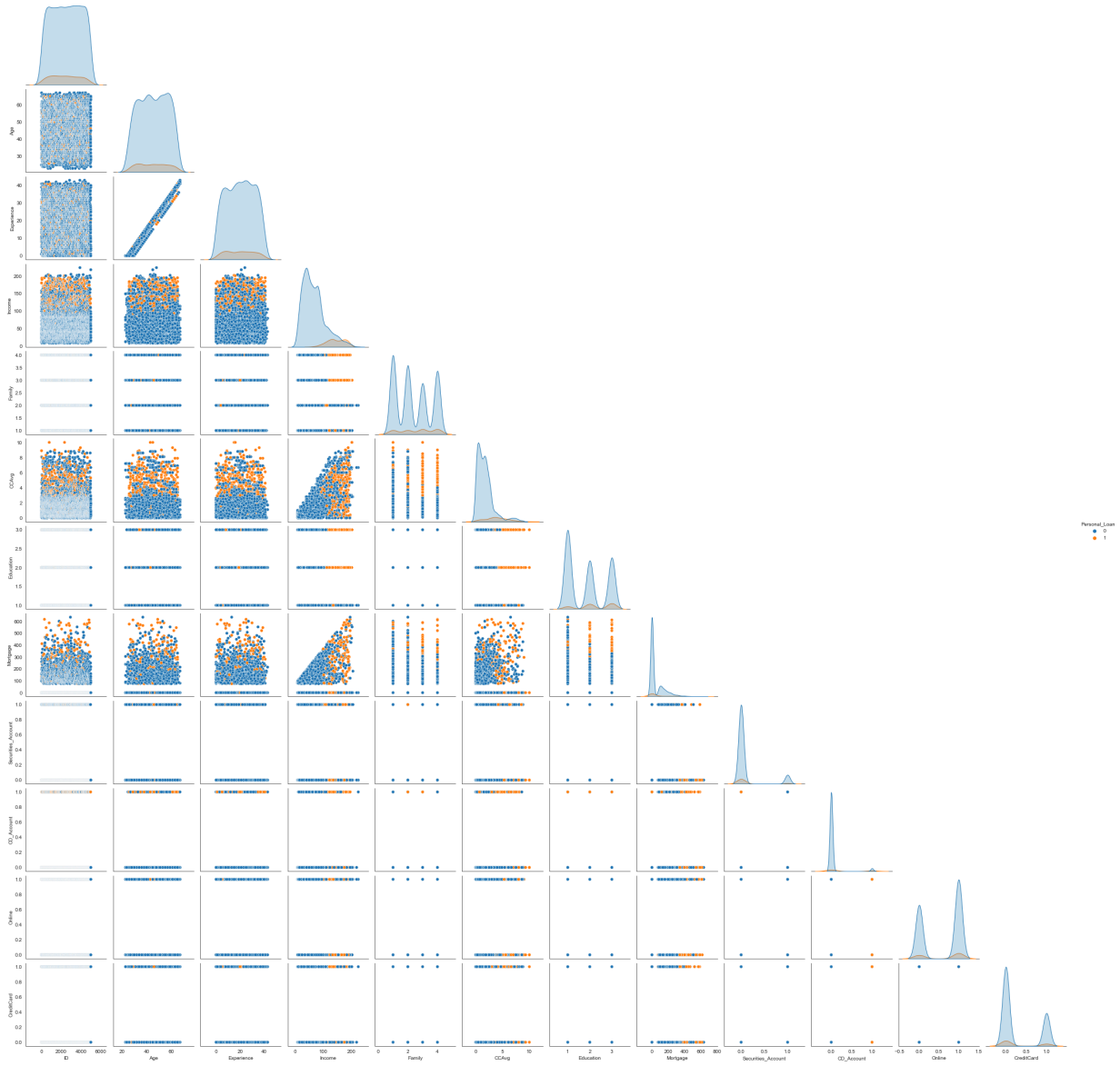
In [65]:
```
plt.figure(figsize=(12, 7))
sns.heatmap(df.corr(), annot=True, cmap="Greens")
```

Out[65]:
```
<AxesSubplot:>
```



- `Experience` shows the highest correlation with `Age` (0.99) which was expected.
- `CCAvg` has a 0.64 correlation with `Income` .
- `Persolnal_Loan` has 0.51 correlation with `income` and 0.37 with `CCAvg`
- There is no relationship with `Id` and `Personal Loan` .

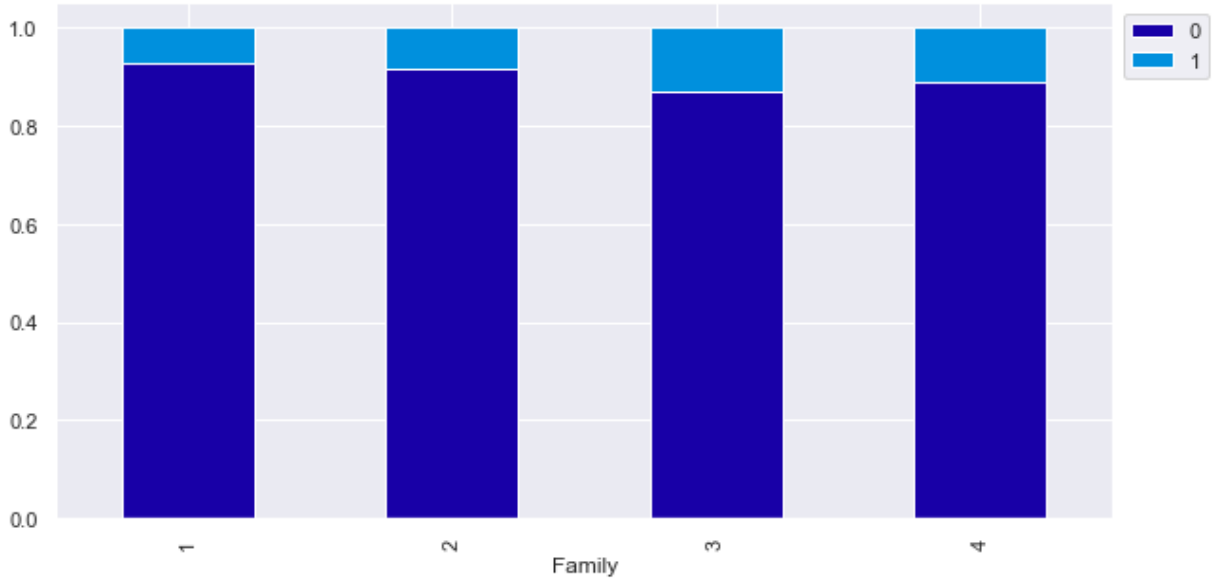
In [66]:
```
sns.pairplot(data=df, hue="Personal_Loan", corner=True)
plt.show()
```



In [67]:
```
def stacked_plot(x):
    sns.set(palette="nipy_spectral")
    tab1 = pd.crosstab(x, df["Personal_Loan"], margins=True)
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(x, df["Personal_Loan"], normalize="index")
    tab.plot(kind="bar", stacked=True, figsize=(10, 5))
    plt.legend(loc="lower left", frameon=False)
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [68]:
```
stacked_plot(df["Family"])
```

```
Personal_Loan     0    1   All
Family
1              1358  106  1464
2              1184  106  1290
3               877  131  1008
4              1088  133  1221
All            4507  476  4983
------------------------------------------------------------------------------------------
------------------------------------------
```

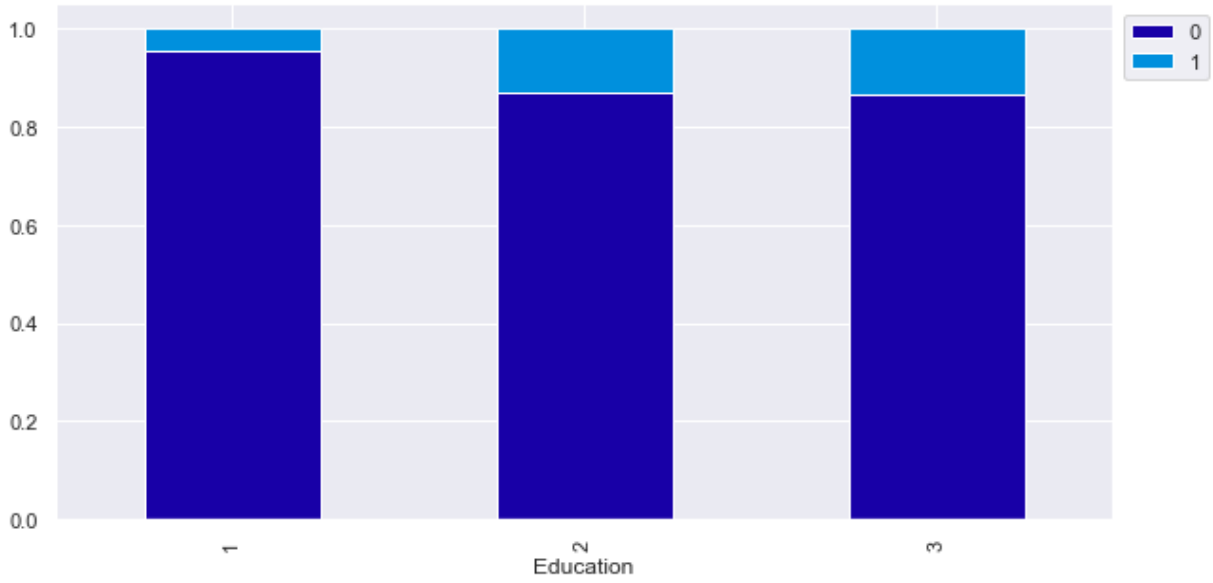


Observation:

- Family size of 3 is more likehood to accept a Personal Loan, followed by size 4.

In [69]:
```
stacked_plot(df["Education"])
```

```
Personal_Loan     0    1   All
Education
1              1990   93  2083
2              1221  180  1401
3              1296  203  1499
All            4507  476  4983
------------------------------------------------------------------------------
------------------------------------------
```



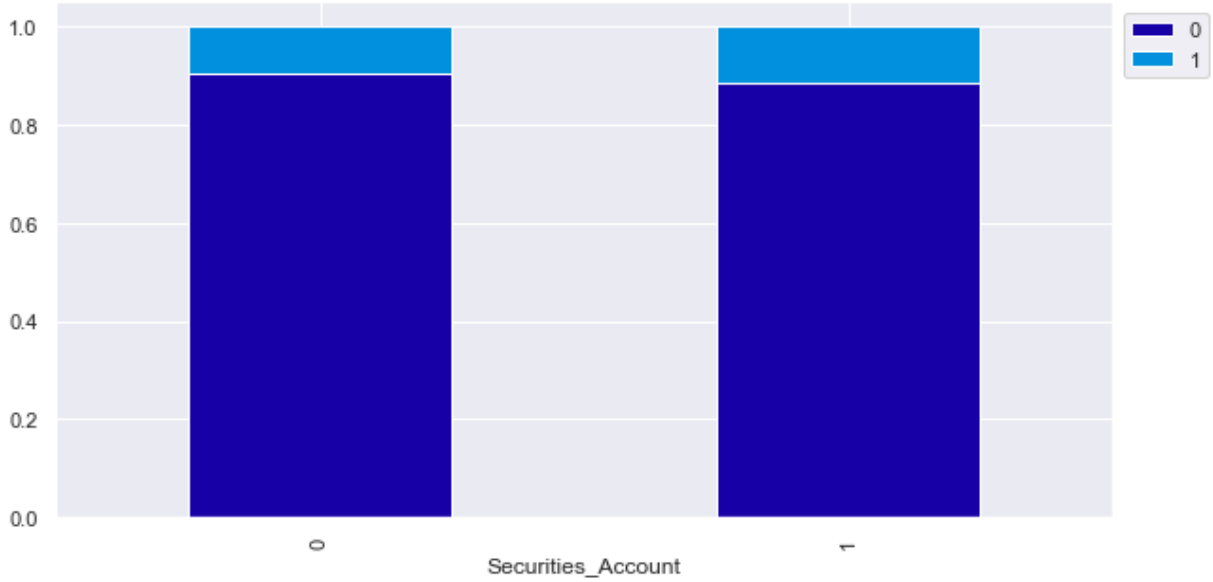
Observation:

- Higher the level os education, higher the chance of accepting a Personal Loan.

In [70]:
```
stacked_plot(df["Securities_Account"])
```

```
Personal_Loan          0    1   All
Securities_Account
0                   4046  417  4463
```

```
1                  461   59   520
All               4507  476  4983
------------------------------------------------------------------------------------------
------------------------------------------
```

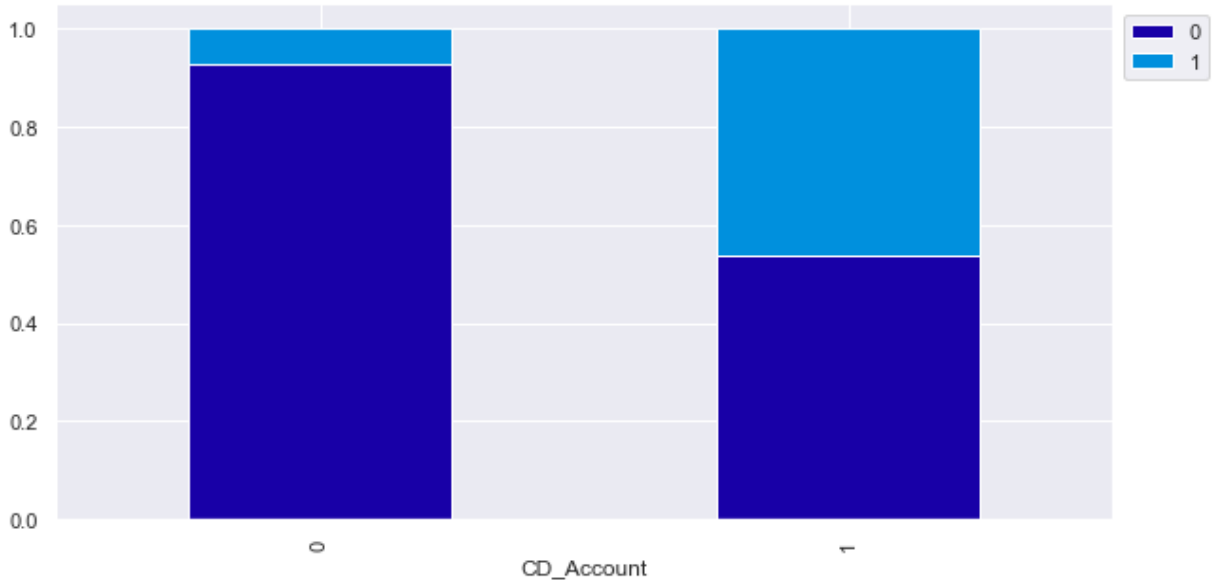


Observation

- Customers with Securities Account has high probability of accepting a Personal Loan

In [71]:
```
stacked_plot(df["CD_Account"])
```

```
Personal_Loan     0    1   All
CD_Account
0              4345  337  4682
1               162  139   301
All            4507  476  4983
------------------------------------------------------------------------------------------
------------------------------------------
```

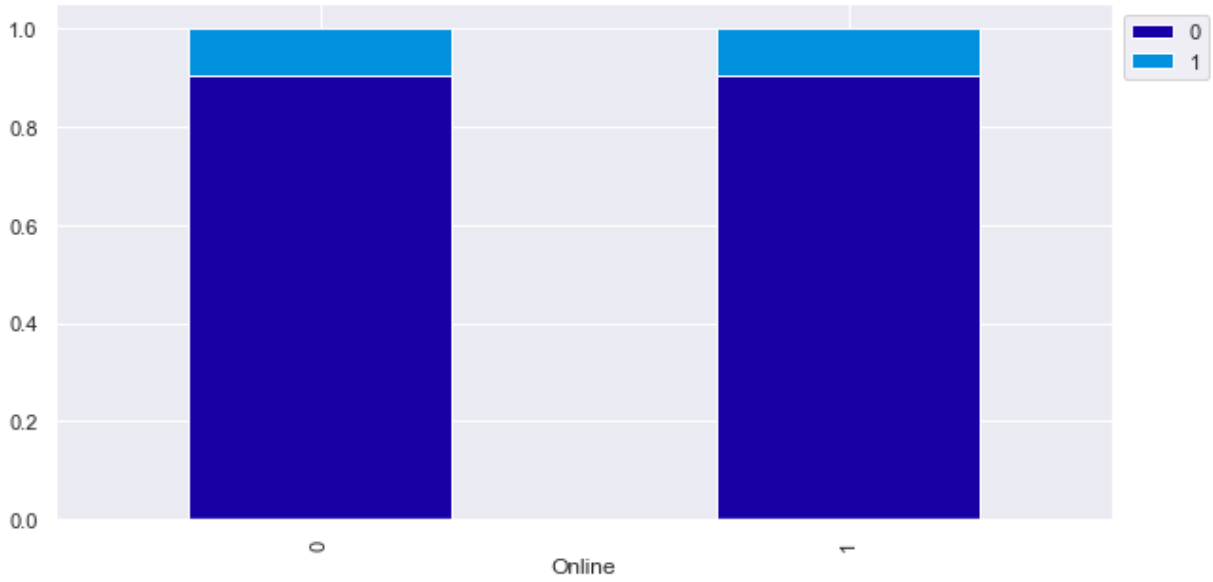


Observations:

- Only 6% of customers have a CD_Account
- customers with CD_Account is more likehood to accept a Personal Loan (46,17%)
- a good strategy is try to convert more clientes to become a CD_Account.

In [72]:
```
stacked_plot(df["Online"])
```

```
Personal_Loan     0    1   All
Online
0              1821  188  2009
1              2686  288  2974
All            4507  476  4983
------------------------------------------------------------------------------
------------------------------------------
```

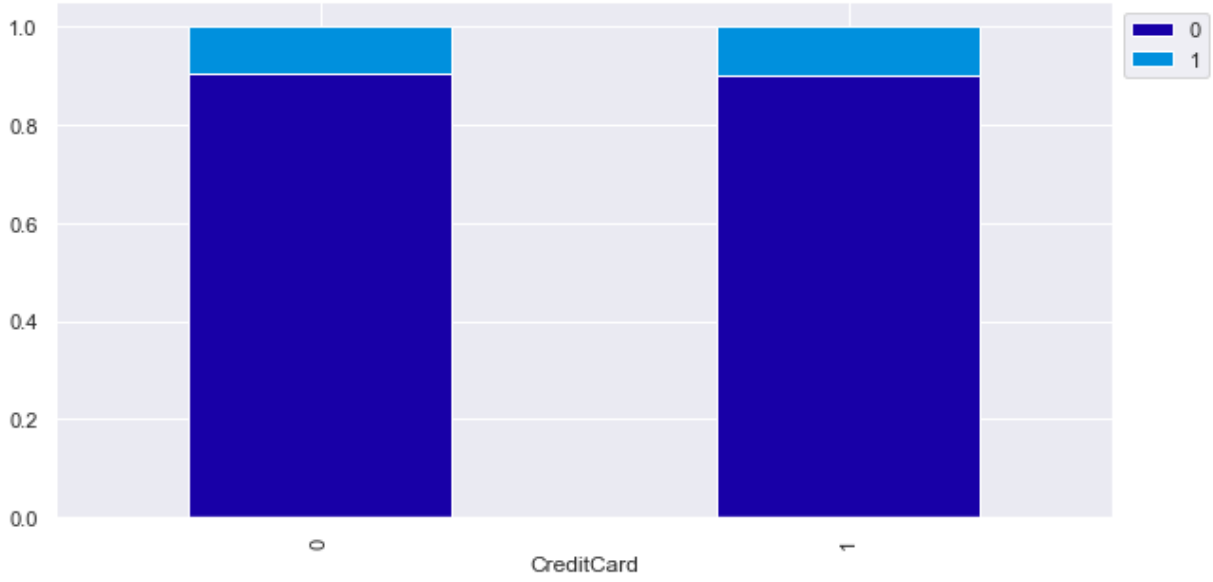


Observation

- Customers with or without Online banking facilities have almost the same probability of accepting a Personal Loan

In [73]:
```
stacked_plot(df["CreditCard"])
```

```
Personal_Loan     0    1   All
CreditCard
0              3184  333  3517
1              1323  143  1466
All            4507  476  4983
------------------------------------------------------------------------------
------------------------------------------
```

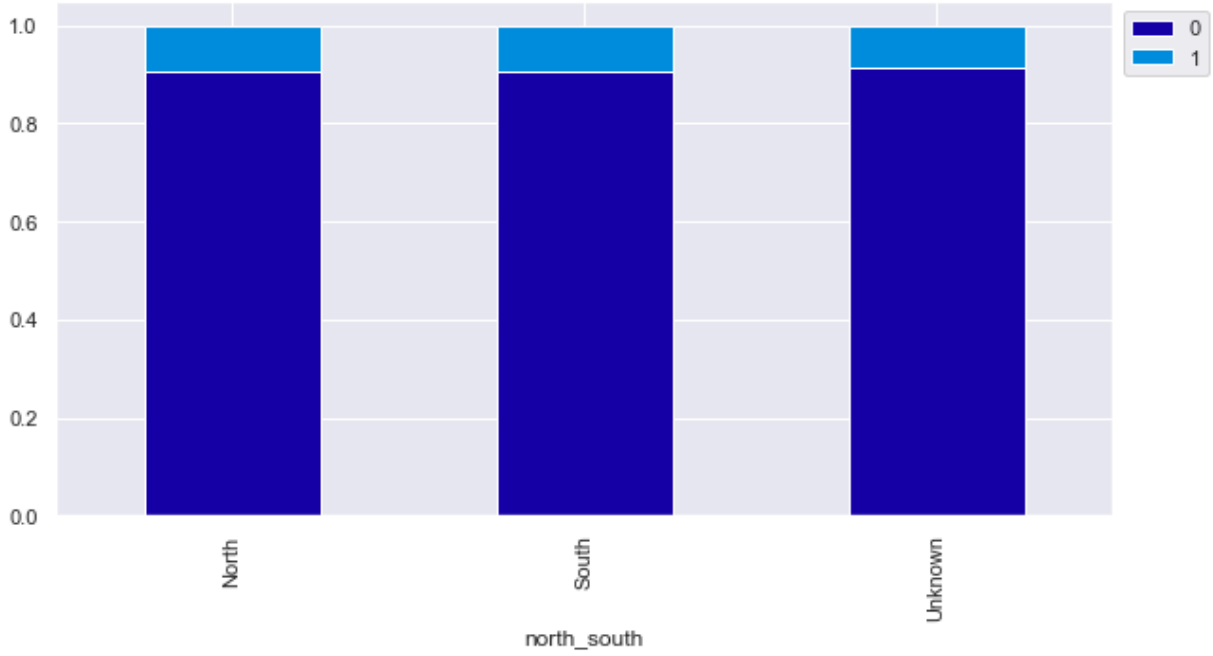


Observation

- Customers with or without CreditCard with other banks have almost the same probability of accepting a Personal Loan

In [74]:
```
stacked_plot(df["north_south"])
```

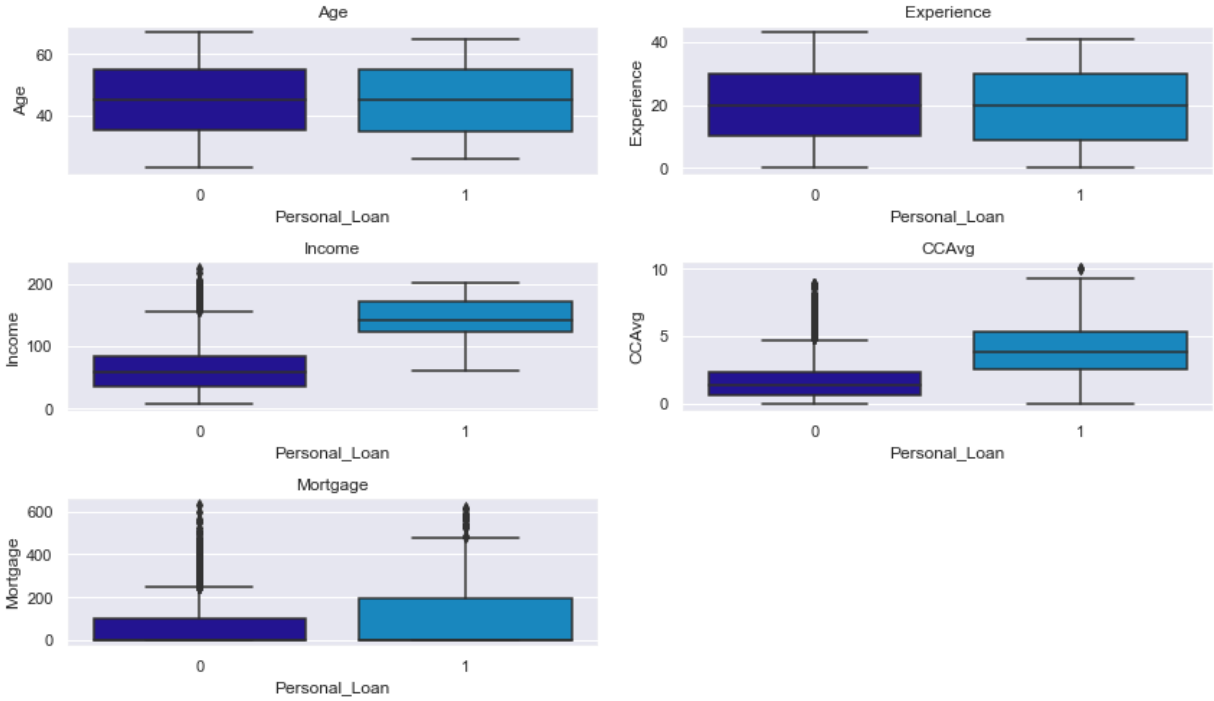
```
Personal_Loan     0    1   All
north_south
North          2238  237  2475
South          2238  236  2474
Unknown          31    3    34
All            4507  476  4983
------------------------------------------------------------------------------
------------------------------------------
```



Observation

- Customers living on North or South of California have almost the same probability of accepting a Personal Loan

In [75]:
```
cols = df[["Age", "Experience", "Income", "CCAvg", "Mortgage"]].columns.tolis
plt.figure(figsize=(12, 7))

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

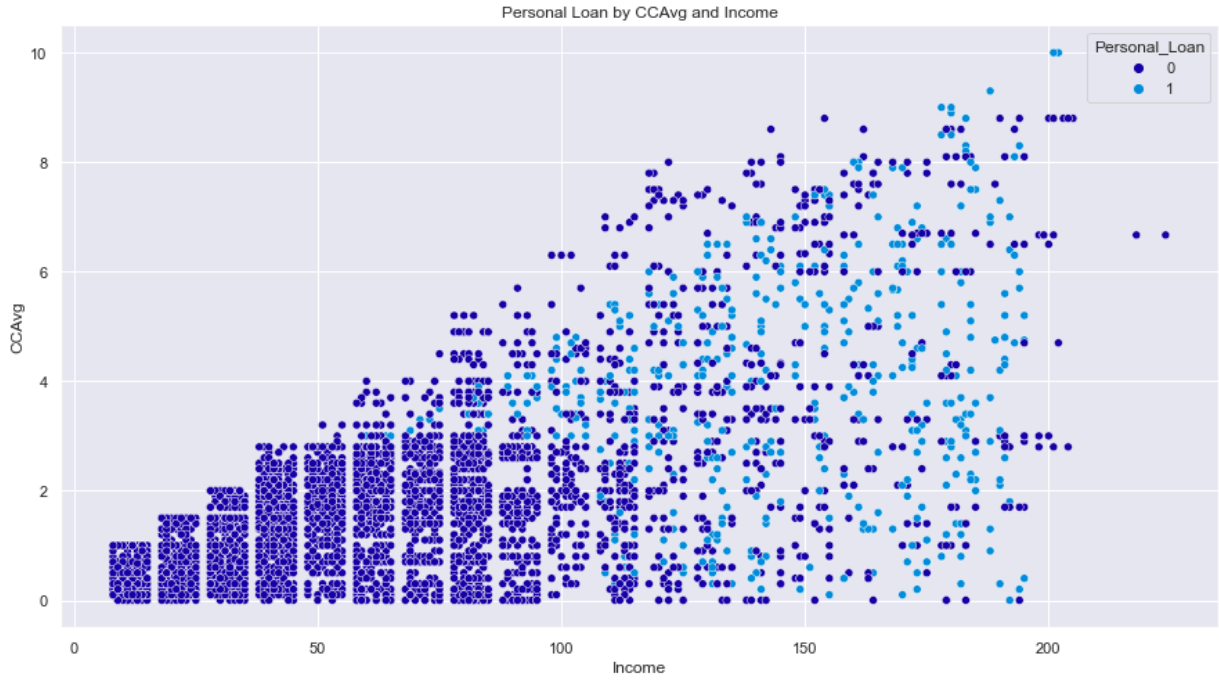


Observations:

- Customers with high `Income` and `CCCAvg` , has high likelihood to get a Personal_Loan.

In [76]:
```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(df["Income"], df["CCAvg"], hue=df["Personal_Loan"])

plt.title("Personal Loan by CCAvg and Income")
```

Out[76]:
```
Text(0.5, 1.0, 'Personal Loan by CCAvg and Income')
```

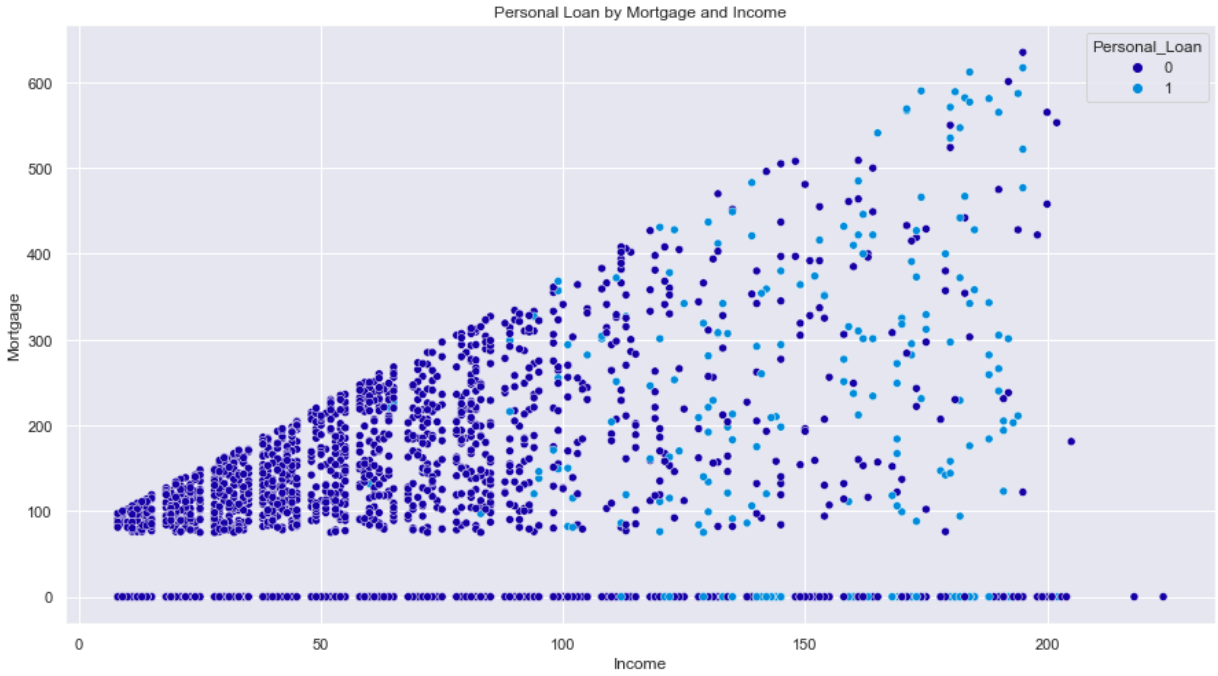


Observations:

- Customers with high `Income` usualy has high `CCAvg` , what is expect to happen.
- Higher `Income` , more likehood to get a Personal Loan.
- There is a wide range on `CCAvg` and `Income` , but doesn't seems to be `outliers` .

In [77]:
```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(df["Income"], df["Mortgage"], hue=df["Personal_Loan"])

plt.title("Personal Loan by Mortgage and Income")
```

Out[77]: `Text(0.5, 1.0, 'Personal Loan by Mortgage and Income')`

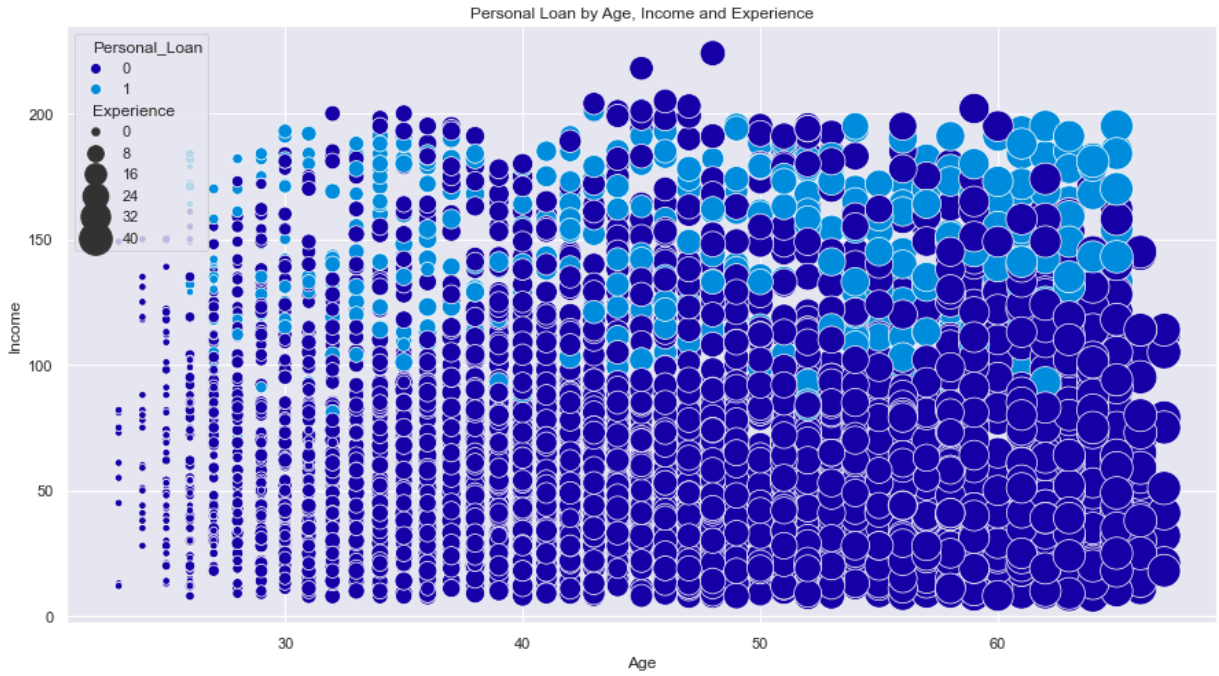


Observations:

- Customers with high `Income` usualy has high `Mortgage` , what is expect to happen.
- Higher `Income` , more likehood to get a Personal Loan.
- There is a wide range on `Mortgage` and `Income` , but doesn't seems to be `outliers` .

In [78]:
```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(
    df["Age"],
    df["Income"],
    hue=df["Personal_Loan"],
    size=df["Experience"],
    sizes=(30, 600),
)

plt.title("Personal Loan by Age, Income and Experience")
```

Out[78]: `Text(0.5, 1.0, 'Personal Loan by Age, Income and Experience')`

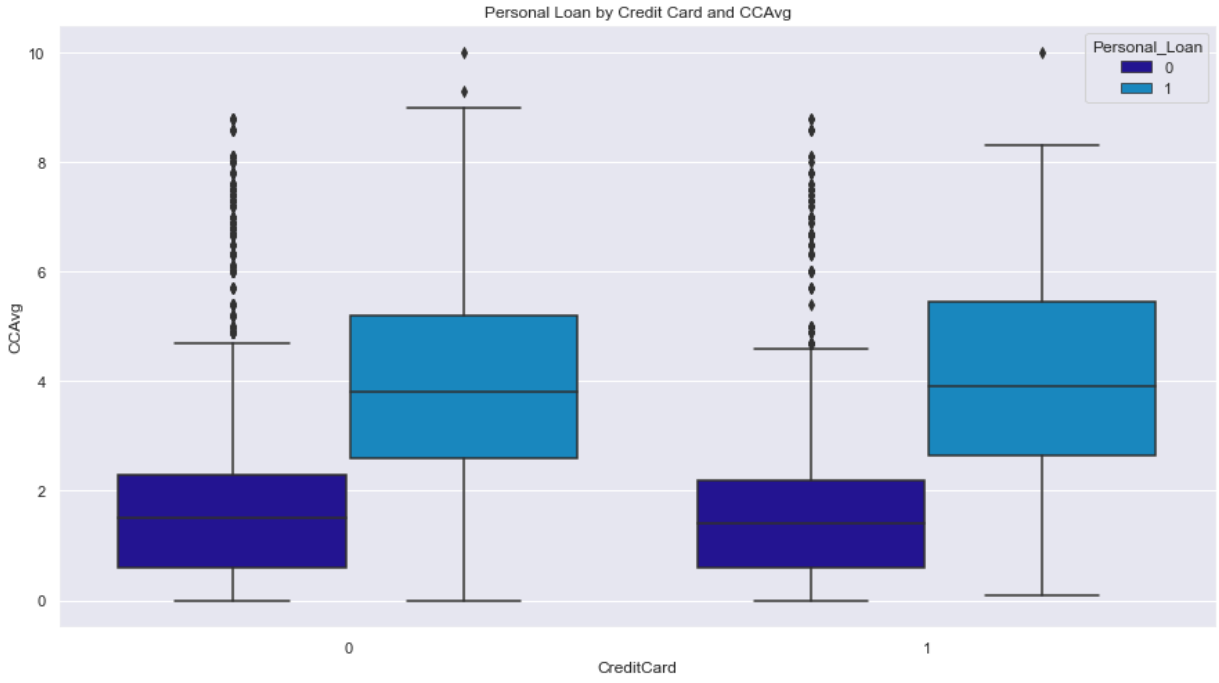


Observations:

- Customers with highest income (greater than $100 thousand/year) tend to accept Personal Loan.
- Age and Experience seems to have a strong correlation.

In [79]:
```
# Ploting boxplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.boxplot(df["CreditCard"], df["CCAvg"], hue=df["Personal_Loan"])

plt.title("Personal Loan by Credit Card and CCAvg")
```

Out[79]:
```
Text(0.5, 1.0, 'Personal Loan by Credit Card and CCAvg')
```



In [80]:
```
df.groupby(["CreditCard"])["CCAvg"].mean().sort_values(ascending=True)
```

Out[80]:
```
CreditCard
1    1.910286
0    1.932983
Name: CCAvg, dtype: float64
```

Observations:

- Customers average spending on credit cards per month its similar for the ones that have or don't have a Credit Card with other bank.

In [81]:
```
pd.crosstab(df.Personal_Loan, df.CreditCard, margins=True)
```

Out[81]:

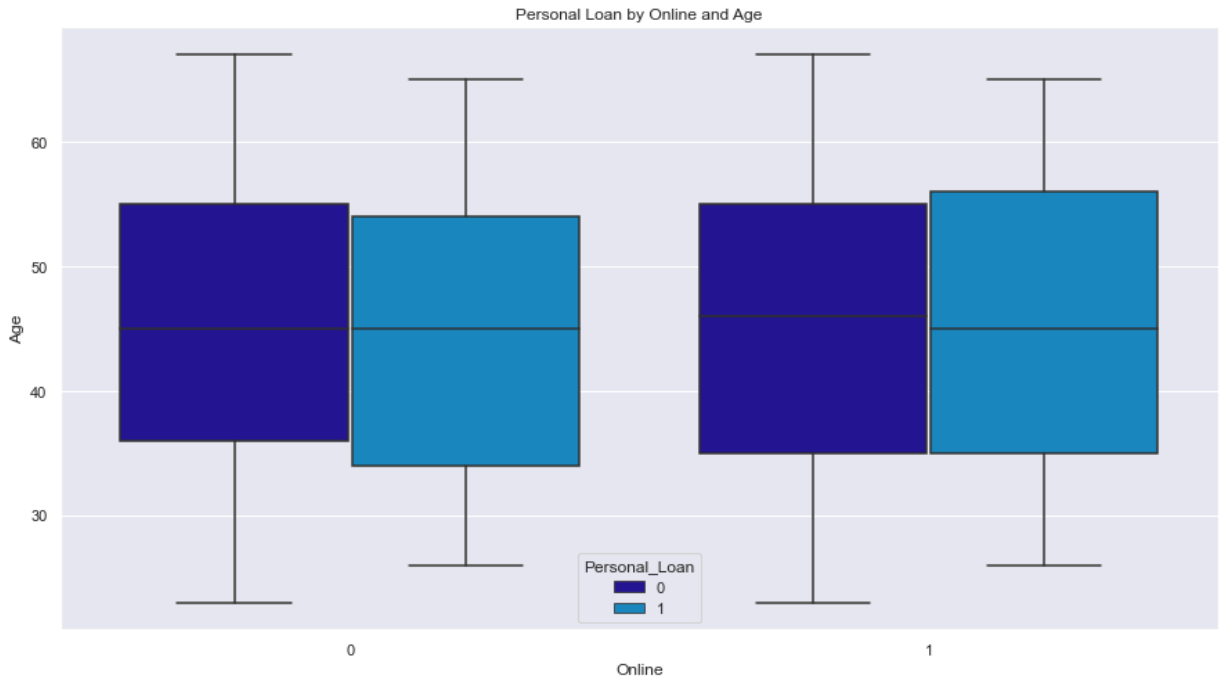
CreditCard	0	1	All
Personal_Loan			
0	3184	1323	4507
1	333	143	476
All	3517	1466	4983

Observations:

- 70.5% of customers doesn't have a Credit Card with other bank.
- Customers that accept a Personal Loan is around 9.5% for both segments (with and without Credit Card with other bank)
- Personal Loan and Credit Card doesn't seems to have correlation.

In [82]:
```
# Ploting boxplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.boxplot(df["Online"], df["Age"], hue=df["Personal_Loan"])

plt.title("Personal Loan by Online and Age")
```

Out[82]:
```
Text(0.5, 1.0, 'Personal Loan by Online and Age')
```



In [83]:
```
pd.crosstab(df.Personal_Loan, df.Online, margins=True)
```

Out[83]:

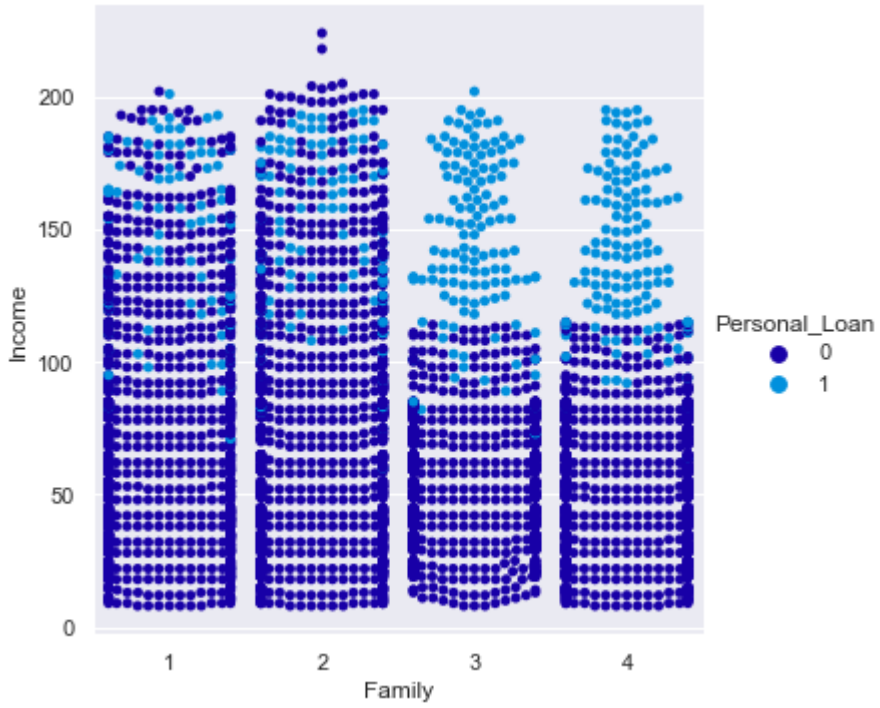
Personal_Loan \ Online	0	1	All
0	1821	2686	4507
1	188	288	476
All	2009	2974	4983

Observations:

- The `Age` between customers that use internet banking facilities (Online) and gets `Personal Loan` doesn't seems to vari.
- Accept a Personal Loan (9.5% both online subsets) doesn't have correlation with customers that uses or not `Online` facilities.

In [84]:
```
plt.figure(figsize=(15, 8))
sns.catplot(x="Family", y="Income", hue="Personal_Loan", data=df, kind="swarm
```

Out[84]:
```
<seaborn.axisgrid.FacetGrid at 0x20fe60f0e20>
<Figure size 1080x576 with 0 Axes>
```



In [85]:
```
pd.crosstab(df.Personal_Loan, df.Family, margins=True)
```

Out[85]:

Personal_Loan \ Family	1	2	3	4	All
0	1358	1184	877	1088	4507
1	106	106	131	133	476
All	1464	1290	1008	1221	4983

Observations:

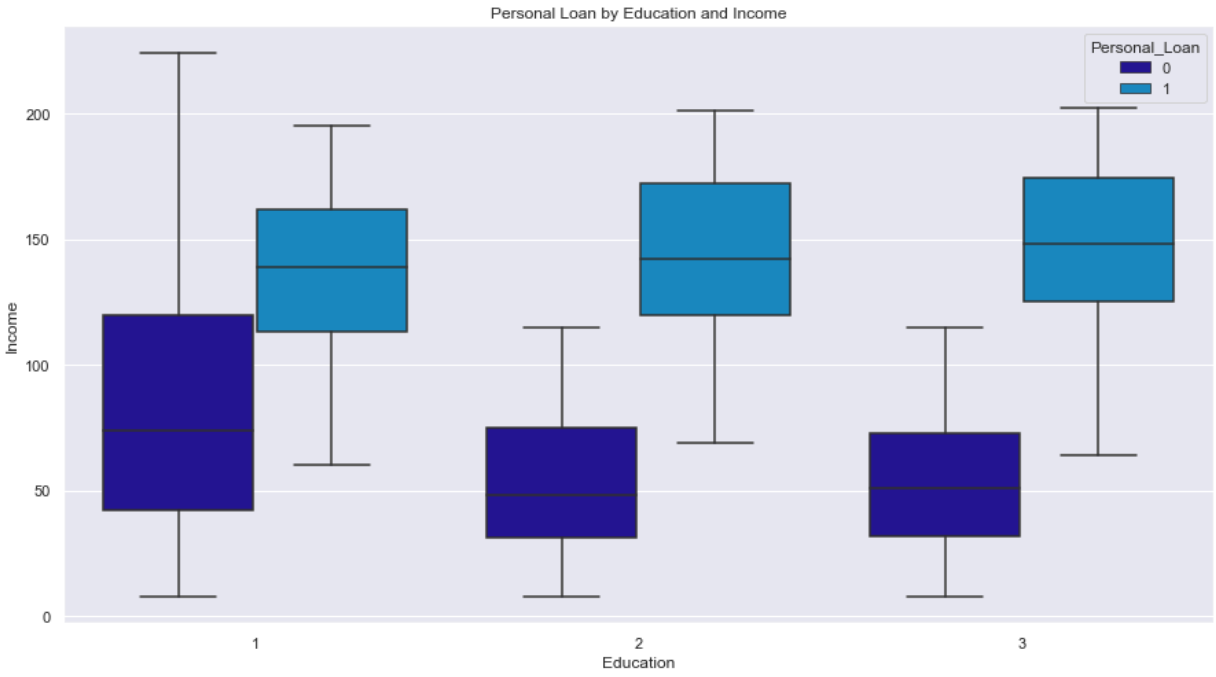
- Customers with high Family size and Income greater than 100 thousand is more likehood to accept a Personal Loan.
- 14.93% and 12.22% of customers with Family size 3 and 4 respectively accepted a Personal Loan.
- Is higher than the avarage of 9.55% of customers accepting the Personal Loan.
- Family size 1 and 2 has a accepting rate lower than avatage (7.24%, 8.95%)

Observations:

- Customers with higher `Education` and `Family` size has a highest likelihood of accepting a Personal Loan.

In [86]:
```
# Ploting boxplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.boxplot(df["Education"], df["Income"], hue=df["Personal_Loan"])

plt.title("Personal Loan by Education and Income")
```

Out[86]:
```
Text(0.5, 1.0, 'Personal Loan by Education and Income')
```



In [87]:
```
pd.crosstab(df.Personal_Loan, df.Education, margins=True)
```

Out[87]:

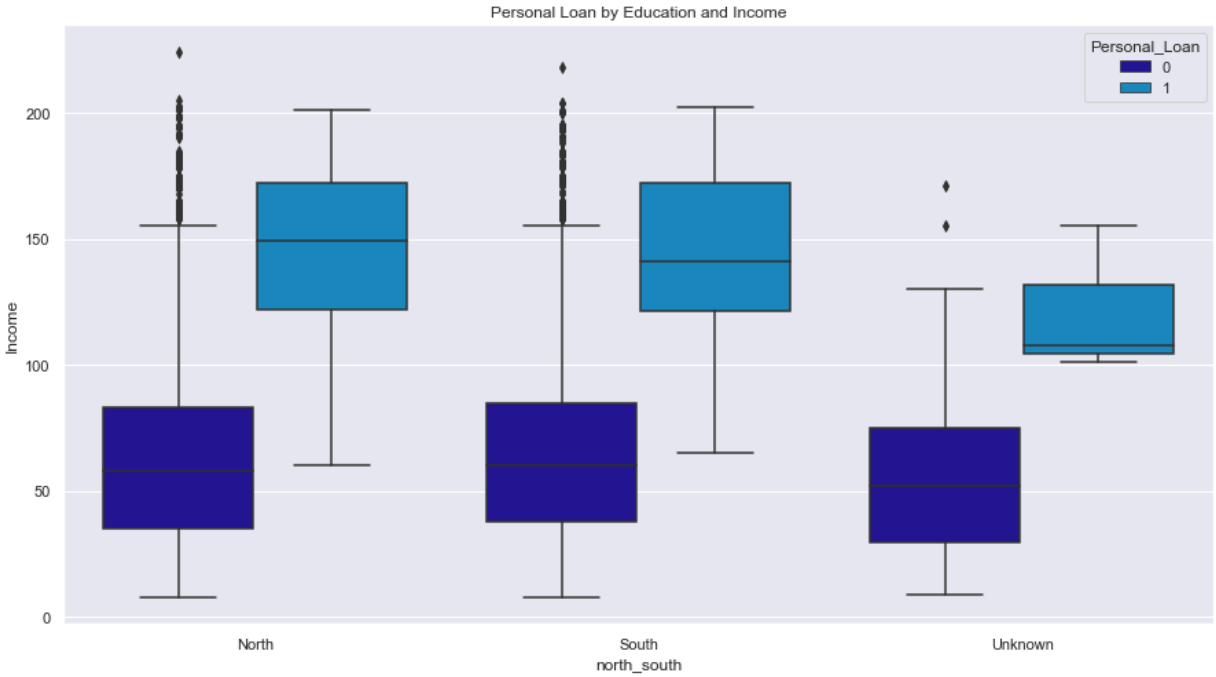
Personal_Loan \ Education	1	2	3	All
0	1990	1221	1296	4507
1	93	180	203	476
All	2083	1401	1499	4983

Observations:

- Customers with high Education is more likehood to accept a Personal Loan.
- 13.54% and 12.84% of customers with Education level 3 and 2 respectively accepted a Personal Loan.
- Is higher than the avarage of 9.55% of customers accepting the Personal Loan.
- Income and Education has no relationship.

In [88]:
```
# Ploting boxplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.boxplot(df["north_south"], df["Income"], hue=df["Personal_Loan"])

plt.title("Personal Loan by Education and Income")
```

Out[88]:
```
Text(0.5, 1.0, 'Personal Loan by Education and Income')
```



In [89]:
```
pd.crosstab(df.Personal_Loan, df.north_south, margins=True)
```

Out[89]:

north_south	North	South	Unknown	All
Personal_Loan				
0	2238	2238	31	4507
1	237	236	3	476
All	2475	2474	34	4983

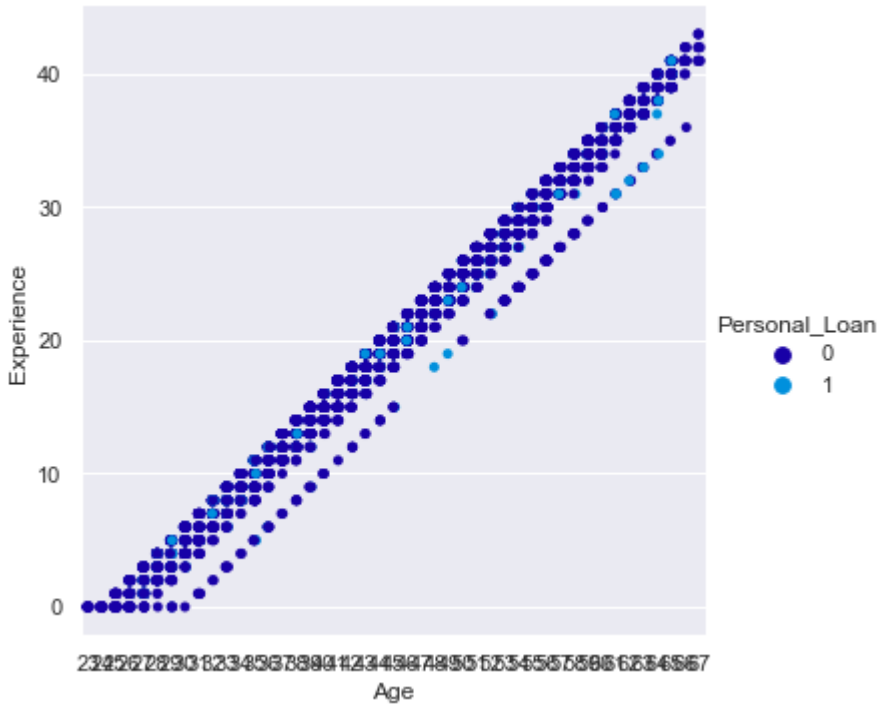
Observations:

- north_south, seems not to be a significante variable, both is giving a 10.5% of accepting rate, with is really close to the avarage (9.55%)

In [90]:
```
# Ploting catplot for analysis about correlation
plt.figure(figsize=(15, 10))
sns.catplot(x="Age", y="Experience", hue="Personal_Loan", data=df)
```

Out[90]:
```
<seaborn.axisgrid.FacetGrid at 0x20fedcc4a90>
```

```
<Figure size 1080x720 with 0 Axes>
```



- Age and Experience have a high, positive and strong correlation.
- I'll drooped on VIF instead of droping before modeling

Data Pre-Processing

Dropping ID

In [91]:
```
df.drop("ID", axis=1, inplace=True)
```

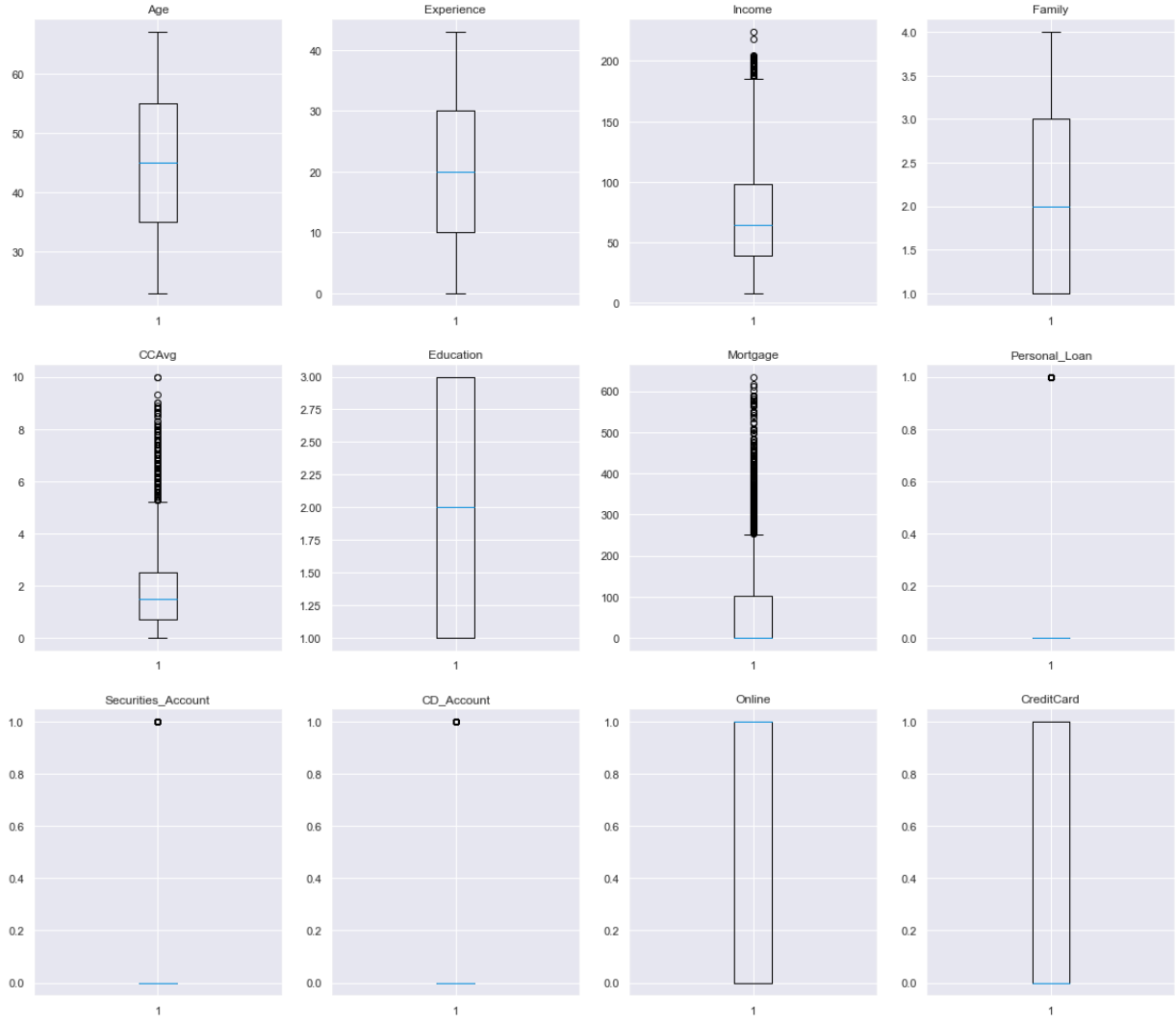
In [92]:
```
df.head()
```

Out[92]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_
0	25	1	49	4	1.6	1	0	0	
1	45	19	34	3	1.5	1	0	0	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	
4	35	8	45	4	1.0	2	0	0	

Outliers detection using boxplot

In [93]:
```
num_col = df.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20, 30))

for i, variable in enumerate(num_col):
    plt.subplot(5, 4, i + 1)
    plt.boxplot(df[variable], whis=1.5)
    plt.title(variable)
plt.show()
```



- `Income` , `CCAvg` , `Mortgage` has a wide range, showing some data greather than upper wiskers, but it all make sense, considering that customers with High Income is more likehood to spend more (CCAvg and Mortgage) (Analysis on EDA).
- It make sense that few customers gonna have high income. We not gonna treat it as outliers once it seems a real data and show the reality where few people have income greather than 75% of others.

In [94]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4983 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   Age                 4983 non-null   int64   
 1   Experience          4983 non-null   int64   
 2   Income              4983 non-null   int64   
 3   Family              4983 non-null   int64   
 4   CCAvg               4983 non-null   float64 
 5   Education           4983 non-null   int64   
 6   Mortgage            4983 non-null   int64   
 7   Personal_Loan       4983 non-null   int64   
 8   Securities_Account  4983 non-null   int64   
 9   CD_Account          4983 non-null   int64   
 10  Online              4983 non-null   int64   
 11  CreditCard          4983 non-null   int64   
 12  north_south         4983 non-null   category
dtypes: category(1), float64(1), int64(11)
memory usage: 671.1 KB
```

Logistic Regression

Creating a function to split, encode and add a constant to X

- The function will save us some time while checking high VIF values as we have to iteratively drop variables and check model performance.

In [95]:
```
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
    """
    X = df.drop([*kwargs], axis=1)
    Y = df["Personal_Loan"]

    X = pd.get_dummies(X, drop_first=True)
    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 [96]:
```
X_train, X_test, y_train, y_test = split("Personal_Loan")
```

In [97]:
```
# Checking shape of set
print("Shape of training set:", X_train.shape)
print("Shape of training set:", y_train.shape)
print("=-" * 17)
print("Shape of test set:", X_test.shape)
print("Shape of test set:", y_test.shape)
```

```
Shape of training set: (3488, 14)
Shape of training set: (3488,)
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Shape of test set: (1495, 14)
Shape of test set: (1495,)
```

In [98]:
```
X_test.head()
```

Out[98]:

	const	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Acco
4611	1.0	34	7	52	2	1.0	2	0	
2989	1.0	42	18	142	1	3.4	1	496	
4371	1.0	64	39	13	4	0.6	2	0	
1380	1.0	60	34	105	2	1.4	1	0	
374	1.0	30	5	98	2	3.1	1	220	

Building the Logistic Regression model

Model evaluation criterion:

Insights:

- **True Positives:**
  - Reality: A customer accepted Personal Loan.
  - Model predicted: The customer will accept a Personal Loan.
  - Outcome: The model is good.
- **True Negatives:**
  - Reality: A customer did NOT accepted Personal Loan.
  - Model predicted: The customer will NOT accept a Personal Loan.
  - Outcome: The business is unaffected.
- **False Positives:**
  - Reality: A customer did NOT accepted Personal Loan.
  - Model predicted: The customer will accepted Personal Loan.
  - Outcome: The team which is targeting the potential customers will be wasting their effort on customers who will not accepted Personal Loan.
- **False Negatives:**
  - Reality: A customer accepted Personal Loan.
  - Model predicted: The customer will NOT accepted Personal Loan.
  - Outcome: The potential customer is missed by the marketing team, the team could have devise campaigns with better target marketing to increase the success ratio. (Earn more through the interest on loans will get affected.)

How to reduce losses?

- We can use accuracy but since the data is imbalanced it would not be the right metric to check the model performance.
- Therefore, `Recall` gives the ratio of True Positives to Actual Positives, so high Recall implies `low false negatives` , i.e. low chances of predicting customers that wont accept Personal Loan, when it actualy will.

In [99]:
```
# The get_metrics_score function will be used to check the model performance
def get_metrics_score(
    model, library, train, test, train_y, test_y, threshold=0.5, flag=True, r
):
    """
    Function to calculate different metric scores of the model - Accuracy, Re
    library: Takes two arguments stats for statsmodels and sklearn for sklear
    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 d
    roc: If the roc is set to True then only roc score will be displayed. The
    """
    # defining an empty list to store train and test results
    if library == "stats":
        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_f1,
            )
        )

    elif library == "sklearn":
        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_f1,
            )
        )

    if flag == True:
        print(
            "Accuracy on training set : {0:0.4f}".format(
                accuracy_score(pred_train, train_y)
            )
        )
        print(
            "Accuracy on test set : {0:0.4f}".format(accuracy_score(pred_test
        )
        print()
        print(
            "Recall on training set :{0:0.4f}".format(recall_score(train_y, p
        )
        print("Recall on test set : {0:0.4f}".format(recall_score(test_y, pre
        print()
        print(
            "Precision on training set : {0:0.4f}".format(
                precision_score(train_y, pred_train)
            )
        )
        print(
            "Precision on test set : {0:0.4f}".format(
                precision_score(test_y, pred_test)
            )
        )
        print()
        print("F1 on training set : {0:0.4f}".format(f1_score(train_y, pred_t
        print("F1 on test set : {0:0.4f}".format(f1_score(test_y, pred_test))
        print()

    if roc == True:
        print(
            "ROC-AUC Score on training set : {0:0.4f}".format(
                roc_auc_score(train_y, pred_train)
            )
        )
        print(
            "ROC-AUC Score on test set : {0:0.4f}".format(
                roc_auc_score(test_y, pred_test)
            )
        )

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

In [100…

```
def make_confusion_matrix(
    model, library, test_X, y_actual, threshold=0.5, labels=[0, 1]
```

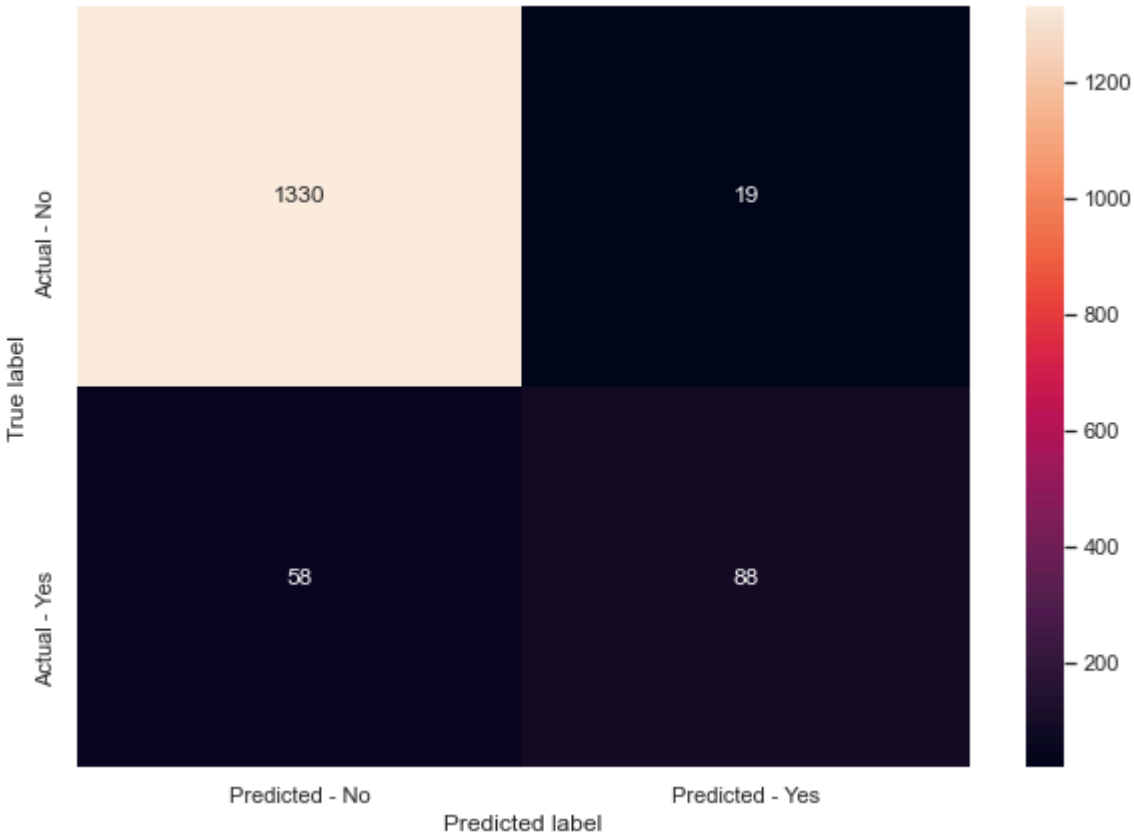
```
):
    """
    model : classifier to predict values of X
    library: Takes two arguments stats for statsmodels and sklearn for sklear
    test_X: test set
    y_actual : ground truth
    threshold: thresold for classifiying the observation as 1

    """

    if library == "sklearn":
        y_predict = model.predict(test_X)
        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_percent
        labels = np.asarray(labels).reshape(2, 2)"""
        plt.figure(figsize=(10, 7))
        sns.heatmap(df_cm, annot=True, fmt="")  #'''annot=labels,'''
        plt.ylabel("True label")
        plt.xlabel("Predicted label")

    elif library == "stats":
        y_predict = model.predict(test_X) > threshold
        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_percent
        labels = np.asarray(labels).reshape(2, 2)"""
        plt.figure(figsize=(10, 7))
        sns.heatmap(df_cm, annot=True, fmt="")
        plt.ylabel("True label")
        plt.xlabel("Predicted label")
```

Logistic Regression (with Sklearn library)

In [101…

```
# There are different solvers available in Sklearn logistic regression
# The newton-cg solver is faster for high-dimensional data

lr = LogisticRegression(solver='newton-cg',random_state=42,fit_intercept=Fals
model  = lr.fit(X_train,y_train)

# confusion matrix

make_confusion_matrix(lr,'sklearn',X_test,y_test)

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

```
Accuracy on training set : 0.9518
Accuracy on test set : 0.9485

Recall on training set :0.6212
Recall on test set : 0.6027

Precision on training set : 0.8266
Precision on test set : 0.8224

F1 on training set : 0.7093
F1 on test set : 0.6957
```



- We have build a logistic regression model which shows good performance on the train and test sets but to identify significant variables we will have to build a logistic regression model using the statsmodels library.
- We will now perform logistic regression using statsmodels, a Python module that provides functions for the estimation of many statistical models, as well as for conducting statistical tests, and statistical data exploration.
- Using statsmodels, we will be able to check the statistical validity of our model - identify the significant predictors from p-values that we get for each predictor variable.

Logistic Regression (with statsmodels library)

In [102…

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

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

```
Optimization terminated successfully.
        Current function value: 0.123589
```

```
         Iterations 9
Accuracy on training set : 0.9558
Accuracy on test set : 0.9472

Recall on training set :0.6576
Recall on test set : 0.6233

Precision on training set : 0.8411
Precision on test set : 0.7913

F1 on training set : 0.7381
F1 on test set : 0.6973
```

In [103…
```
lg.summary()
```

Out[103…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3488
Model:	Logit	**Df Residuals:**	3474
Method:	MLE	**Df Model:**	13
Date:	Fri, 16 Jul 2021	**Pseudo R-squ.:**	0.6052
Time:	23:31:30	**Log-Likelihood:**	-431.08
converged:	True	**LL-Null:**	-1092.0
Covariance Type:	nonrobust	**LLR p-value:**	1.040e-274

	coef	std err	z	P>|z|	[0.025	0.975]
const	-12.7522	2.064	-6.177	0.000	-16.798	-8.706
Age	-0.0355	0.077	-0.461	0.645	-0.187	0.116
Experience	0.0416	0.077	0.541	0.588	-0.109	0.192
Income	0.0561	0.003	17.279	0.000	0.050	0.062
Family	0.6543	0.090	7.249	0.000	0.477	0.831
CCAvg	0.1482	0.047	3.136	0.002	0.056	0.241
Education	1.7605	0.143	12.303	0.000	1.480	2.041
Mortgage	7.945e-05	0.001	0.119	0.906	-0.001	0.001
Securities_Account	-0.8750	0.340	-2.571	0.010	-1.542	-0.208
CD_Account	4.2116	0.397	10.612	0.000	3.434	4.989
Online	-0.8920	0.193	-4.615	0.000	-1.271	-0.513
CreditCard	-1.1071	0.251	-4.419	0.000	-1.598	-0.616
north_south_South	0.1336	0.182	0.734	0.463	-0.223	0.491
north_south_Unknown	1.2767	0.891	1.433	0.152	-0.469	3.023

We will need to remove multicollinearity from the data to get reliable coefficients and p-values.

- Let's use: Variation Inflation Factor.

Multicollinearity

In [104…
```
# changing datatype of colums to numeric for checking vif
X_train_num = X_train.astype(float).copy()
```

In [105…
```
vif_series1 = pd.Series(
    [
        variance_inflation_factor(X_train_num.values, i)
        for i in range(X_train_num.shape[1])
    ],
    index=X_train_num.columns,
    dtype=float,
)
print("Series before feature selection: \n\n{}\n".format(vif_series1))
```

```
Series before feature selection:

const                  462.845291
Age                     96.415012
Experience              96.396367
Income                   1.813867
Family                   1.036775
CCAvg                    1.691666
Education                1.105620
Mortgage                 1.051600
Securities_Account       1.133194
CD_Account               1.319910
Online                   1.036037
CreditCard               1.104534
north_south_South        1.013598
north_south_Unknown      1.010990
dtype: float64
```

- Only `Age` and `Experience` has VIF > 5.
- Other variables does not exhibit high multicollinearity, so the values in summary are reliable.

1. Removing Age

> `AGE` has the highest VIF, let's remove it and check other values

In [106…
```
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])
    ],
    index=X_train_num1.columns,
)
print("Series before feature selection: \n\n{}\n".format(vif_series2))
```

```
Series before feature selection:

const                  22.825653
Experience              1.007771
Income                  1.811555
Family                  1.035542
CCAvg                   1.689412
Education               1.042778
Mortgage                1.051596
Securities_Account      1.132801
CD_Account              1.319406
Online                  1.036005
```

```
CreditCard              1.104531
north_south_South       1.013312
north_south_Unknown     1.010673
dtype: float64
```

- Removal of `Age` has fixed multicollinearity in `Experience` column

Model with all the features except `Age`

In [107…
```
X_train1, X_test1, y_train, y_test = split("Personal_Loan", "Age")
```

In [108…
```
logit2 = sm.Logit(y_train, X_train1.astype(float))
lg2 = logit2.fit(warn_convergence=False)

# Let's check model performances for this model
scores_LR = get_metrics_score(
    lg2, "stats", X_train1, X_test1, y_train, y_test, flag=True
)
```

```
Optimization terminated successfully.
         Current function value: 0.123620
         Iterations 9
Accuracy on training set : 0.9553
Accuracy on test set : 0.9465

Recall on training set :0.6545
Recall on test set : 0.6164

Precision on training set : 0.8372
Precision on test set : 0.7895

F1 on training set : 0.7347
F1 on test set : 0.6923
```

- A small change in the model performance.

> Recall on training set : 0.6576 to 0.6545
>
> Recall on test set : 0.6233 to 0.6164

2. Removing Experience

> `Experience` has a high VIF, if we keep `Age` . Let's remove it and check if our model performe better without `Experience` .

In [109…
```
# From completed data, we drop Experience | Age is back on data
X_train_num2 = X_train_num.drop("Experience", axis=1)
vif_series3 = pd.Series(
    [
        variance_inflation_factor(X_train_num2.values, i)
        for i in range(X_train_num2.shape[1])
    ],
    index=X_train_num2.columns,
)
print("Series before feature selection: \n\n{}\n".format(vif_series3))
```

```
Series before feature selection:
```

```
const                 35.597961
Age                    1.007966
Income                 1.811859
Family                 1.035109
CCAvg                  1.689183
Education              1.043247
Mortgage               1.051597
Securities_Account     1.132789
CD_Account             1.319292
Online                 1.036016
CreditCard             1.104521
north_south_South      1.013167
north_south_Unknown    1.010751
dtype: float64
```

Model with all the features except `Experience`

In [110…
```
X_train2, X_test2, y_train, y_test = split("Personal_Loan", "Experience")
```

In [111…
```
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_score(
    lg3, "stats", X_train2, X_test2, y_train, y_test, flag=True
)
```

```
Optimization terminated successfully.
         Current function value: 0.123632
         Iterations 9
Accuracy on training set : 0.9556
Accuracy on test set : 0.9465

Recall on training set :0.6545
Recall on test set : 0.6164

Precision on training set : 0.8405
Precision on test set : 0.7895

F1 on training set : 0.7359
F1 on test set : 0.6923
```

- No change in the model performance when removing `Age` or `Experience` .

> Recall on training set : AGE:0.6545 | Experience: 0.6545
>
> Recall on test set : AGE:0.6164| Experience: 0.6164

- We gonna keep Experience because the VIF was smaller | Lg2.

Summary of the model

In [112…
```
# Model without AGE
lg2.summary()
```

Out[112…

Logit Regression Results

Dep. Variable:	Personal_Loan	**No. Observations:**	3488
Model:	Logit	**Df Residuals:**	3475

Method:	MLE	**Df Model:**	12
Date:	Fri, 16 Jul 2021	**Pseudo R-squ.:**	0.6051
Time:	23:31:31	**Log-Likelihood:**	-431.19
converged:	True	**LL-Null:**	-1092.0
Covariance Type:	nonrobust	**LLR p-value:**	1.079e-275

	coef	std err	z	P>|z|	[0.025	0.975]
const	-13.6433	0.739	-18.459	0.000	-15.092	-12.195
Experience	0.0064	0.008	0.815	0.415	-0.009	0.022
Income	0.0561	0.003	17.317	0.000	0.050	0.062
Family	0.6544	0.090	7.246	0.000	0.477	0.831
CCAvg	0.1480	0.047	3.133	0.002	0.055	0.241
Education	1.7498	0.141	12.400	0.000	1.473	2.026
Mortgage	7.052e-05	0.001	0.105	0.916	-0.001	0.001
Securities_Account	-0.8698	0.340	-2.562	0.010	-1.535	-0.204
CD_Account	4.2134	0.396	10.633	0.000	3.437	4.990
Online	-0.8890	0.193	-4.603	0.000	-1.267	-0.510
CreditCard	-1.1043	0.250	-4.410	0.000	-1.595	-0.614
north_south_South	0.1340	0.182	0.736	0.462	-0.223	0.491
north_south_Unknown	1.2748	0.890	1.432	0.152	-0.470	3.020

- Experience, Mortage and north_south categori has high p-value and can be dropped.

In [113…
```
# Checking columns and making sure this is the correct data, without `Age`
X_train1.columns
```

Out[113…
```
Index(['const', 'Experience', 'Income', 'Family', 'CCAvg', 'Education',
       'Mortgage', 'Securities_Account', 'CD_Account', 'Online', 'CreditCar
d',
       'north_south_South', 'north_south_Unknown'],
      dtype='object')
```

In [114…
```
# Dropping Experience because it is no a significante variable for our model
X_train3 = X_train1.drop(["Experience"], axis=1)
X_test3 = X_test1.drop(["Experience"], axis=1)

logit4 = sm.Logit(y_train, X_train3.astype(float))
lg4 = logit4.fit(warn_convergence=False)

print(lg4.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.123716
         Iterations 9
                           Logit Regression Results
==============================================================================
=
Dep. Variable:          Personal_Loan   No. Observations:                 348
8
Model:                          Logit   Df Residuals:                     347
6
```

```
Method:                           MLE   Df Model:                           1
1
Date:                Fri, 16 Jul 2021   Pseudo R-squ.:                   0.604
8
Time:                        23:31:31   Log-Likelihood:                -431.5
2
converged:                       True   LL-Null:                       -1092.
0
Covariance Type:            nonrobust   LLR p-value:                 1.338e-27
6
==============================================================================
==========
                           coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----------
const                -13.4774      0.706    -19.103      0.000     -14.860
-12.095
Income                 0.0560      0.003     17.323      0.000       0.050
0.062
Family                 0.6491      0.090      7.205      0.000       0.473
0.826
CCAvg                  0.1443      0.047      3.068      0.002       0.052
0.237
Education              1.7439      0.141     12.403      0.000       1.468
2.019
Mortgage            7.675e-05      0.001      0.115      0.909      -0.001
0.001
Securities_Account    -0.8719      0.339     -2.570      0.010      -1.537
-0.207
CD_Account             4.2197      0.396     10.648      0.000       3.443
4.996
Online                -0.8864      0.193     -4.593      0.000      -1.265
-0.508
CreditCard            -1.1024      0.251     -4.397      0.000      -1.594
-0.611
north_south_South      0.1423      0.182      0.783      0.434      -0.214
0.499
north_south_Unknown    1.2621      0.891      1.416      0.157      -0.485
3.009
==============================================================================
==========
```

In [115…

```
# Dropping Mortgage because it is no a significante variable for our model
X_train4 = X_train3.drop(["Mortgage"], axis=1)
X_test4 = X_test3.drop(["Mortgage"], axis=1)

logit5 = sm.Logit(y_train, X_train4.astype(float))
lg5 = logit5.fit(warn_convergence=False)

print(lg5.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.123717
         Iterations 9
                           Logit Regression Results
==============================================================================
=
Dep. Variable:          Personal_Loan   No. Observations:                348
8
Model:                          Logit   Df Residuals:                    347
7
Method:                           MLE   Df Model:                           1
0
Date:                Fri, 16 Jul 2021   Pseudo R-squ.:                   0.604
8
Time:                        23:31:31   Log-Likelihood:                -431.5
3
```

```
converged:                       True   LL-Null:                       -1092.
0
Covariance Type:            nonrobust   LLR p-value:                 1.142e-27
7
==============================================================================
==========
                             coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----------
const                   -13.4741      0.705    -19.118      0.000     -14.855
-12.093
Income                    0.0561      0.003     17.417      0.000       0.050
0.062
Family                    0.6494      0.090      7.211      0.000       0.473
0.826
CCAvg                     0.1439      0.047      3.069      0.002       0.052
0.236
Education                 1.7432      0.140     12.414      0.000       1.468
2.018
Securities_Account       -0.8721      0.339     -2.571      0.010      -1.537
-0.207
CD_Account                4.2224      0.396     10.670      0.000       3.447
4.998
Online                   -0.8857      0.193     -4.592      0.000      -1.264
-0.508
CreditCard               -1.1040      0.250     -4.410      0.000      -1.595
-0.613
north_south_South         0.1428      0.182      0.786      0.432      -0.213
0.499
north_south_Unknown       1.2634      0.891      1.418      0.156      -0.483
3.010
==============================================================================
==========
```

In [116…

```
# Dropping north_south_South because it is no a significante variable for our
X_train5 = X_train4.drop(["north_south_South"], axis=1)
X_test5 = X_test4.drop(["north_south_South"], axis=1)

logit6 = sm.Logit(y_train, X_train5.astype(float))
lg6 = logit6.fit(warn_convergence=False)

print(lg6.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.123806
         Iterations 9
                           Logit Regression Results
==============================================================================
=
Dep. Variable:          Personal_Loan   No. Observations:                  348
8
Model:                          Logit   Df Residuals:                      347
8
Method:                           MLE   Df Model:
9
Date:                Fri, 16 Jul 2021   Pseudo R-squ.:                   0.604
5
Time:                        23:31:31   Log-Likelihood:                -431.8
4
converged:                       True   LL-Null:                       -1092.
0
Covariance Type:            nonrobust   LLR p-value:                 1.247e-27
8
==============================================================================
==========
                             coef    std err          z      P>|z|      [0.025
0.975]
```

```
------------------------------------------------------------------------------
----------
const                -13.3813      0.693    -19.319      0.000     -14.739
-12.024
Income                 0.0560      0.003     17.424      0.000       0.050
0.062
Family                 0.6461      0.090      7.182      0.000       0.470
0.822
CCAvg                  0.1451      0.047      3.096      0.002       0.053
0.237
Education              1.7423      0.140     12.411      0.000       1.467
2.017
Securities_Account    -0.8680      0.339     -2.560      0.010      -1.532
-0.204
CD_Account             4.2041      0.395     10.653      0.000       3.431
4.978
Online                -0.8808      0.193     -4.572      0.000      -1.258
-0.503
CreditCard            -1.0993      0.250     -4.393      0.000      -1.590
-0.609
north_south_Unknown    1.1865      0.885      1.340      0.180      -0.549
2.922
==============================================================================
==========
```

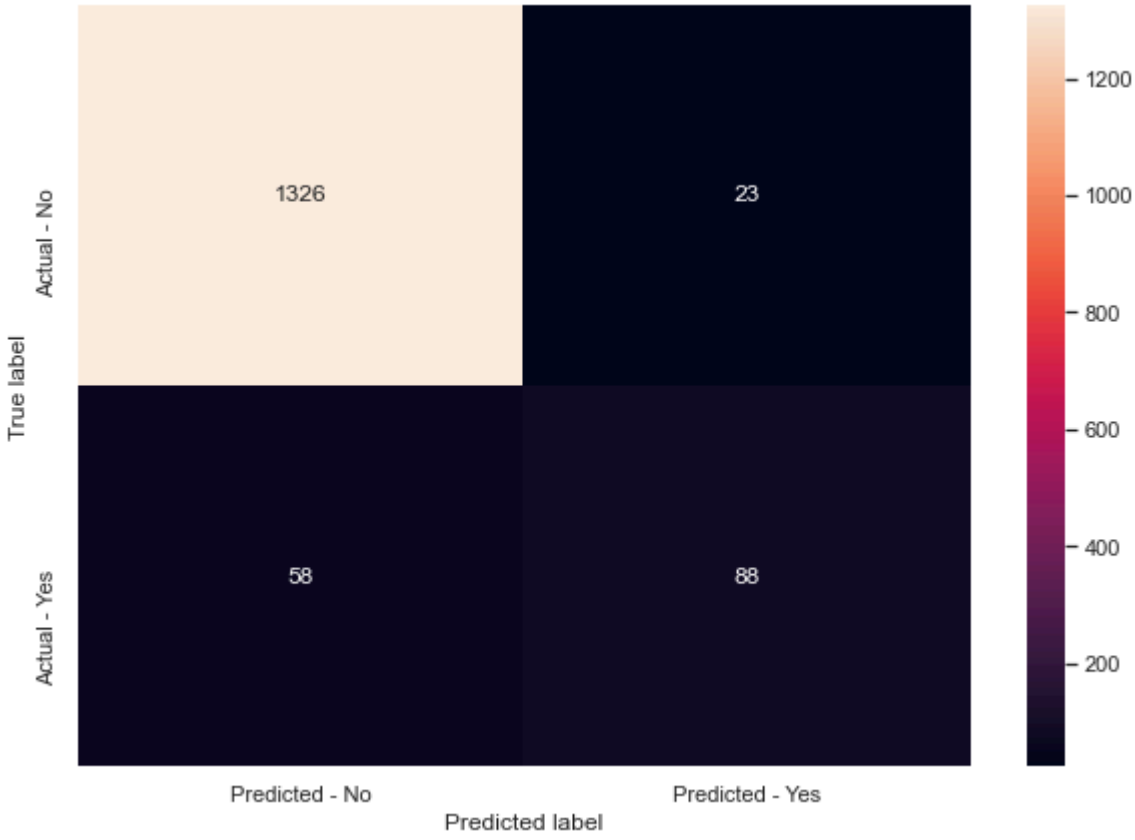
- Now all the columns left are significant predictors, let's check the model performance and make interpretations.

Metrics of final model 'lg6'

In [117…

```
# confusion matrix
make_confusion_matrix(lg6, "stats", X_test5, y_test)

# metrics
scores_LR = get_metrics_score(lg6, "stats", X_train5, X_test5, y_train, y_tes
```

```
Accuracy on training set : 0.9567
Accuracy on test set : 0.9458

Recall on training set :0.6606
Recall on test set : 0.6027

Precision on training set : 0.8482
Precision on test set : 0.7928

F1 on training set : 0.7428
F1 on test set : 0.6848
```



ROC-AUC

In [118…

```
# ROC-AUC on training set

logit_roc_auc_train = roc_auc_score(y_train, lg6.predict(X_train5))
fpr, tpr, thresholds = roc_curve(y_train, lg6.predict(X_train5))
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, label="Logistic Regression (area = %0.2f)" % logit_roc_auc
plt.plot([0, 1], [0, 1], "r--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")
plt.show()
```

In [119…

```
# ROC-AUC on test set

logit_roc_auc_test = roc_auc_score(y_test, lg6.predict(X_test5))
fpr, tpr, thresholds = roc_curve(y_test, lg6.predict(X_test5))
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, label="Logistic Regression (area = %0.2f)" % logit_roc_auc
plt.plot([0, 1], [0, 1], "r--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")
plt.show()
```

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

Coefficient interpretations

- Coefficient of Income, Family, CCAvg, Education, CD_Account are positive an increase in these will lead to increase in chances of a person accept the Personal Loan.
- Coefficient of Online, CreditCard and Securities_Account are negative increase in these will lead to decrease in chances of a person person accept the Personal Loan.

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 [120…
```
odds = np.exp(lg6.params)  # converting coefficients to odds
pd.set_option(
    "display.max_columns", None
)  # removing limit from number of columns to display
pd.DataFrame(
    odds, X_train5.columns, columns=["odds"]
).T  # adding the odds to a dataframe
```

Out[120…

	const	Income	Family	CCAvg	Education	Securities_Account	CD_Account	C
odds	0.000002	1.057551	1.908088	1.156192	5.710689	0.419804	66.960991	0.4

Percentage change in odds

In [121…
```
perc_change_odds = (np.exp(lg6.params) - 1) * 100  # finding the percentage c
pd.set_option(
    "display.max_columns", None
)  # removing limit from number of columns to display
pd.DataFrame(
    perc_change_odds, X_train5.columns, columns=["change_odds%"]
).T  # adding the change_odds% to a dataframe
```

Out[121…

	const	Income	Family	CCAvg	Education	Securities_Account	CD
change_odds%	-99.999846	5.75514	90.80881	15.619173	471.068865	-58.019563	65

Coefficient interpretations

- `Income` : Holding all other features constant a 1 unit change in Income will increase the odds of a person accepting Personal Loan by 1.05 times or a 5.73% increase in odds of accepting Personal Loan.
- `Family` : Holding all other features constant a 1 unit change in the Family will increase the odds of a person accepting Personal Loan by by 1.90 times or a increase of 90.8% in odds of accepting Personal Loan.

`Interpretation for other attributes is similarly.`

Model Performance Improvement

- Let's see if the Recall score can be improved further, by changing the model threshold using AUC-ROC Curve.

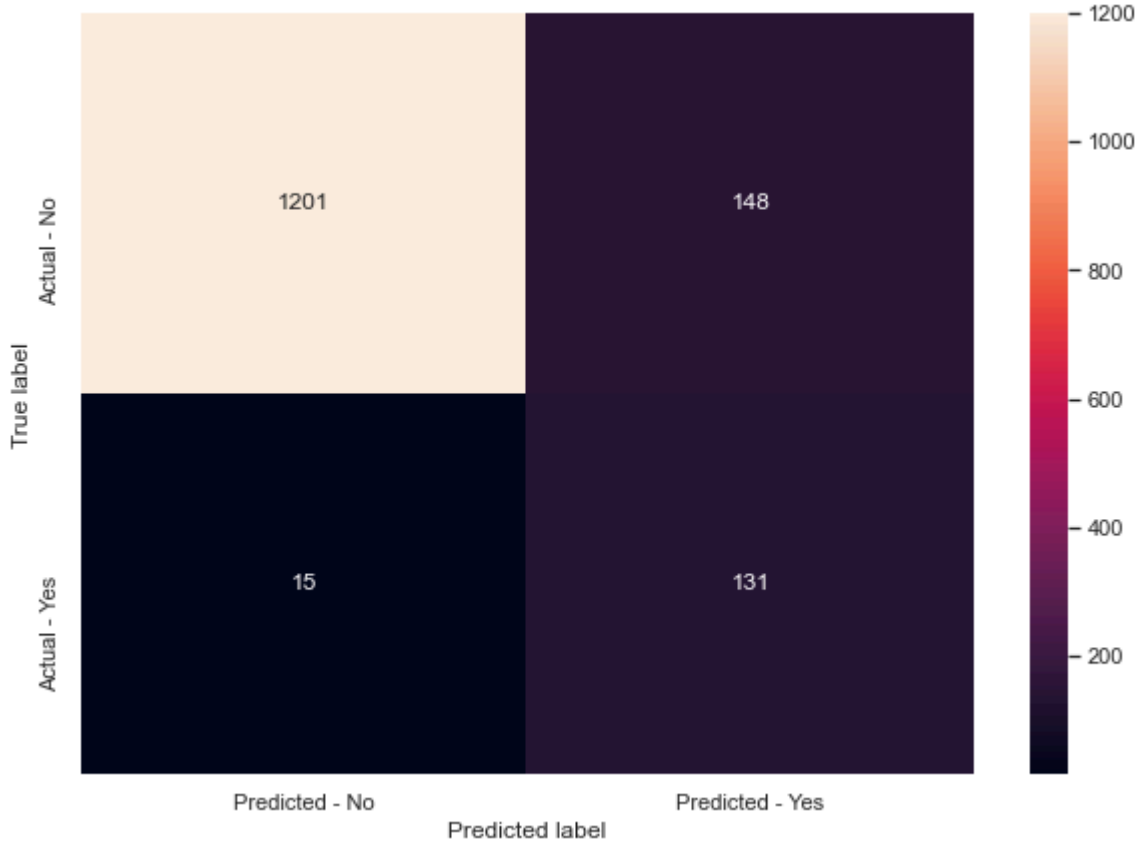
Optimal threshold using AUC-ROC curve

In [122…
```
# 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, lg6.predict(X_test5))

optimal_idx = np.argmax(tpr - fpr)
optimal_threshold_auc_roc = thresholds[optimal_idx]
print(f"Optimal Threshold: {optimal_threshold_auc_roc}")
```

```
Optimal Threshold: 0.08061059109804392
```

In [123…
```
# confusion matrix
make_confusion_matrix(
    lg6, "stats", X_test5, y_test, threshold=optimal_threshold_auc_roc
)

# checking model performance
scores_LR = get_metrics_score(
    lg6,
    "stats",
    X_train5,
    X_test5,
    y_train,
    y_test,
    threshold=optimal_threshold_auc_roc,
    roc=True,
)
```

```
Accuracy on training set : 0.8790
Accuracy on test set : 0.8910

Recall on training set :0.9061
Recall on test set : 0.8973

Precision on training set : 0.4333
Precision on test set : 0.4695

F1 on training set : 0.5863
F1 on test set : 0.6165

ROC-AUC Score on training set : 0.8911
ROC-AUC Score on test set : 0.8938
```

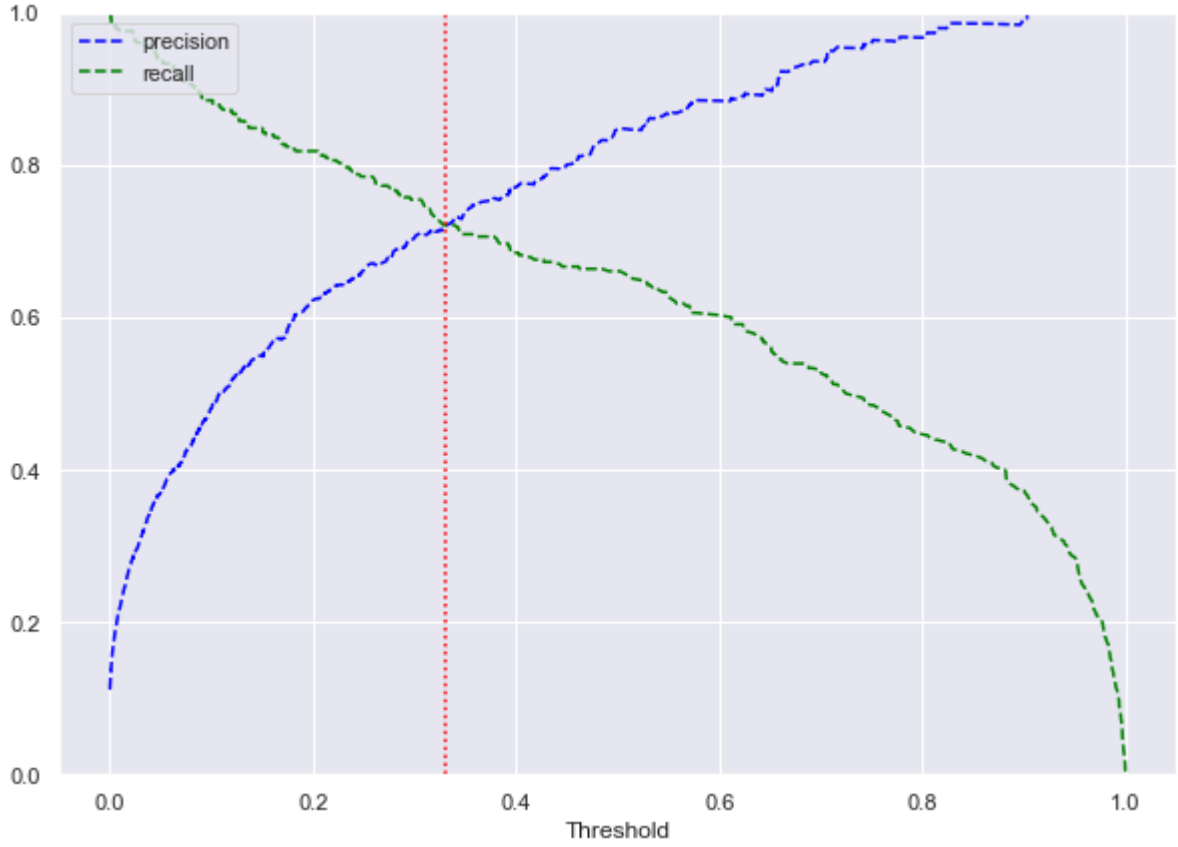


- Recall of model has increased but the other metrics have reduced.
- Threshold is 0.08, this mean everything over that is considered 1 (accept Personal Loan), it reduces de False Negatives incriasing Recall, but is increasing False Positives.
- This low threshold doesn't seems to be a good metric.

Let's use Precision-Recall curve and see if we can find a better threshold

In [124…

```
y_scores = lg6.predict(X_train5)
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.axvline(0.33, color="r", linestyle="dotted")


plt.figure(figsize=(10, 7))
plot_prec_recall_vs_tresh(prec, rec, tre)
plt.show()
```

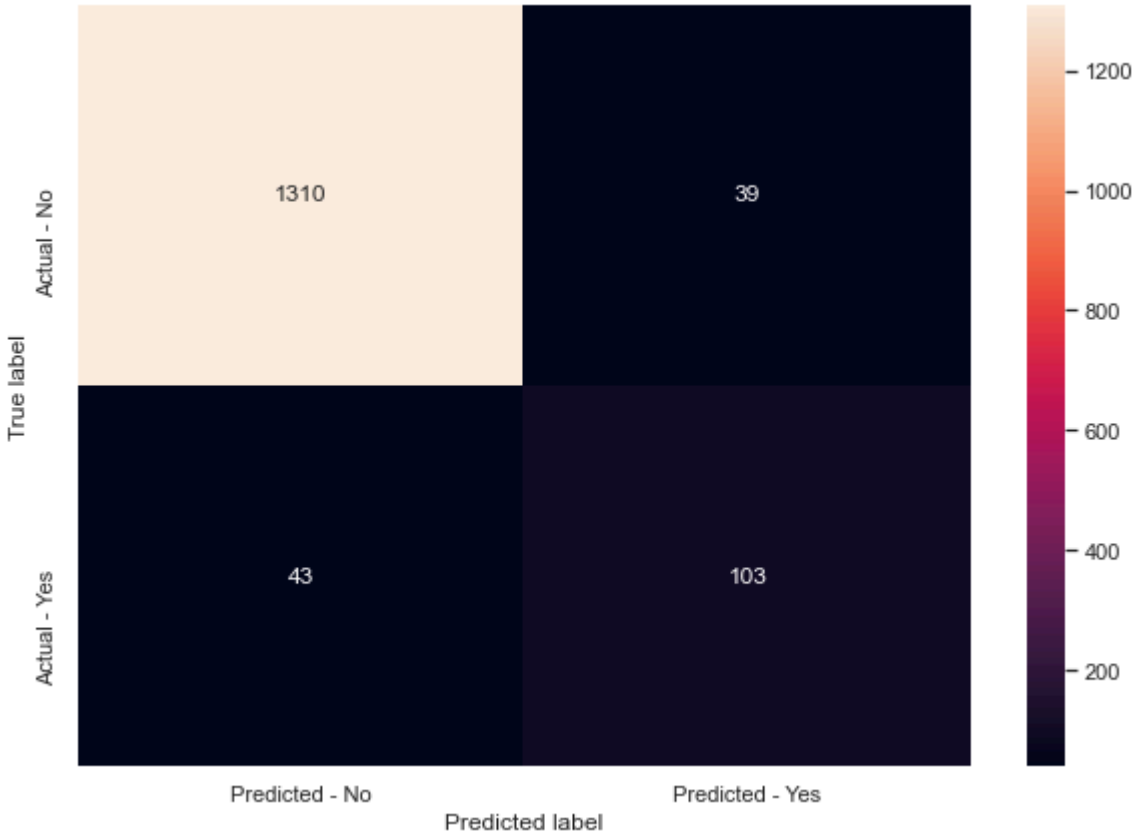


- At the threshold of 0.33, we get balanced recall and precision.

In [125…

```
optimal_threshold_curve = 0.33

# confusion matrix
make_confusion_matrix(lg6, "stats", X_test5, y_test, threshold=optimal_thresh

# checking model performance
scores_LR = get_metrics_score(
    lg6,
    "stats",
    X_train5,
    X_test5,
    y_train,
    y_test,
    threshold=optimal_threshold_curve,
    roc=True,
)
```

```
Accuracy on training set : 0.9470
Accuracy on test set : 0.9452

Recall on training set :0.7212
Recall on test set : 0.7055

Precision on training set : 0.7190
Precision on test set : 0.7254

F1 on training set : 0.7201
F1 on test set : 0.7153

ROC-AUC Score on training set : 0.8459
ROC-AUC Score on test set : 0.8383
```



- There's a big change in the model performance.
- Default threshold is 0.5 with recall training 0.6212 and test 0.6027.
- Threshold as 0.33, the model performance as 0.7212 on training data and 0.7055 on test data.

Model Performance Summary

In [126…

```
# defining list of model
models = [lg6]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []
f1_train = []
f1_test = []

# looping through the models list to get the metrics score - Accuracy, Recall
for model in models:

    j = get_metrics_score(
        model, "stats", X_train5, X_test5, y_train, y_test, flag=False
    )
    k = get_metrics_score(
        model,
        "stats",
        X_train5,
        X_test5,
        y_train,
        y_test,
```

```
        threshold=optimal_threshold_auc_roc,
        flag=False,
    )
    l = get_metrics_score(
        model,
        "stats",
        X_train5,
        X_test5,
        y_train,
        y_test,
        threshold=optimal_threshold_curve,
        flag=False,
    )

    # intial logistic regression model
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
    f1_train.append(j[6])
    f1_test.append(j[7])

    # logistic regression with threshold = 0.82
    acc_train.append(k[0])
    acc_test.append(k[1])
    recall_train.append(k[2])
    recall_test.append(k[3])
    precision_train.append(k[4])
    precision_test.append(k[5])
    f1_train.append(k[6])
    f1_test.append(k[7])

    # logistic regression with threshold = 0.33
    acc_train.append(l[0])
    acc_test.append(l[1])
    recall_train.append(l[2])
    recall_test.append(l[3])
    precision_train.append(l[4])
    precision_test.append(l[5])
    f1_train.append(l[6])
    f1_test.append(l[7])
```

In [127…

```
comparison_frame = pd.DataFrame(
    {
        "Model": [
            "Logistic Regression Model - Statsmodels",
            "Logistic Regression - Optimal threshold =  0.08",
            "Logistic Regression - Optimal threshold =  0.33",
        ],
        "Train_Accuracy": acc_train,
        "Test_Accuracy": acc_test,
        "Train Recall": recall_train,
        "Test Recall": recall_test,
        "Train Precision": precision_train,
        "Test Precision": precision_test,
        "Train F1": f1_train,
        "Test F1": f1_test,
    }
)
```

```
comparison_frame
```

Out[127…

	Model	Train_Accuracy	Test_Accuracy	Train Recall	Test Recall	Train Precision	Test Precision	Tra
0	Logistic Regression Model - Statsmodels	0.956709	0.945819	0.660606	0.602740	0.848249	0.792793	0.74
1	Logistic Regression - Optimal threshold = 0.08	0.879014	0.890970	0.906061	0.897260	0.433333	0.469534	0.58
2	Logistic Regression - Optimal threshold = 0.33	0.946961	0.945151	0.721212	0.705479	0.719033	0.725352	0.72

Conclusion

- Our Logistic Regression predictive model that can be used by AllLife bank to find the segment of customers who will buy a personal loan with a `Recall` of `0.7054` on the test set and formulate devise campaigns accordingly.
- Threshold 0.08 is to low, which imply in high False Positives, we don't wanna target wrong customers, we need a balanced Recall and Precision, even Recall is more important for us.
- Significant predictors can be checked on Statsmodels - Logistic Regression.
- Coefficient of some levels of Income, Family size, CCAvg (Average spending on credit cards per month), Education level and CD_Accouting (Customers with CD accounti with the bank) are positive an increase in these will lead to increase in chances of a person accepting a Personal Loan.
- Coefficient of Onile, CreditCard, Securities_Account are negative increase in these will lead to decrease in chances of a person accepting a Personal Loan.

Decision Tree

Data Preparation

In [128…

```
dummy_data = pd.get_dummies(df, drop_first=True)
dummy_data.head(5)
```

Out[128…

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_
0	25	1	49	4	1.6	1	0	0	
1	45	19	34	3	1.5	1	0	0	
2	39	15	11	1	1.0	1	0	0	

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_
3	35	9	100	1	2.7	2	0	0	
4	35	8	45	4	1.0	2	0	0	

Split Data

In [129…
```
# Spliting the data into Independente (X) and dependent(y) set:
X = dummy_data.drop("Personal_Loan", axis=1)
y = dummy_data["Personal_Loan"]
```

In [130…
```
# Spliting the data into training and test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
```

In [131…
```
# Checking shape of set
print("Shape of training set:", X_train.shape)
print("Shape of training set:", y_train.shape)
print("=-" * 17)
print("Shape of test set:", X_test.shape)
print("Shape of test set:", y_test.shape)
```

```
Shape of training set: (3488, 13)
Shape of training set: (3488,)
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Shape of test set: (1495, 13)
Shape of test set: (1495,)
```

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 {0:0.09,1:0.90} 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.

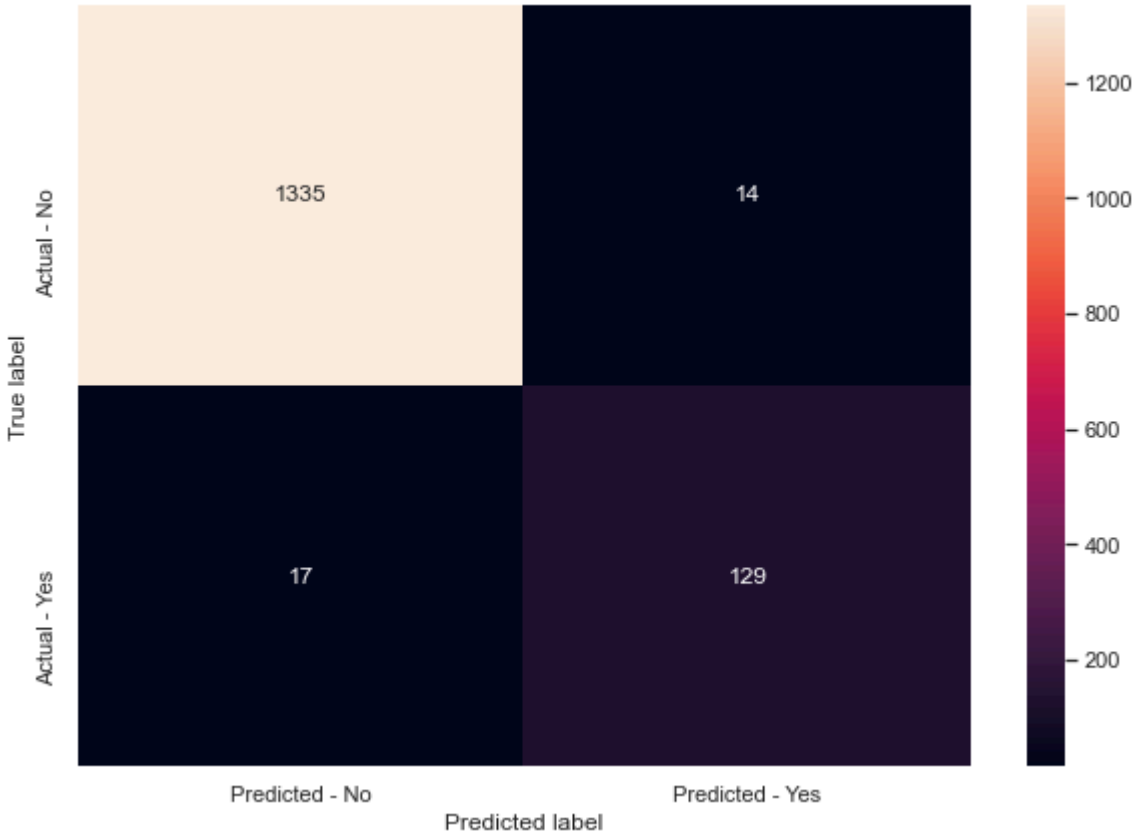
In [132…
```
# measure the quality of a split, using 'Gini' once is faster to compute and
model = DecisionTreeClassifier(
    criterion="gini", class_weight={0: 0.095, 1: 0.905}, random_state=1
)
```

In [133…
```
model.fit(X_train, y_train)
```

Out[133…
```
DecisionTreeClassifier(class_weight={0: 0.095, 1: 0.905}, random_state=1)
```

In [134…
```
def make_confusion_matrix(model, y_actual):
    """
    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() /
    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=True, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

In [135…
```
make_confusion_matrix(model, y_test)
```



In [136…
```
y_train.value_counts(1)
```

Out[136…
```
0    0.90539
1    0.09461
Name: Personal_Loan, dtype: float64
```

We only have 9.5% of positive classes, so if our model marks each sample as negative, then also we'll get 90.5% accuracy, hence accuracy is not a good metric to evaluate here.

Insights:

- **True Positives:**
  - Reality: A customer accepted Personal Loan.
  - Model predicted: The customer will accept a Personal Loan.
  - Outcome: The model is good.
- **True Negatives:**
  - Reality: A customer did NOT accepted Personal Loan.
  - Model predicted: The customer will NOT accept a Personal Loan.
  - Outcome: The business is unaffected.
- **False Positives:**
  - Reality: A customer did NOT accepted Personal Loan.
  - Model predicted: The customer will accepted Personal Loan.
  - Outcome: The team which is targeting the potential customers will be wasting their effort on customers who will not accepted Personal Loan.
- **False Negatives:**
  - Reality: A customer accepted Personal Loan.
  - Model predicted: The customer will NOT accepted Personal Loan.
  - Outcome: The potential customer is missed by the marketing team, the team could have devise campaigns with better target marketing to increase the success ratio. (Earn more through the interest on loans will get affected.)

How to reduce losses?

- We can use accuracy but since the data is imbalanced it would not be the right metric to check the model performance.
- Therefore, `Recall` gives the ratio of True Positives to Actual Positives, so high Recall implies `low false negatives` , i.e. low chances of predicting customers that wont accept Personal Loan, when it actualy will.

In [137…

```
##  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 : {0:0.4f}".format(
            metrics.recall_score(y_train, pred_train)
        )
    )
    print(
        "Recall on test set : {0:0.4f}".format(metrics.recall_score(y_test, p
    )
    print()
    print(
        "Precision on training set : {0:0.4f}".format(
            metrics.precision_score(y_train, pred_train)
```

```
        )
    )
    print(
        "Precision on test set : {0:0.4f}".format(
            metrics.precision_score(y_test, pred_test)
        )
    )
```

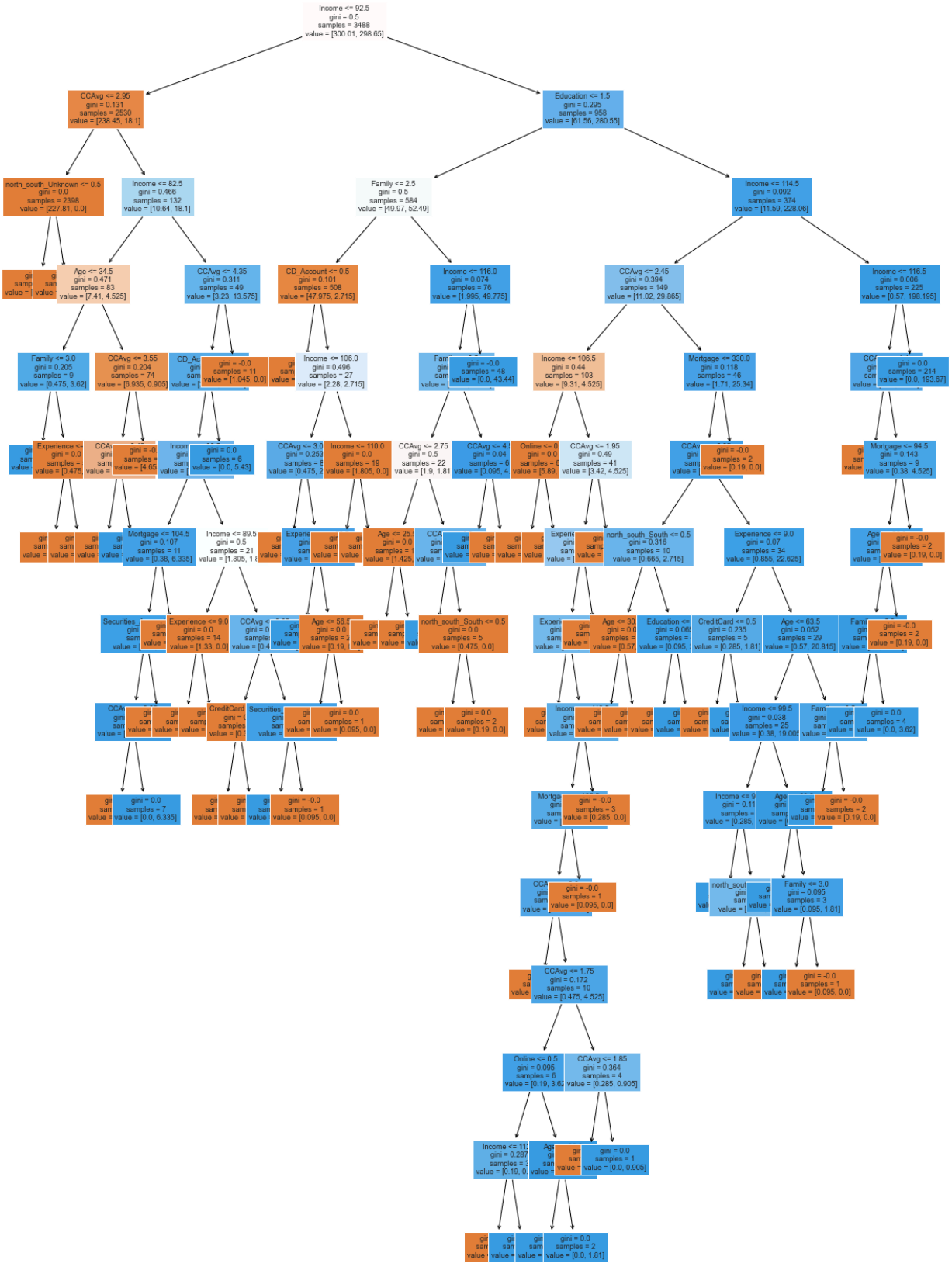
In [138…
```
get_recall_score(model)
```

```
Recall on training set : 1.0000
Recall on test set : 0.8836

Precision on training set : 1.0000
Precision on test set : 0.9021
```

- There is a disparity in the performance of the model on the training set and the test set, which suggests that the model is overfitting.

Visualizing the Decision Tree

In [139…
```
col_names = list(dummy_data.columns)
col_names.remove("Personal_Loan")
plt.figure(figsize=(20, 30))
out = tree.plot_tree(
    model,
    feature_names=col_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 [140…

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

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

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- north_south_Unknown <= 0.50
|   |   |   |--- weights: [226.19, 0.00] class: 0
|   |   |--- north_south_Unknown >  0.50
|   |   |   |--- weights: [1.61, 0.00] class: 0
```

```
|   |--- CCAvg >  2.95
|   |   |--- Income <= 82.50
|   |   |   |--- Age <= 34.50
|   |   |   |   |--- Family <= 3.00
|   |   |   |   |   |--- weights: [0.00, 3.62] class: 1
|   |   |   |   |--- Family >  3.00
|   |   |   |   |   |--- Experience <= 2.00
|   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |--- Experience >  2.00
|   |   |   |   |   |   |--- weights: [0.38, 0.00] class: 0
|   |   |   |--- Age >  34.50
|   |   |   |   |--- CCAvg <= 3.55
|   |   |   |   |   |--- CCAvg <= 3.45
|   |   |   |   |   |   |--- weights: [2.28, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  3.45
|   |   |   |   |   |   |--- weights: [0.00, 0.91] class: 1
|   |   |   |   |--- CCAvg >  3.55
|   |   |   |   |   |--- weights: [4.66, 0.00] class: 0
|   |   |--- Income >  82.50
|   |   |   |--- CCAvg <= 4.35
|   |   |   |   |--- CD_Account <= 0.50
|   |   |   |   |   |--- Income <= 83.50
|   |   |   |   |   |   |--- Mortgage <= 104.50
|   |   |   |   |   |   |   |--- Securities_Account <= 0.50
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 6.34] class: 1
|   |   |   |   |   |   |   |--- Securities_Account >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Mortgage >  104.50
|   |   |   |   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |   |   |   |--- Income >  83.50
|   |   |   |   |   |   |--- Income <= 89.50
|   |   |   |   |   |   |   |--- Experience <= 9.00
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  9.00
|   |   |   |   |   |   |   |   |--- weights: [1.23, 0.00] class: 0
|   |   |   |   |   |   |--- Income >  89.50
|   |   |   |   |   |   |   |--- CCAvg <= 3.35
|   |   |   |   |   |   |   |   |--- CreditCard <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CreditCard >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.29, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  3.35
|   |   |   |   |   |   |   |   |--- Securities_Account <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.81] class: 1
|   |   |   |   |   |   |   |   |--- Securities_Account >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |--- CD_Account >  0.50
|   |   |   |   |   |--- weights: [0.00, 5.43] class: 1
|   |   |   |--- CCAvg >  4.35
|   |   |   |   |--- weights: [1.04, 0.00] class: 0
|--- Income >  92.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- CD_Account <= 0.50
|   |   |   |   |--- weights: [45.69, 0.00] class: 0
|   |   |   |--- CD_Account >  0.50
|   |   |   |   |--- Income <= 106.00
|   |   |   |   |   |--- CCAvg <= 3.06
|   |   |   |   |   |   |--- weights: [0.29, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  3.06
|   |   |   |   |   |   |--- Experience <= 26.50
|   |   |   |   |   |   |   |--- weights: [0.00, 2.71] class: 1
|   |   |   |   |   |   |--- Experience >  26.50
|   |   |   |   |   |   |   |--- Age <= 56.50
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Age >  56.50
```

```
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |---  Income >  106.00
|   |   |   |   |   |   |--- Income <= 110.00
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Income >  110.00
|   |   |   |   |   |   |   |--- weights: [1.71, 0.00] class: 0
|   |   |---  Family >  2.50
|   |   |   |--- Income <= 116.00
|   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |--- CCAvg <= 2.75
|   |   |   |   |   |   |--- Age <= 25.50
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  25.50
|   |   |   |   |   |   |   |--- weights: [1.33, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  2.75
|   |   |   |   |   |   |--- CCAvg <= 4.20
|   |   |   |   |   |   |   |--- weights: [0.00, 1.81] class: 1
|   |   |   |   |   |   |--- CCAvg >  4.20
|   |   |   |   |   |   |   |--- north_south_South <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.29, 0.00] class: 0
|   |   |   |   |   |   |   |--- north_south_South >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |   |   |--- Family >  3.50
|   |   |   |   |   |--- CCAvg <= 4.50
|   |   |   |   |   |   |--- weights: [0.00, 4.53] class: 1
|   |   |   |   |   |--- CCAvg >  4.50
|   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |--- Income >  116.00
|   |   |   |   |--- weights: [0.00, 43.44] class: 1
|   |---  Education >  1.50
|   |   |--- Income <= 114.50
|   |   |   |--- CCAvg <= 2.45
|   |   |   |   |--- Income <= 106.50
|   |   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |   |--- weights: [3.33, 0.00] class: 0
|   |   |   |   |   |--- Online >  0.50
|   |   |   |   |   |   |--- weights: [2.57, 0.00] class: 0
|   |   |   |   |--- Income >  106.50
|   |   |   |   |   |--- CCAvg <= 1.95
|   |   |   |   |   |   |--- Experience <= 30.50
|   |   |   |   |   |   |   |--- Experience <= 3.50
|   |   |   |   |   |   |   |   |--- weights: [0.38, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  3.50
|   |   |   |   |   |   |   |   |--- Income <= 113.50
|   |   |   |   |   |   |   |   |   |--- Mortgage <= 105.50
|   |   |   |   |   |   |   |   |   |   |--- CCAvg <= 0.30
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class:
0
|   |   |   |   |   |   |   |   |   |   |--- CCAvg >  0.30
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |--- Mortgage >  105.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Income >  113.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.29, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  30.50
|   |   |   |   |   |   |   |   |--- weights: [0.85, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  1.95
|   |   |   |   |   |   |   |--- weights: [1.23, 0.00] class: 0
|   |   |   |--- CCAvg >  2.45
|   |   |   |   |--- Mortgage <= 330.00
|   |   |   |   |   |--- CCAvg <= 2.95
|   |   |   |   |   |   |--- north_south_South <= 0.50
|   |   |   |   |   |   |   |--- Age <= 30.50
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Age >  30.50
|   |   |   |   |   |   |   |   |--- weights: [0.47, 0.00] class: 0
|   |   |   |   |   |   |--- north_south_South >  0.50
|   |   |   |   |   |   |   |--- Education <= 2.50
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
```

```
|   |   |   |   |   |   |   |--- Education >  2.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.71] class: 1
|   |   |   |   |   |--- CCAvg >  2.95
|   |   |   |   |   |   |--- Experience <= 9.00
|   |   |   |   |   |   |   |--- CreditCard <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.29, 0.00] class: 0
|   |   |   |   |   |   |   |--- CreditCard >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.81] class: 1
|   |   |   |   |   |   |--- Experience >  9.00
|   |   |   |   |   |   |   |--- Age <= 63.50
|   |   |   |   |   |   |   |   |--- Income <= 99.50
|   |   |   |   |   |   |   |   |   |--- Income <= 94.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.62] class: 1
|   |   |   |   |   |   |   |   |   |--- Income >  94.50
|   |   |   |   |   |   |   |   |   |   |--- north_south_South <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.91] class:
1
|   |   |   |   |   |   |   |   |   |   |--- north_south_South >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.29, 0.00] class:
0
|   |   |   |   |   |   |   |   |--- Income >  99.50
|   |   |   |   |   |   |   |   |   |--- Age <= 60.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 12.67] class: 1
|   |   |   |   |   |   |   |   |   |--- Age >  60.00
|   |   |   |   |   |   |   |   |   |   |--- Family <= 3.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.81] class:
1
|   |   |   |   |   |   |   |   |   |   |--- Family >  3.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class:
0
|   |   |   |   |   |   |   |--- Age >  63.50
|   |   |   |   |   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.81] class: 1
|   |   |   |   |   |   |   |   |--- Family >  3.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |   |   |--- Mortgage >  330.00
|   |   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |--- Income >  114.50
|   |   |   |--- Income <= 116.50
|   |   |   |   |--- CCAvg <= 1.10
|   |   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |   |   |--- CCAvg >  1.10
|   |   |   |   |   |--- Mortgage <= 94.50
|   |   |   |   |   |   |--- Age <= 56.00
|   |   |   |   |   |   |   |--- Family <= 2.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.91] class: 1
|   |   |   |   |   |   |   |--- Family >  2.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 3.62] class: 1
|   |   |   |   |   |   |--- Age >  56.00
|   |   |   |   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |   |   |   |--- Mortgage >  94.50
|   |   |   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |   |--- Income >  116.50
|   |   |   |   |--- weights: [0.00, 193.67] class: 1
```

In [141…

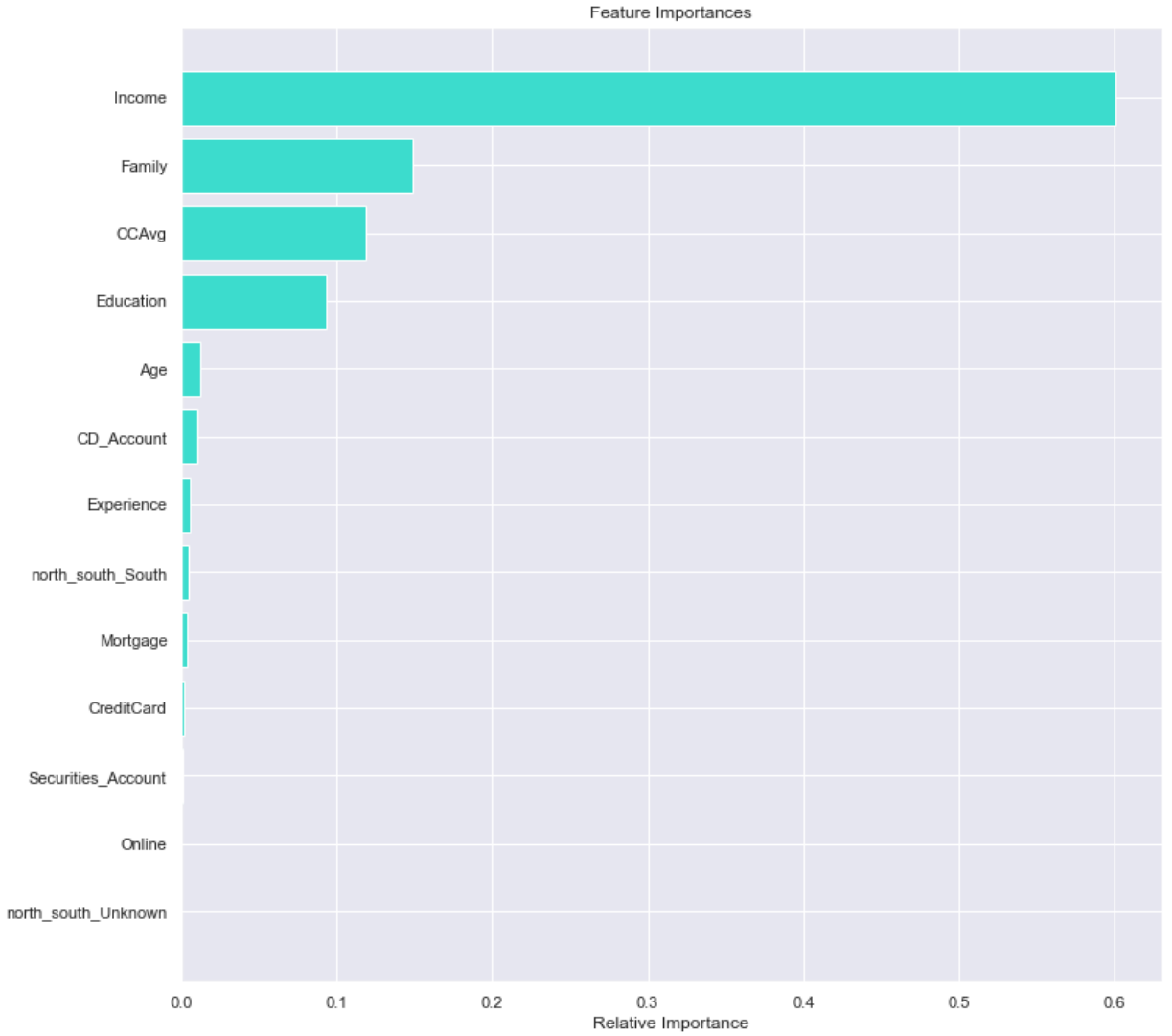
```
# importance of features in the tree building ( Gini importance )

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

```
                         Imp
Income              6.004061e-01
Family              1.489409e-01
CCAvg               1.183350e-01
```

```
Education            9.317900e-02
Age                  1.193163e-02
CD_Account           9.952551e-03
Experience           5.938187e-03
north_south_South    4.403989e-03
Mortgage             3.900220e-03
CreditCard           1.645210e-03
Securities_Account   1.210266e-03
Online               1.569712e-04
north_south_Unknown  7.604620e-16
```

In [142…

```
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="turquoise", align=
plt.yticks(range(len(indices)), [col_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Our Decision Tree is showing that `Income` is the most important variable for predicting `Personal_Loan`, followed for `Family`

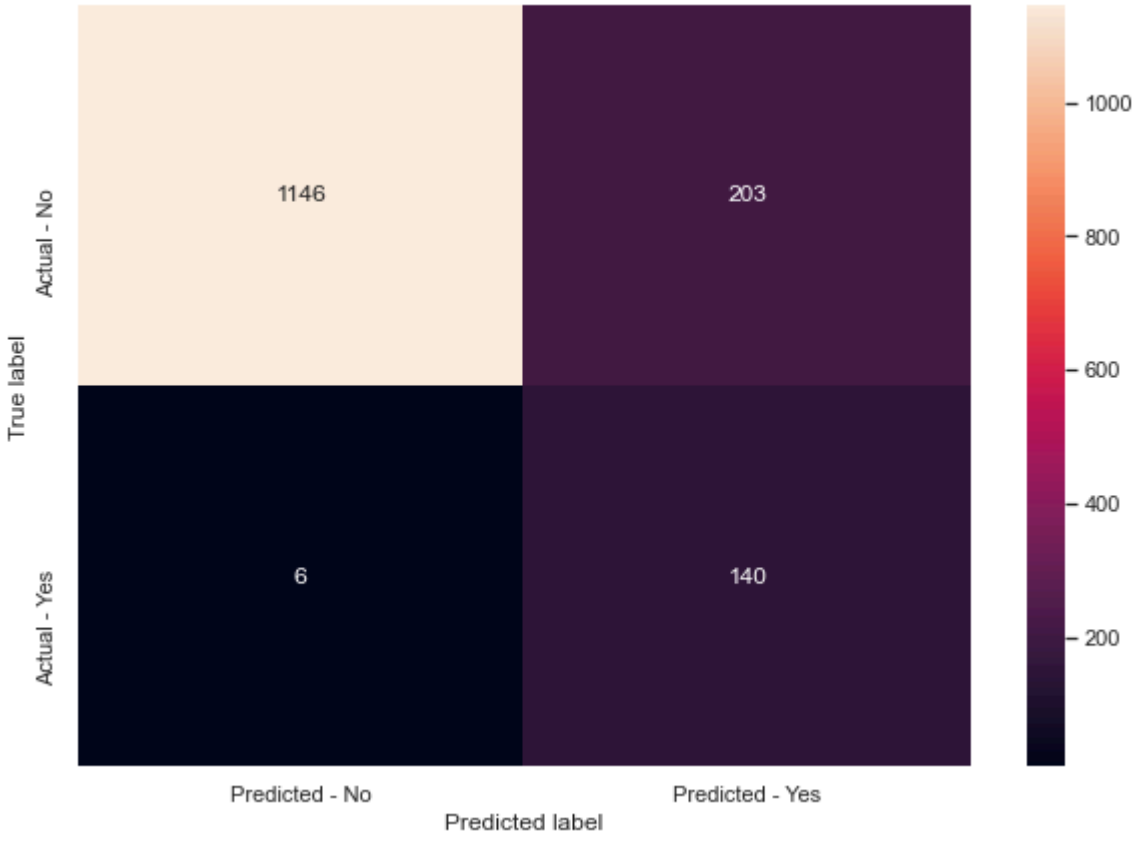
Reducing Overfitting

- Using GridSearch for Hyperparameter tuning of our tree model

In [143…
```
# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1, class_weight={0: 0.095, 1:

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

# 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[143…
```
DecisionTreeClassifier(class_weight={0: 0.095, 1: 0.905}, max_depth=5,
                       max_features='log2', min_impurity_decrease=1e-06,
                       random_state=1)
```

In [144…
```
make_confusion_matrix(estimator, y_test)
```



In [145…
```
get_recall_score(estimator)
```

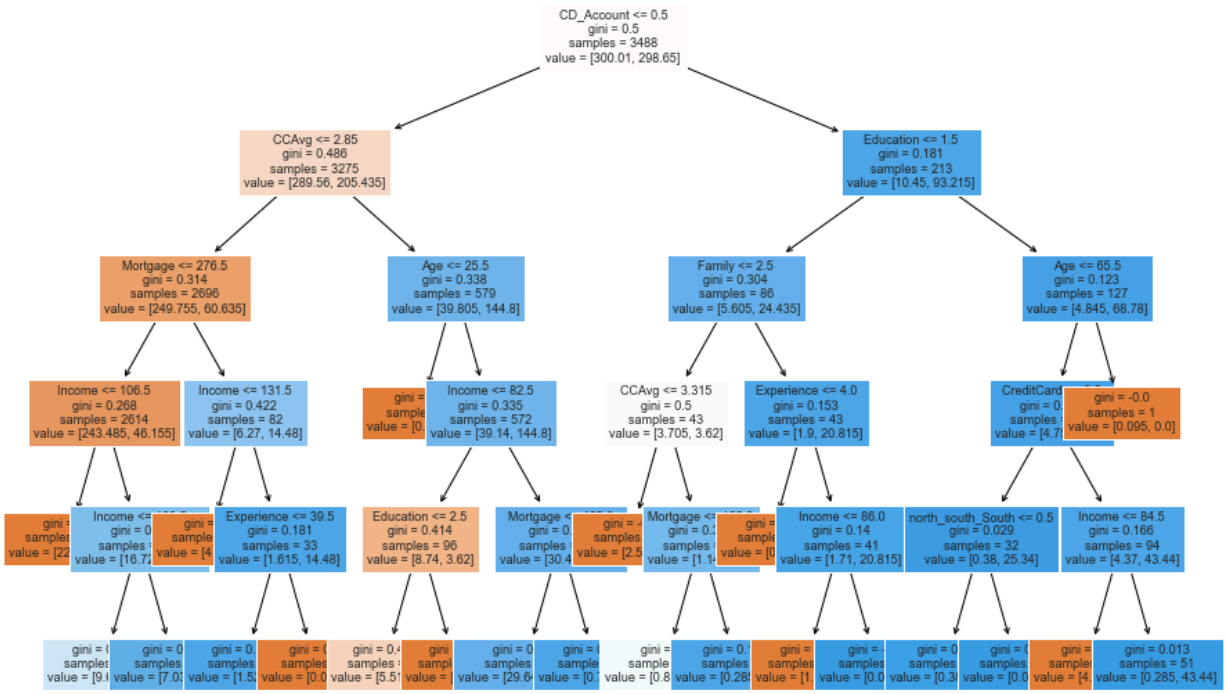
```
Recall on training set : 0.9879
Recall on test set : 0.9589
```

```
Precision on training set : 0.3804
Precision on test set : 0.4082
```

Visualizing the Decision Tree

In [146…

```
plt.figure(figsize=(15, 10))
out = tree.plot_tree(
    estimator,
    feature_names=col_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 [147…

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

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

```
|--- CD_Account <= 0.50
|   |--- CCAvg <= 2.85
|   |   |--- Mortgage <= 276.50
|   |   |   |--- Income <= 106.50
|   |   |   |   |--- weights: [226.76, 0.00] class: 0
|   |   |   |--- Income >  106.50
|   |   |   |   |--- Income <= 129.50
|   |   |   |   |   |--- weights: [9.69, 12.67] class: 1
|   |   |   |   |--- Income >  129.50
|   |   |   |   |   |--- weights: [7.03, 33.49] class: 1
|   |   |--- Mortgage >  276.50
|   |   |   |--- Income <= 131.50
```

```
|   |   |   |   |--- weights: [4.66, 0.00] class: 0
|   |   |   |--- Income >  131.50
|   |   |   |   |--- Experience <= 39.50
|   |   |   |   |   |--- weights: [1.52, 14.48] class: 1
|   |   |   |   |--- Experience >  39.50
|   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |--- CCAvg >  2.85
|   |   |--- Age <= 25.50
|   |   |   |--- weights: [0.66, 0.00] class: 0
|   |   |--- Age >  25.50
|   |   |   |--- Income <= 82.50
|   |   |   |   |--- Education <= 2.50
|   |   |   |   |   |--- weights: [5.51, 3.62] class: 0
|   |   |   |   |--- Education >  2.50
|   |   |   |   |   |--- weights: [3.23, 0.00] class: 0
|   |   |   |--- Income >  82.50
|   |   |   |   |--- Mortgage <= 425.00
|   |   |   |   |   |--- weights: [29.64, 133.04] class: 1
|   |   |   |   |--- Mortgage >  425.00
|   |   |   |   |   |--- weights: [0.76, 8.15] class: 1
|--- CD_Account >  0.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- CCAvg <= 3.31
|   |   |   |   |--- weights: [2.57, 0.00] class: 0
|   |   |   |--- CCAvg >  3.31
|   |   |   |   |--- Mortgage <= 162.00
|   |   |   |   |   |--- weights: [0.85, 0.91] class: 1
|   |   |   |   |--- Mortgage >  162.00
|   |   |   |   |   |--- weights: [0.29, 2.71] class: 1
|   |   |--- Family >  2.50
|   |   |   |--- Experience <= 4.00
|   |   |   |   |--- weights: [0.19, 0.00] class: 0
|   |   |   |--- Experience >  4.00
|   |   |   |   |--- Income <= 86.00
|   |   |   |   |   |--- weights: [1.71, 0.00] class: 0
|   |   |   |   |--- Income >  86.00
|   |   |   |   |   |--- weights: [0.00, 20.82] class: 1
|   |--- Education >  1.50
|   |   |--- Age <= 65.50
|   |   |   |--- CreditCard <= 0.50
|   |   |   |   |--- north_south_South <= 0.50
|   |   |   |   |   |--- weights: [0.38, 14.48] class: 1
|   |   |   |   |--- north_south_South >  0.50
|   |   |   |   |   |--- weights: [0.00, 10.86] class: 1
|   |   |   |--- CreditCard >  0.50
|   |   |   |   |--- Income <= 84.50
|   |   |   |   |   |--- weights: [4.09, 0.00] class: 0
|   |   |   |   |--- Income >  84.50
|   |   |   |   |   |--- weights: [0.29, 43.44] class: 1
|   |   |--- Age >  65.50
|   |   |   |--- weights: [0.10, 0.00] class: 0
```

In [148…

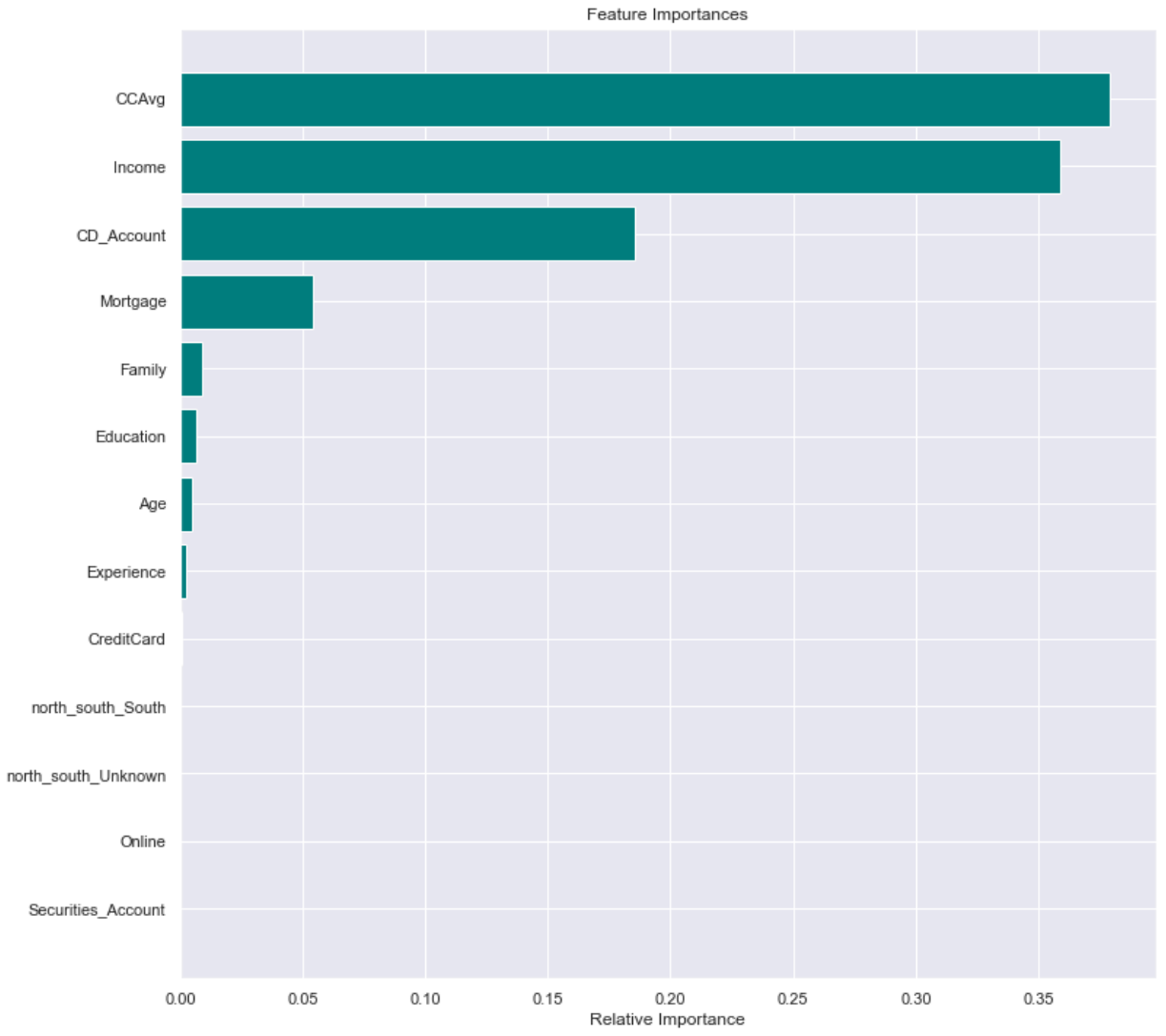
```
# importance of features in the tree building ( Gini importance )

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

```
                       Imp
CCAvg             0.378985
Income            0.358718
CD_Account        0.185161
Mortgage          0.054029
Family            0.009096
Education         0.006324
Age               0.004549
Experience        0.002195
CreditCard        0.000905
north_south_South 0.000038
```

```
Securities_Account   0.000000
Online               0.000000
north_south_Unknown  0.000000
```

- importance of features has increased

In [149…
```
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="teal", align="cent
plt.yticks(range(len(indices)), [col_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Pre-Prunning Decision Tree is showing that `CCAvg` is the most important variable for predicting `Personal_Loan`, followed by `Income`.
- The relative Importance has increased.

Cost Complexity Pruning

In [150…
```
clf = DecisionTreeClassifier(random_state=1, class_weight={0: 0.095, 1: 0.905
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

In [151…
```
pd.DataFrame(path)
```

Out[151…

	ccp_alphas	impurities
0	0.000000e+00	-2.063703e-16
1	2.642682e-19	-2.061060e-16
2	3.523576e-19	-2.057536e-16
3	9.513654e-19	-2.048023e-16
4	1.268487e-18	-2.035338e-16
5	1.673698e-18	-2.018601e-16
6	1.849877e-18	-2.000102e-16
7	2.517502e-18	-1.974927e-16
8	3.524503e-18	-1.939682e-16
9	4.933006e-18	-1.890352e-16
10	8.720850e-18	-1.803143e-16
11	7.427697e-17	-1.060374e-16
12	3.802290e-16	2.741917e-16
13	1.576534e-04	3.153068e-04
14	3.015483e-04	9.184035e-04
15	3.015483e-04	1.219952e-03
16	3.066457e-04	1.526598e-03
17	3.076379e-04	3.064787e-03
18	3.081339e-04	3.681055e-03
19	3.108494e-04	3.991904e-03
20	3.334680e-04	5.325776e-03
21	4.494353e-04	6.224647e-03
22	4.746980e-04	8.123439e-03
23	5.813934e-04	8.704832e-03
24	5.932354e-04	9.298068e-03
25	6.860888e-04	9.984157e-03
26	7.573515e-04	1.074151e-02
27	7.642021e-04	1.303411e-02
28	8.108854e-04	1.384500e-02
29	9.554530e-04	1.480045e-02
30	1.257001e-03	1.605745e-02
31	1.277645e-03	1.989039e-02
32	1.337207e-03	2.256481e-02
33	1.402807e-03	2.396761e-02
34	1.488688e-03	2.545630e-02

	ccp_alphas	impurities
35	1.584665e-03	2.704097e-02
36	1.613399e-03	3.188116e-02
37	2.429131e-03	3.431029e-02
38	2.789590e-03	3.709988e-02
39	3.665466e-03	4.076535e-02
40	4.444269e-03	4.520962e-02
41	4.796197e-03	5.480201e-02
42	9.712030e-03	7.422607e-02
43	3.381590e-02	1.080420e-01
44	5.840640e-02	2.248548e-01
45	2.751426e-01	4.999974e-01

In [152…

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(
    ccp_alphas[:-1],
    impurities[:-1],
    marker="H",
    drawstyle="steps-post",
    color="salmon",
)
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()
```

In [153…

```
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)
```

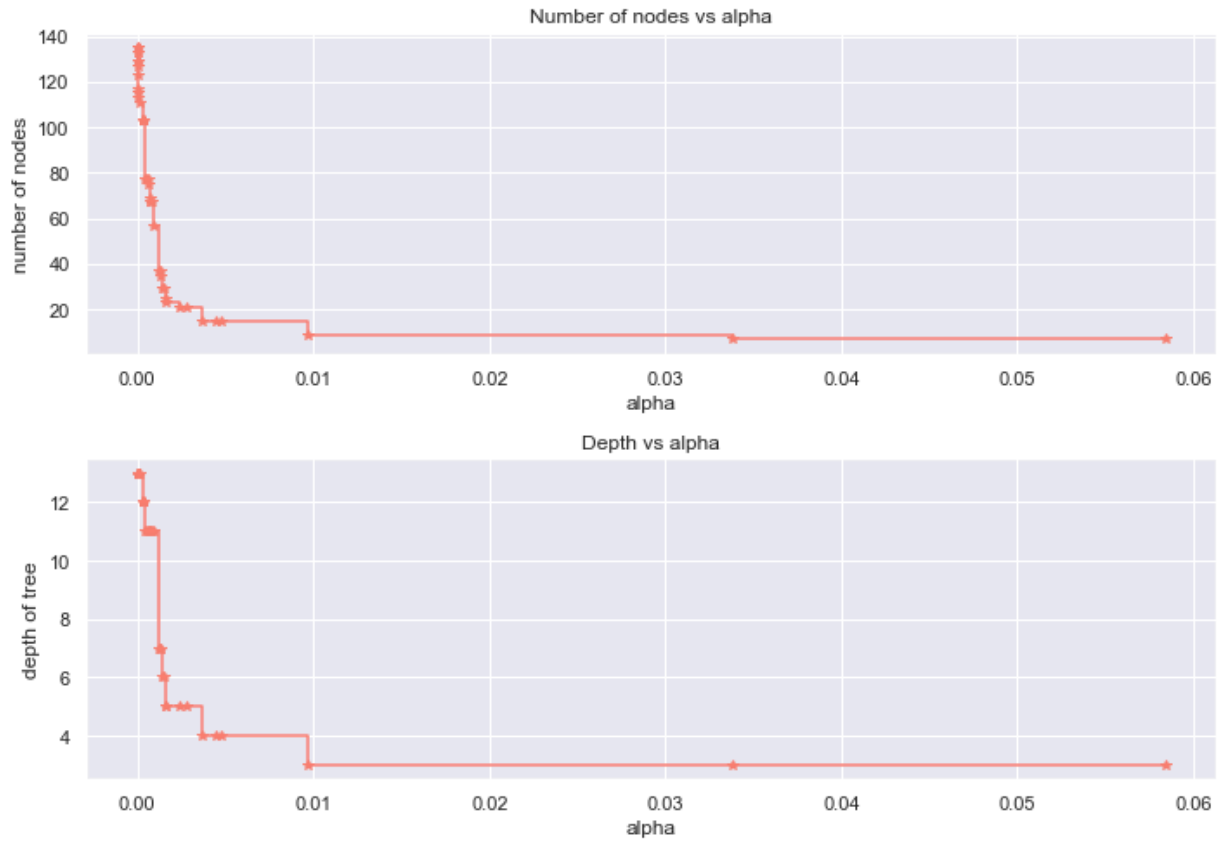
```
print(
    "Number of nodes in the last tree is: {} with ccp_alpha:{}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

```
Number of nodes in the last tree is: 1 with ccp_alpha:0.27514264456786364
```

- The number of nodes and tree depth decreases as alpha an inputity increases.

In [154…

```
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="*", drawstyle="steps-post", color
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="*", drawstyle="steps-post", color="salm
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```



In [155…

```
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)
```

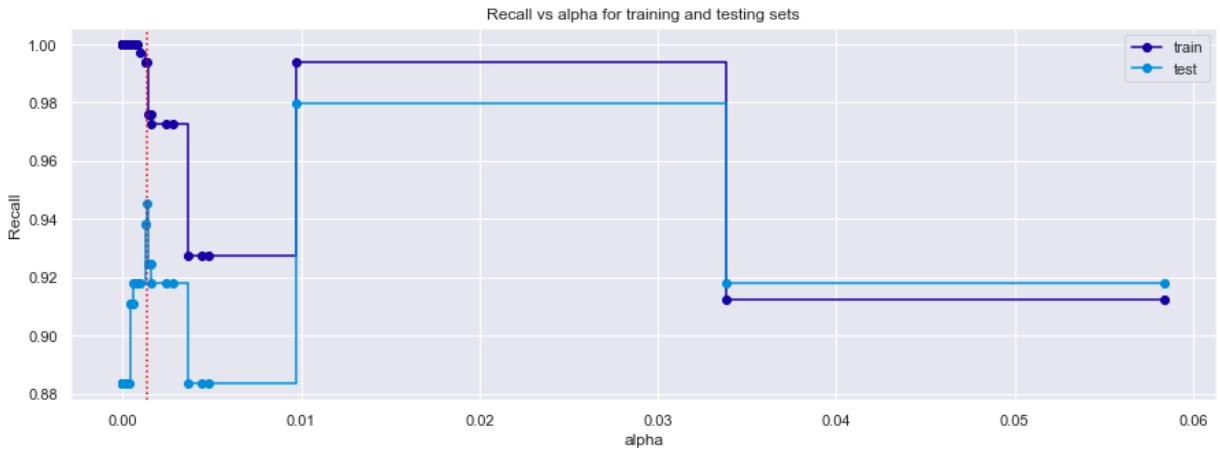
In [156…

```
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 [157…
```
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 [158…
```
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-p
plt.axvline(0.00135, color="r", linestyle="dotted")
ax.legend()
plt.show()
```



Maximum value of Recall is around 0.010 alpha, but if we choose decision tree will only have 3 decision root and we would lose the business rules, and our type I erro gonna be high (precission is low). Instead we can choose alpha 0.00135 retaining information and getting a higher recall and a good precision.

- We wanna a better recall but also we don't wanna a low precision, meaning that we predict a lot of customers as Yes and at the end they say NO, we not gonna be targeting the best customer segments.

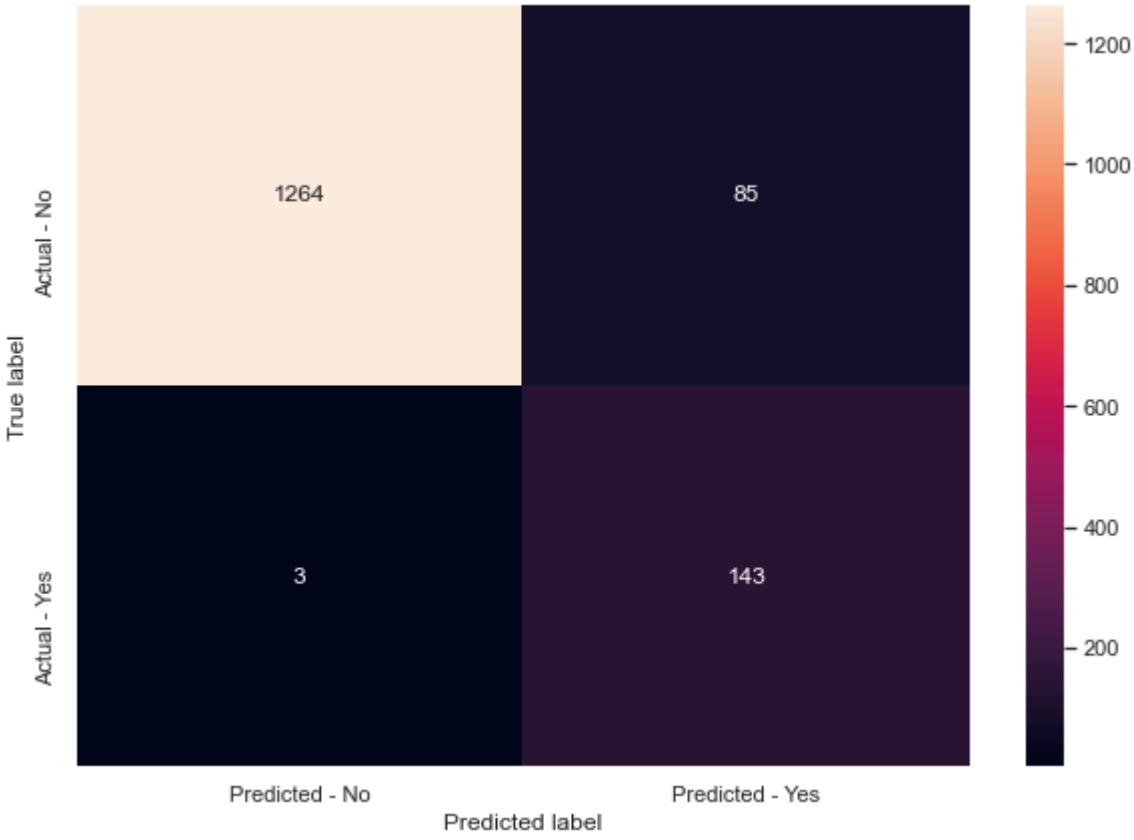
In [159…
```
# creating the model where we get highest train and test recall
index_best_model = np.argmax(recall_test)
best_model = clfs[index_best_model]
print(best_model)
```

```
DecisionTreeClassifier(ccp_alpha=0.009712030069120282,
                       class_weight={0: 0.15, 1: 0.85}, random_state=1)
```

In [160…
```
best_model.fit(X_train, y_train)
```

Out[160…
```
DecisionTreeClassifier(ccp_alpha=0.009712030069120282,
                       class_weight={0: 0.15, 1: 0.85}, random_state=1)
```

In [161…
```
make_confusion_matrix(best_model, y_test)
```



In [162…
```
get_recall_score(best_model)
```

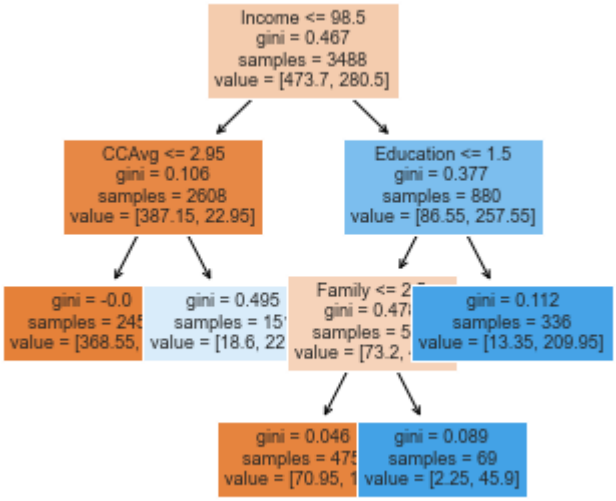
```
Recall on training set : 0.9939
Recall on test set : 0.9795

Precision on training set : 0.5899
Precision on test set : 0.6272
```

- This model has the better Recall, on the other hand, Precision is low.
- Low precision is meaning that the model is predicting a lot of YES, customer will accept, but at the end, they actual says NO.
- We need to find a better balence for our model.

Visualizing the Decision Tree

In [163…
```
plt.figure(figsize=(5, 5))

out = tree.plot_tree(
    best_model,
    feature_names=col_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()
```



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

In [164…
```
# importance of features in the tree building ( The importance of a feature i
# (normalized) total reduction of the 'criterion' brought by that feature. It

print(
    pd.DataFrame(
        best_model.feature_importances_, columns=["Imp"], index=X_train.colum
    ).sort_values(by="Imp", ascending=False)
)
```

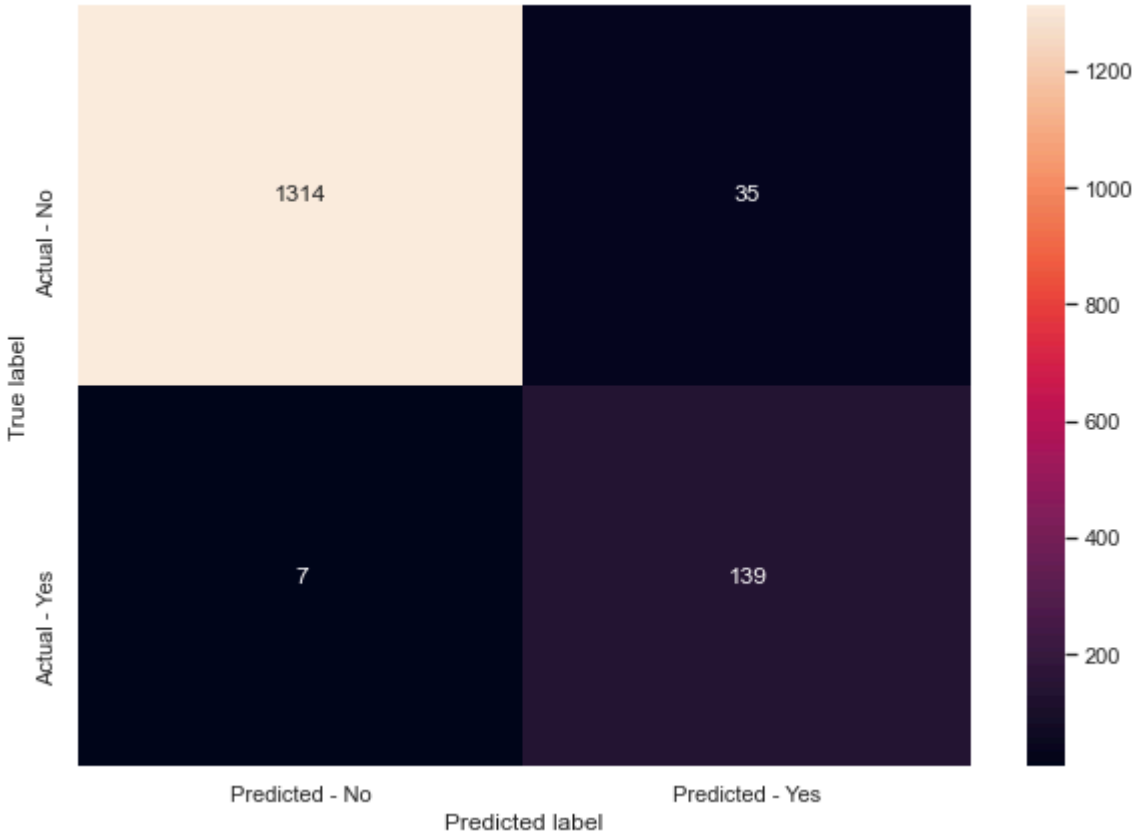
```
                       Imp
Income            0.600021
Family            0.167430
Education         0.156372
CCAvg             0.076177
Age               0.000000
Experience        0.000000
Mortgage          0.000000
Securities_Account  0.000000
CD_Account        0.000000
Online            0.000000
CreditCard        0.000000
north_south_South  0.000000
north_south_Unknown  0.000000
```

Creating model with 0.00135 ccp_alpha

In [165…
```
best_model2 = DecisionTreeClassifier(
    ccp_alpha=0.00135, class_weight={0: 0.095, 1: 0.905}, random_state=1
)
best_model2.fit(X_train, y_train)
```

Out[165…
```
DecisionTreeClassifier(ccp_alpha=0.00135, class_weight={0: 0.095, 1: 0.905},
                       random_state=1)
```

In [166…
```
make_confusion_matrix(best_model2, y_test)
```



In [167…
```
get_recall_score(best_model2)
```

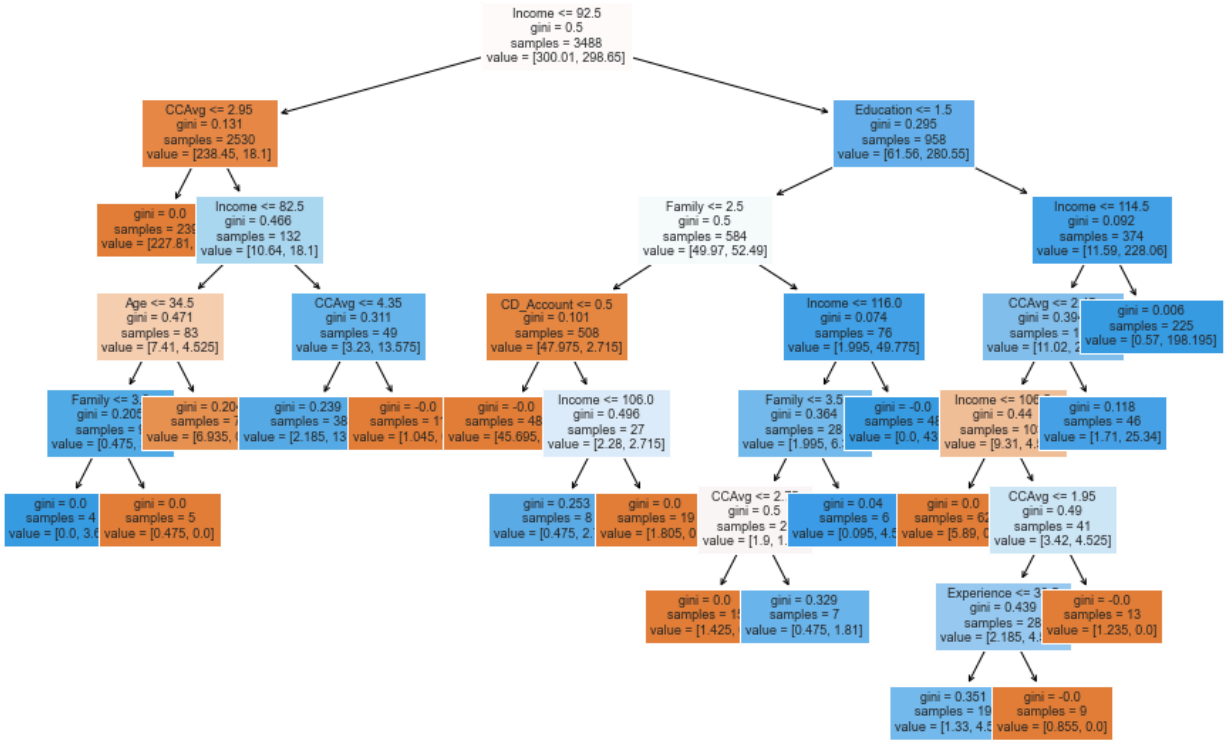
```
Recall on training set : 0.9970
Recall on test set : 0.9521

Precision on training set : 0.8204
Precision on test set : 0.7989
```

Observation:

- This seems to be the mest model.
- High Recall with a good Precision, meaning our model is performing good and being precises on the segment of customers and Feature Importances.

Visualizing the Decision Tree

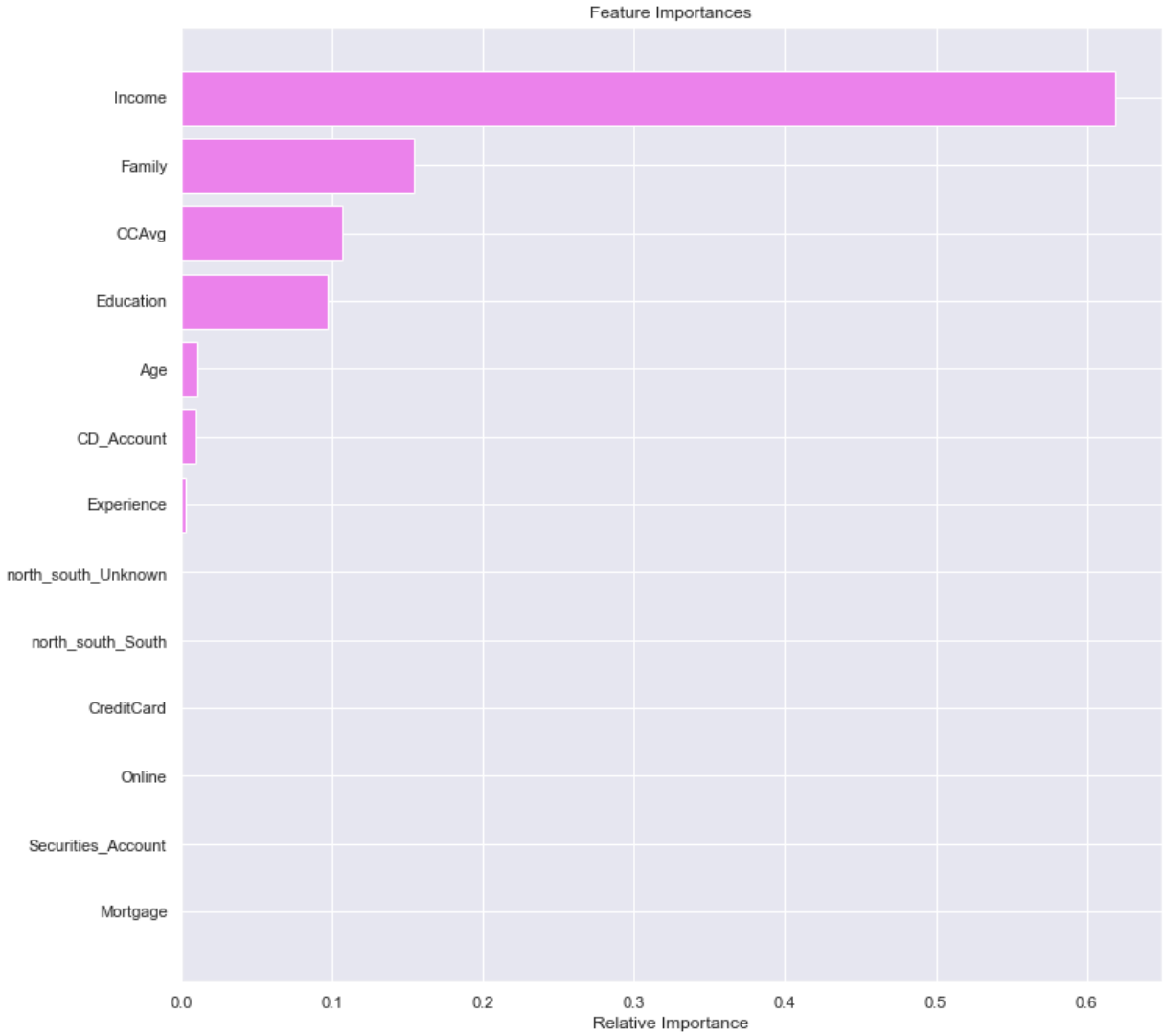
In [168…
```
plt.figure(figsize=(15, 10))

out = tree.plot_tree(
    best_model2,
    feature_names=col_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 [169…
```
# importance of features in the tree building ( The importance of a feature i
# (normalized) total reduction of the 'criterion' brought by that feature. It

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

	Imp
Income	0.618469
Family	0.154145
CCAvg	0.106900
Education	0.096941
Age	0.011119
CD_Account	0.009309
Experience	0.003118
Mortgage	0.000000
Securities_Account	0.000000
Online	0.000000
CreditCard	0.000000
north_south_South	0.000000
north_south_Unknown	0.000000

In [170…
```
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="ce
plt.yticks(range(len(indices)), [col_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Comparing all the decision tree models

In [171…

```
comparison_frame = pd.DataFrame(
    {
        "Model": [
            "Initial decision tree model",
            "Decision treee with hyperparameter tuning",
            "Decision tree with post-pruning",
            "Decision tree with post-pruning - alpha 0.00135",
        ],
        "Train_Recall": [1, 0.9879, 0.9939, 0.9970],
        "Test_Recall": [0.8836, 0.9589, 0.9795, 0.9521],
        "Train_Precision": [1, 0.3804, 0.5899, 0.8209],
        "Test_Precision": [0.9021, 0.4082, 0.6204, 0.7989],
    }
)
comparison_frame
```

Out[171…

	Model	Train_Recall	Test_Recall	Train_Precision	Test_Precision
0	Initial decision tree model	1.0000	0.8836	1.0000	0.9021
1	Decision treee with hyperparameter tuning	0.9879	0.9589	0.3804	0.4082
2	Decision tree with post-pruning	0.9939	0.9795	0.5899	0.6204
3	Decision tree with post-pruning - alpha 0.00135	0.9970	0.9521	0.8209	0.7989

Conclusion

- Our Decision Tree predictive model that can be used by AllLife bank to find the segment of customers who will buy a personal loan with a `Recall` of `0.9521` | alpha 0.00135 on the test set and formulate devise campaigns accordingly.
- Post-pruning with best model, has low precision which imply in high False Positives, we don't wanna target wrong customers, we need a balanced Recall and Precision, even though Recall is more important for us.
- Feature Importances are Income, Family size, CCAvg (Average spending on credit cards per month), Education level and Age these are the Customer Segmentation that will lead to increase in chances of a person accepting a Personal Loan.

Perform an Exploratory Data Analysis on the incorrectly predicted data

As our Target Column(Personal Loan) is in middle of dataframe, we gonna drop it and appended at the end, so we can compar it to Personal Loan Predict

In [172…
```
Personal_Loan = df["Personal_Loan"]
df.drop(["Personal_Loan"], axis=1, inplace=True)
df["Personal_Loan"] = Personal_Loan
df.head()
```

Out[172…

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_Ac
0	25	1	49	4	1.6	1	0	1	
1	45	19	34	3	1.5	1	0	1	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	
4	35	8	45	4	1.0	2	0	0	

In [173…
```
# Spliting data

X = df.drop(["Personal_Loan"], axis=1)
y = df["Personal_Loan"]

# Creating dummies for categorical variable
X = pd.get_dummies(X, drop_first=True)
```

In [174…
```
X.head()
```

Out[174…

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_Ac
0	25	1	49	4	1.6	1	0	1	
1	45	19	34	3	1.5	1	0	1	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_Ac
4	35	8	45	4	1.0	2	0	0	

In [175…
```
# Using our best_model (Decision Tree), to predict Personal Loan
y_pred = best_model2.predict(X)
```

In [176…
```
# Adding the predict value in our dataframe, so we can compare predict Vs rea
df["y_predict"] = y_pred
df.head()
```

Out[176…

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_Ac
0	25	1	49	4	1.6	1	0	1	
1	45	19	34	3	1.5	1	0	1	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	
4	35	8	45	4	1.0	2	0	0	

In [177…
```
# Creating a new data with only predictions different from real result, so th
df_error = df[(df["y_predict"] != df["Personal_Loan"])]

# Creating a columns to identify the type of error
df_error["type_error"] = df_error["Personal_Loan"] - df_error["y_predict"]

df_error.head()
```

Out[177…

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_
3	35	9	100	1	2.7	2	0	0	
12	48	23	114	2	3.8	3	0	1	
73	41	16	85	1	4.0	3	0	0	
85	27	2	109	4	1.8	3	0	0	
123	37	13	84	1	3.6	2	0	1	

In [178…
```
# Replacing -1 as FN and 1 as FP
df_error.type_error = df_error.type_error.apply(lambda x: "FN" if x == 1 else
df_error.head()
```

Out[178…

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_
3	35	9	100	1	2.7	2	0	0	
12	48	23	114	2	3.8	3	0	1	
73	41	16	85	1	4.0	3	0	0	
85	27	2	109	4	1.8	3	0	0	
123	37	13	84	1	3.6	2	0	1	

In [179…
```
df_error.shape
```

Out[179…
```
(115, 15)
```

In [180…
```
# We have low number of error (2.3%), High recall and a good precision
df_error.type_error.value_counts()
```

Out[180…
```
FP    107
FN      8
Name: type_error, dtype: int64
```

In [181…
```
# Lets check the sumarry for error type FP
df_error[df_error["type_error"] == "FP"].describe().T
```

Out[181…

	count	mean	std	min	25%	50%	75%	max
Age	107.0	44.794393	11.280923	27.0	35.0	43.0	54.0	65.0
Experience	107.0	19.345794	11.336286	2.0	9.0	18.0	28.0	41.0
Income	107.0	98.504673	11.190776	81.0	88.5	98.0	110.0	115.0
Family	107.0	2.102804	1.181098	1.0	1.0	2.0	3.0	4.0
CCAvg	107.0	2.987850	1.168188	0.1	2.2	3.2	3.8	4.9
Education	107.0	2.242991	0.822343	1.0	2.0	2.0	3.0	3.0
Mortgage	107.0	73.074766	114.863553	0.0	0.0	0.0	165.5	408.0
Securities_Account	107.0	0.177570	0.383949	0.0	0.0	0.0	0.0	1.0
CD_Account	107.0	0.102804	0.305132	0.0	0.0	0.0	0.0	1.0
Online	107.0	0.654206	0.477865	0.0	0.0	1.0	1.0	1.0
CreditCard	107.0	0.280374	0.451296	0.0	0.0	0.0	1.0	1.0
Personal_Loan	107.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
y_predict	107.0	1.000000	0.000000	1.0	1.0	1.0	1.0	1.0

In [182…
```
# Lets check the sumarry for error type FN
df_error[df_error["type_error"] == "FN"].describe().T
```

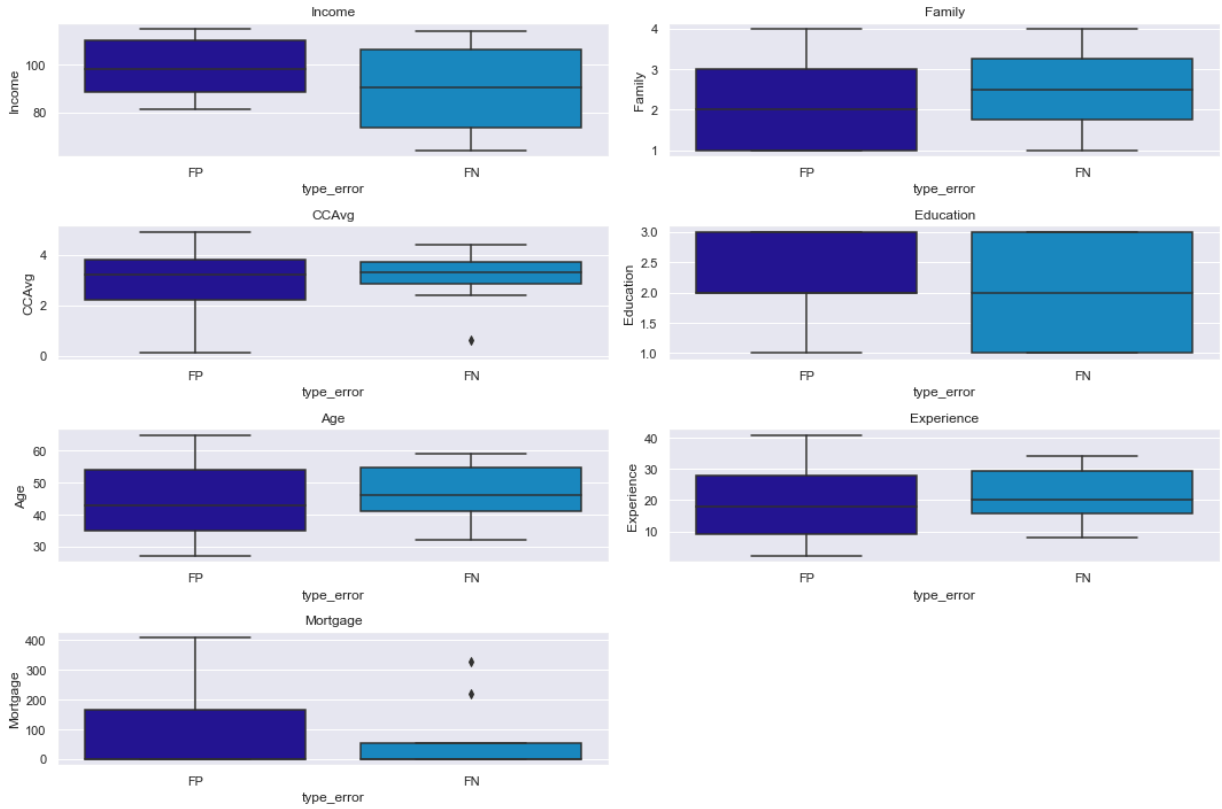
Out[182…

	count	mean	std	min	25%	50%	75%	max
Age	8.0	46.625	9.738546	32.0	41.25	46.0	54.75	59.0
Experience	8.0	21.375	9.318760	8.0	15.75	20.0	29.50	34.0
Income	8.0	90.000	19.957097	64.0	73.50	90.5	106.25	114.0
Family	8.0	2.500	1.195229	1.0	1.75	2.5	3.25	4.0
CCAvg	8.0	3.050	1.152637	0.6	2.85	3.3	3.70	4.4
Education	8.0	2.000	0.925820	1.0	1.00	2.0	3.00	3.0
Mortgage	8.0	68.500	129.962632	0.0	0.00	0.0	55.25	327.0
Securities_Account	8.0	0.000	0.000000	0.0	0.00	0.0	0.00	0.0
CD_Account	8.0	0.000	0.000000	0.0	0.00	0.0	0.00	0.0
Online	8.0	0.750	0.462910	0.0	0.75	1.0	1.00	1.0

	count	mean	std	min	25%	50%	75%	max
CreditCard	8.0	0.125	0.353553	0.0	0.00	0.0	0.00	1.0
Personal_Loan	8.0	1.000	0.000000	1.0	1.00	1.0	1.00	1.0
y_predict	8.0	0.000	0.000000	0.0	0.00	0.0	0.00	0.0

In [183…

```
cols = df_error[
    ["Income", "Family", "CCAvg", "Education", "Age", "Experience", "Mortgage
].columns.tolist()
plt.figure(figsize=(15, 10))

for i, variable in enumerate(cols):
    plt.subplot(4, 2, i + 1)
    sns.boxplot(df_error["type_error"], df_error[variable])
    plt.tight_layout()
    plt.title(variable)
plt.show()
```

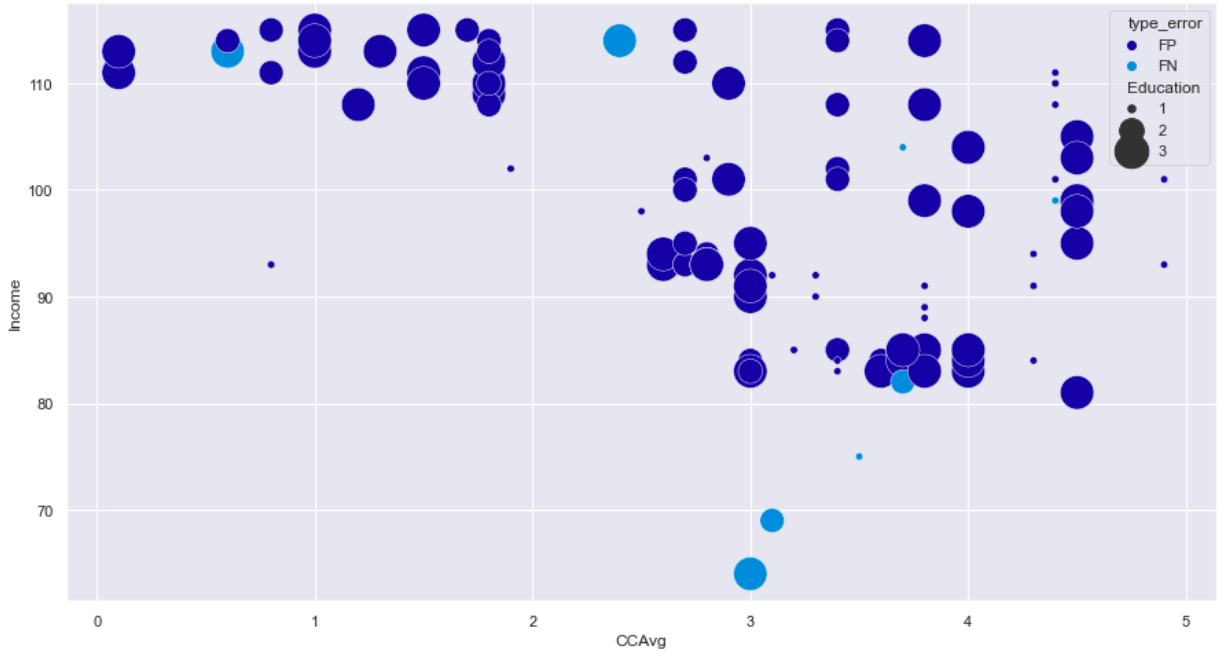


In [186…

```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(
    df_error["CCAvg"],
    df_error["Income"],
    hue=df_error["type_error"],
    size=df_error["Education"],
    sizes=(30, 600),
)
```

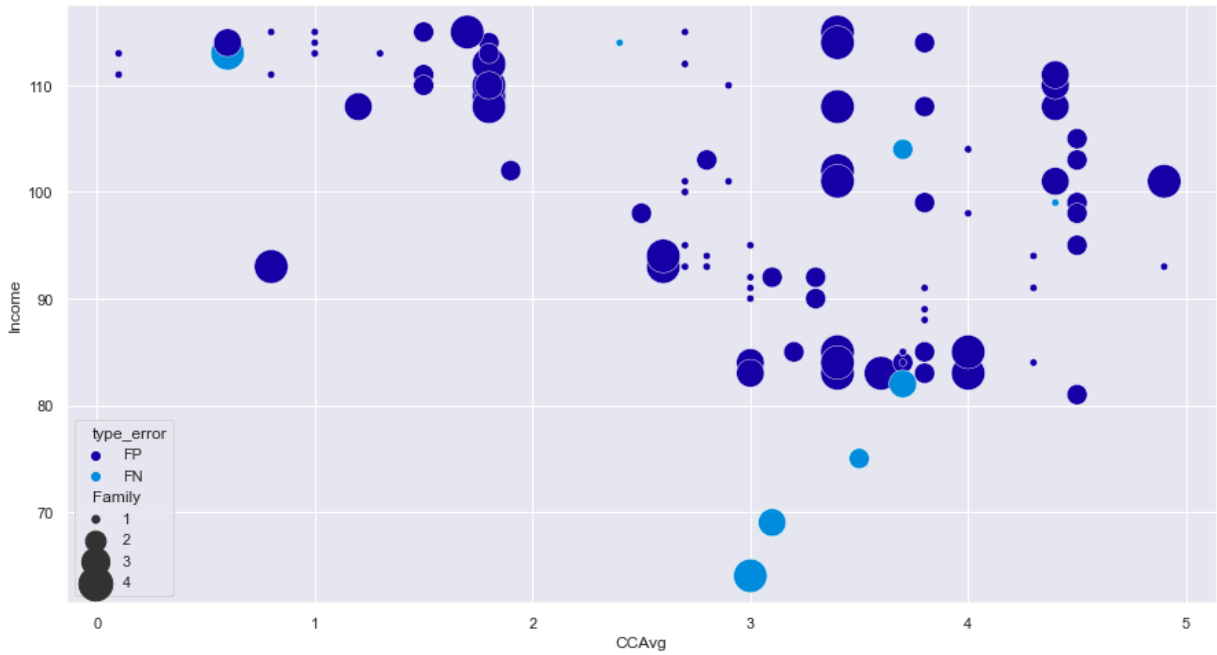
Out[186…

```
<AxesSubplot:xlabel='CCAvg', ylabel='Income'>
```



In [187…
```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(
    df_error["CCAvg"],
    df_error["Income"],
    hue=df_error["type_error"],
    size=df_error["Family"],
    sizes=(30, 600),
)
```

Out[187…
```
<AxesSubplot:xlabel='CCAvg', ylabel='Income'>
```



Observations:

- We can see that the error occurs due customer profille.
- Some customers even having a accepting Personal Loan profile (High Income, CCAvg, and family size) it does not accept.

- On the other hand, customers with no accepting Personal Loan profile, will say yes, Those ones we need to target then as well.
- FP customers, have a high income (>80k), with high Education (>=2) and High CCAvg (mean >=3k), but did not accept Loan.
- FN customers, have a High CCAvg (mean >=3.05k), but a min income of 64k and max 114k, with Education mean around 2 and and family size around 2, our model fail in detect it as a accepting Loan customer because they significant variables is oposit of a regular accepting Loan customer.

Conclusions

• We analyzed the "Personal Loan accepting" using different techniques and used Logiatic Regression and Decision Tree Classifier to build a predictive model for the same.

• The built model can be used to predict if a customer is going to accept or not a Personal_Loan and to create a Customer segment considering the significance of independente variables

• 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, followed by less data pre-processing (outliers, missing data, features engenneiring...). In the other hand Logistic Regression request all this data pre processing and it is difficult to correctly interpret the results and outliers affects the model. logistic regression looks at the simultaneous effects of all the predictors, so can perform much better with a small sample size.

• 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.

• The models predicted in logistic regression was 0.8973 on test and the decision tree was `0.9521` on test. The data sets did better with `Decision Tree` .

• Income, Family, CCAvg, Education, and Age are the most important variable in predicting the customers that will accept the Personal Loan

• We used post pruning to reduce overfitting a and choose the alpha 0.00135 to get a best Recall considering a good Precision as well. Meaning that, Recall is the most important metric, but we also wanna a good precision to make sure we are targeting the correct customers.

Recommendations | Insight

• According to the Decision Tree model, pos tunning bestmodel2:

a) If a customer `Income` is greater than 100k, there's a very high chance the customer will accept the Personal Loan.

b) If a customer has high Income and `Family` size greater or equal to 3, there is a high chance of this customer accept a Loan

- It is observed that the more the customers spend on CCAvg more is the likelihood of them to accept the Loan, also the bank can enhance the customer experience with tthe bank products as Securities Account, CD Account. Bank needs to do a strategi to convert more customers with CD Account, and CCAvg usually they will by a Personal Loan.
- After all campaign the model needs to be review (new information) and a new segmentation of customers should come out.
- Customer Segment is Income greater than 100k, Family size bigger than 2, High CCAvg and Higher Education.