Machine Learning: AllLife Bank Personal Loan Campaign

Problem Statement

Context

AllLife Bank is a US bank that has a growing customer base. The majority of these customers are liability customers (depositors) with varying sizes of deposits. The number of customers who are also borrowers (asset customers) is quite small, and the bank is interested in expanding this base rapidly to bring in more loan business and in the process, earn more through the interest on loans. In particular, the management wants to explore ways of converting its liability customers to personal loan customers (while retaining them as depositors).

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

You as a Data scientist at AllLife bank have to build a model that will help the marketing department to identify the potential customers who have a higher probability of purchasing the loan.

Objective

To predict whether a liability customer will buy personal loans, to understand which customer attributes are most significant in driving purchases, and identify which segment of customers to target 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? (0: No, 1: Yes)
- `Securities_Account` : Does the customer have securities account with the bank? (0: No, 1: Yes)
- `CD_Account` : Does the customer have a certificate of deposit (CD) account with the bank? (0: No, 1: Yes)
- `Online` : Do customers use internet banking facilities? (0: No, 1: Yes)
- `CreditCard` : Does the customer use a credit card issued by any other Bank (excluding All life Bank)? (0: No, 1: Yes)

Importing necessary libraries

In [ ]:
```
# Installing the libraries with the specified version.
!pip install numpy==1.25.2 pandas==1.5.3 matplotlib==3.7.1 seaborn==0.13.1 scikit-learn==1.2.2 sklearn-pandas==
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 38.1 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This
behaviour is the source of the following dependency conflicts.
cudf-cu12 24.4.1 requires pandas<2.2.2dev0,>=2.0, but you have pandas 1.5.3 which is incompatible.
google-colab 1.0.0 requires pandas==2.0.3, but you have pandas 1.5.3 which is incompatible.
```

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Note: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.*

In [2]:
```
import pandas as pd
import numpy as np

import matplotlib.pylab as plt
%matplotlib inline

import seaborn as sns
from sklearn.model_selection  import train_test_split
from sklearn.cluster import KMeans

from scipy.stats import zscore
```

```
from sklearn import metrics
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

Loading the dataset

In [3]:
```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Loan_Modelling.csv")
```

Data Overview

- Observations
- Sanity checks

In [39]:
```
# Viewing the first 5 rows of the dataset
df.head()
```

Out[39]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Account	CD_Account	Online	C
0	1	25	1	49	91107	4	1.6	Undergrad	0	0	1	0	0	
1	2	45	19	34	90089	3	1.5	Undergrad	0	0	1	0	0	
2	3	39	15	11	94720	1	1.0	Undergrad	0	0	0	0	0	
3	4	35	9	100	94112	1	2.7	Graduate	0	0	0	0	0	
4	5	35	8	45	91330	4	1.0	Graduate	0	0	0	0	0	

In [5]:
```
# Viewing the last 5 rows of the dataset
df.tail()
```

Out[5]:

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

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

Out[6]:
```
(5000, 14)
```

The data has 5000 rows and 14 columns

In [7]:
```
# Checking the data types of the columns in the dataset
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
```

- All of the columns execpt one are integers, but the column 'CCAvg' has a float data type.
- There are no missing values throughout the all the 13 columns in the data.

In [8]:
```
# Another check for missing values if any
df.isnull().sum()
```

Out[8]:
```
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
```

As seen above, there is not any sign of missing values in any column in the data.

In [9]:
```
# Checking for duplicated entries in the data.
df.duplicated().sum()
```

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

There is zero duplicated entries in the data.

In [10]:
```
# Check the statistical summary of the data.
df.describe().T
```

Out[10]:

	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
ZIPCode	5000.0	93169.257000	1759.455086	90005.0	91911.00	93437.0	94608.00	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.70	1.5	2.50	10.0
Education	5000.0	1.881000	0.839869	1.0	1.00	2.0	3.00	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101.00	635.0
Personal_Loan	5000.0	0.096000	0.294621	0.0	0.00	0.0	0.00	1.0
Securities_Account	5000.0	0.104400	0.305809	0.0	0.00	0.0	0.00	1.0
CD_Account	5000.0	0.060400	0.238250	0.0	0.00	0.0	0.00	1.0
Online	5000.0	0.596800	0.490589	0.0	0.00	1.0	1.00	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1.00	1.0

- `Age:` The average customer's age is 45. 75% of the ages are less than 55 years old. The maximum age of the customers has reached 67 years old.
- `Experience` : On average the years of professional experience is 20 years, the same as the median. The minimum value for the experience column is -3 which doesn't make any sense. This will have to be changed.
- `Income` : On average the annual income of the customers is 73 thousand dollars. However, the maximum value compared to the 75% percentile has a great margin which indicates that there is a possibility of there being an outlier.
- The distribution of the columns ZIPCode, Family, CCAvg, and Education is fine. The maximum values of these columns might require a check.
- `Mortgage` : The average value of a house mortgage if any in the data is 56 thousand dollars. However, similar to the Income column, the maximum value is much greater than the 75% percentile, which could result in the column containing outliers.
- The rest of the columns Personal_Loan, Securities_Account, CD_Account, Online, and CreditCard all contain 1's and 0's due to them being columns that ask questions thus resulting in yes for a value of 1 and no for a value of 0. In those columns, there doesn't seem to be any issue either.

In [11]:
```
# Checking the total number of unique values in the zipcode column
df['ZIPCode'].nunique()
```

Out[11]:
```
467
```

- There is a total of 467 different zipcodes in the data. This indicates that the area in which the data is collected is quite large.

Exploratory Data Analysis.

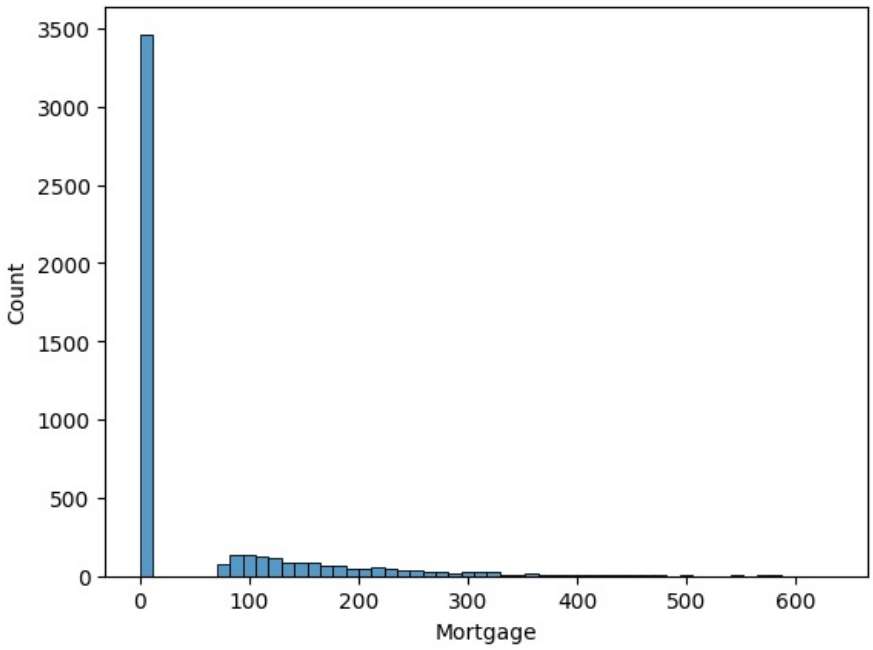
- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis correctly and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

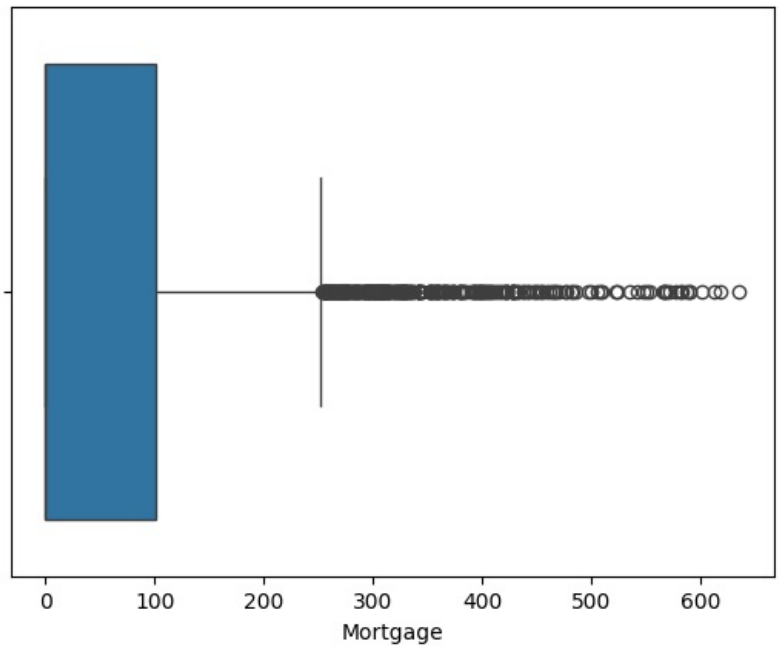
Questions:

1. What is the distribution of mortgage attribute? Are there any noticeable patterns or outliers in the distribution?
2. How many customers have credit cards?
3. What are the attributes that have a strong correlation with the target attribute (personal loan)?
4. How does a customer's interest in purchasing a loan vary with their age?
5. How does a customer's interest in purchasing a loan vary with their education?

Question 1

In [34]:
```
sns.histplot(data=df,x='Mortgage')
plt.show()
sns.boxplot(data=df,x='Mortgage')
plt.show()
```





- The distribution is skewed towards the right as seen in the boxplot above.
- There are many outliers present in the Mortgage column, specifically from the value 250 and above following the 75% percentile whisker.

Question 2

In [35]:
```
# Check how many customers have a credit card in the data
df['CreditCard'].value_counts()
```

Out[35]:
```
CreditCard
0    3530
1    1470
Name: count, dtype: int64
```

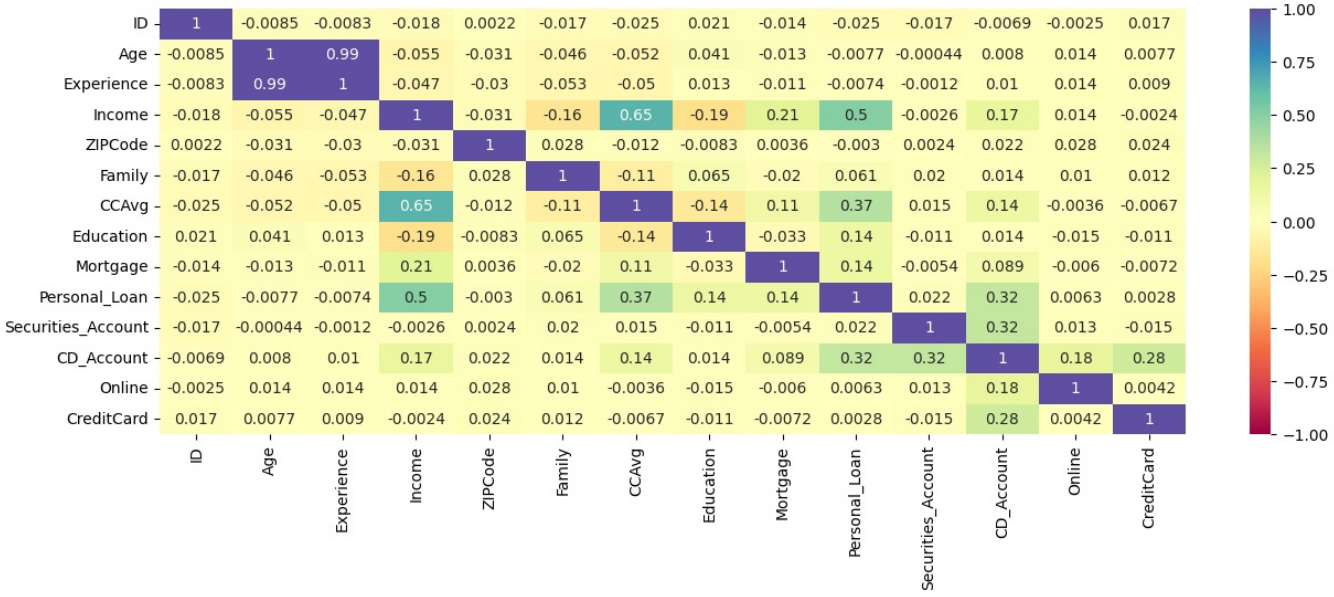
- 1470 customers use a credit card, this is much less than 50% of the total amount of customers. More than half the amount of customers do not use a credit card in the data.

Question 3

In [60]:
```
# Finding the attributes that have a strong correlation with the personal loan
df.corr()['Personal_Loan']
```

Out[60]:
```
ID                  -0.024801
Age                 -0.007726
Experience          -0.007413
Income               0.502462
ZIPCode             -0.002974
Family               0.061367
CCAvg                0.366889
Education            0.136722
Mortgage             0.142095
Personal_Loan        1.000000
Securities_Account   0.021954
CD_Account           0.316355
Online               0.006278
CreditCard           0.002802
Name: Personal_Loan, dtype: float64
```

In [61]:
```
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True,cmap='Spectral',vmin=-1,vmax=1)
plt.show()
```

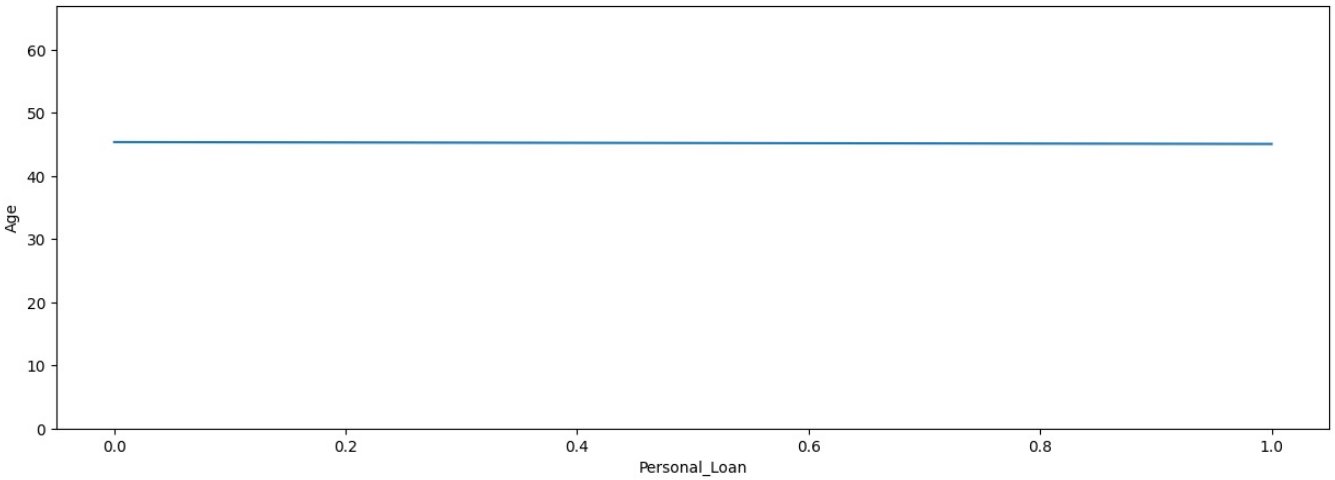


- As seen in the above chart and visualization, the Income attribute contains the strongest correlation with the target attribute, personal loan.

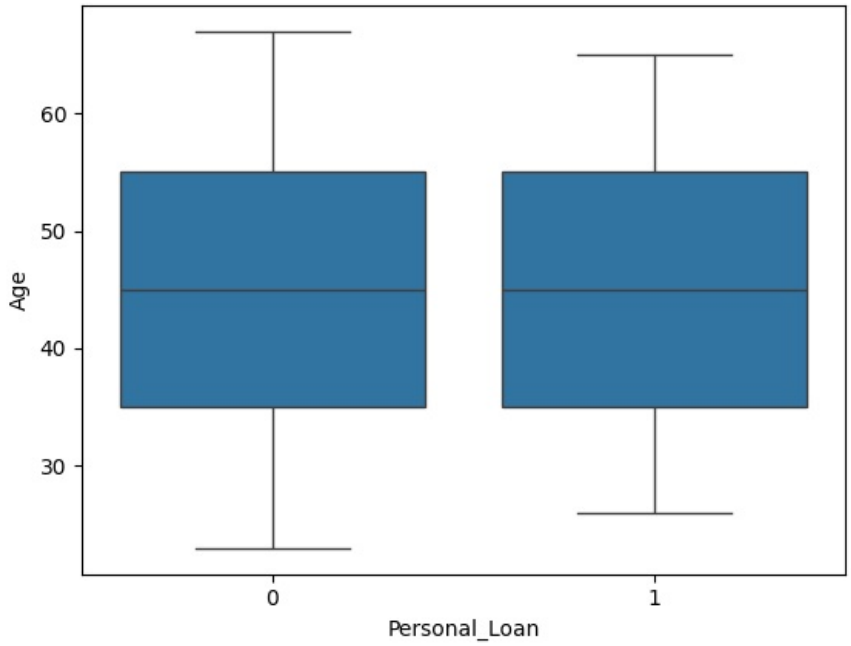
Question 4

In [40]:
```
# How does a customer's interest in purchasing a loan vary with their age?
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15, 5))
sns.lineplot(data=df, y='Age', x='Personal_Loan', ci=False);
plt.ylim(0, 67)
plt.show()
```

```
<ipython-input-40-1f47c83bd636>:6: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effect.

  sns.lineplot(data=df, y='Age', x='Personal_Loan', ci=False);
```

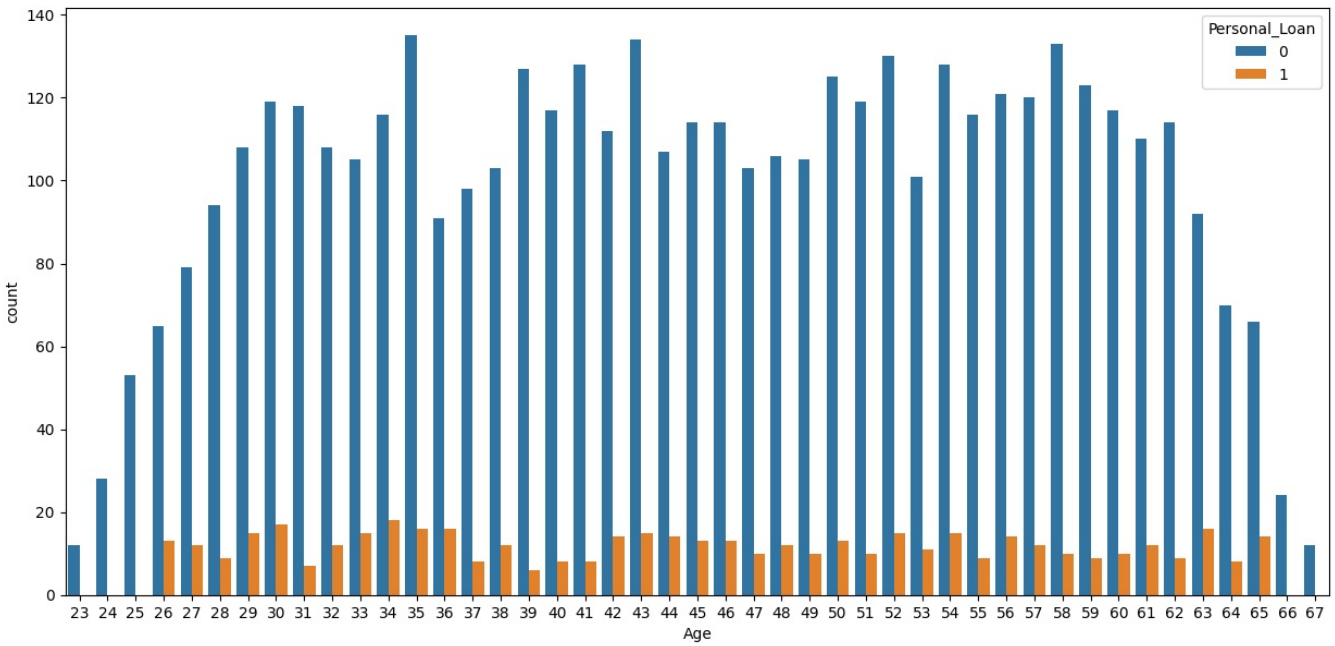


In [41]:
```
sns.boxplot(data=df, y='Age', x='Personal_Loan') ;
```



In [42]:
```
plt.figure(figsize=(15,7))
sns.countplot(data=df, x='Age',hue='Personal_Loan')
```

Out[42]:
```
<Axes: xlabel='Age', ylabel='count'>
```

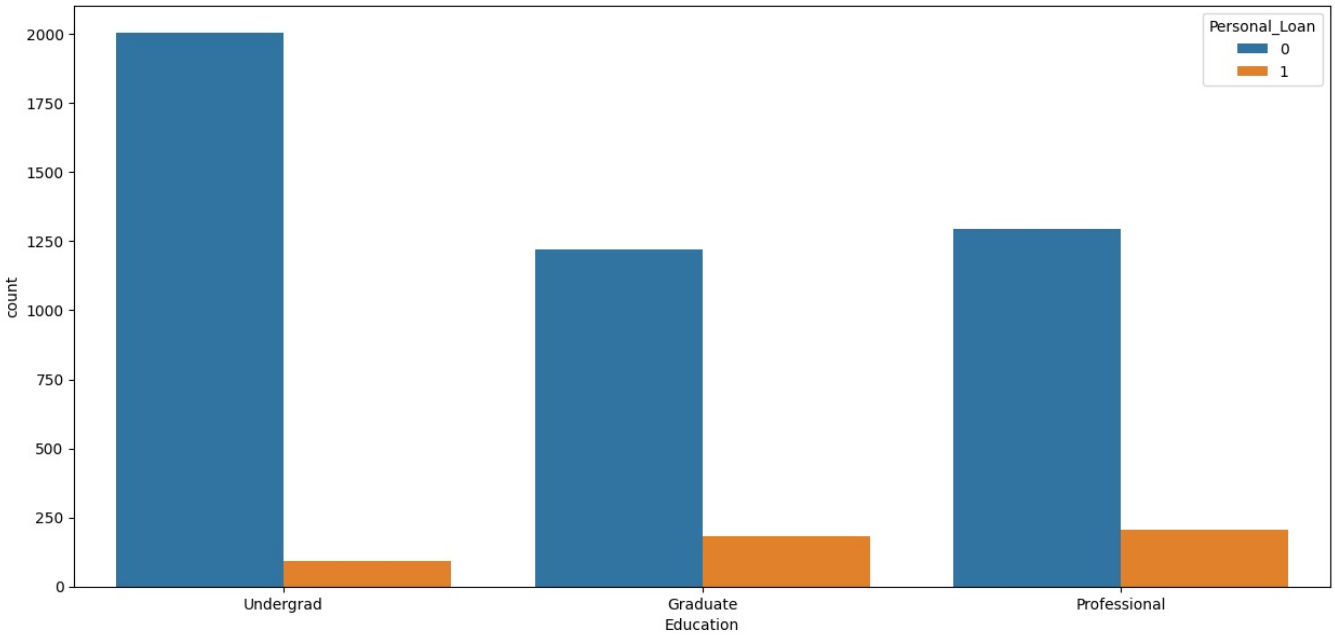


- As seen in the above lineplot, boxplot, and countplot, there really is no association between the customers age in their decision for purchasing a loan, the data more or so is mostly a normal distribution.

Question 5

In [43]:
```
# How does a customer's interest in purchasing a loan vary with their education?
# Note: 1 = Undergrad, 2 = Graduate and 3 = Professional
plt.figure(figsize=(15,7))
sns.countplot(data=df, x='Education',hue='Personal_Loan')
```

Out[43]:
```
<Axes: xlabel='Education', ylabel='count'>
```

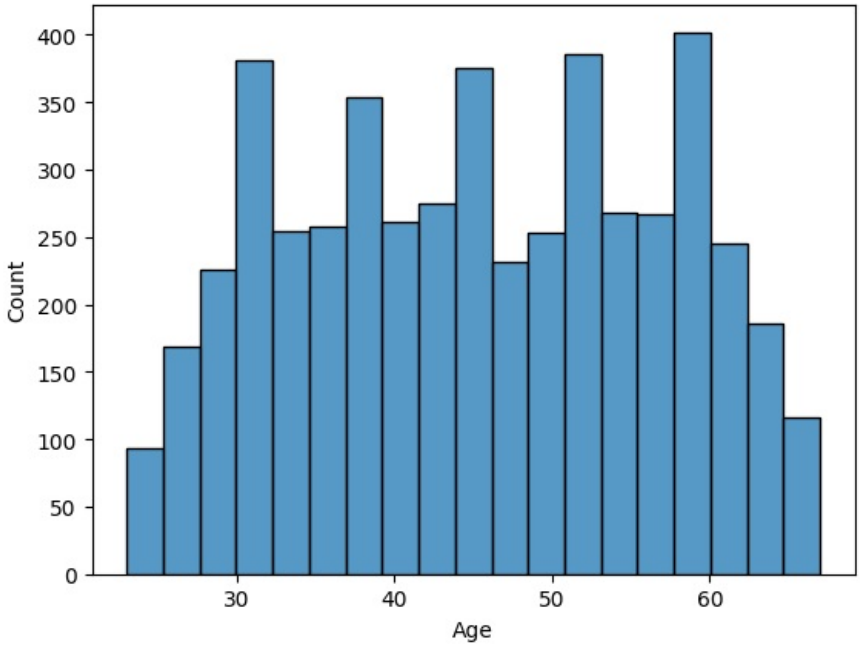


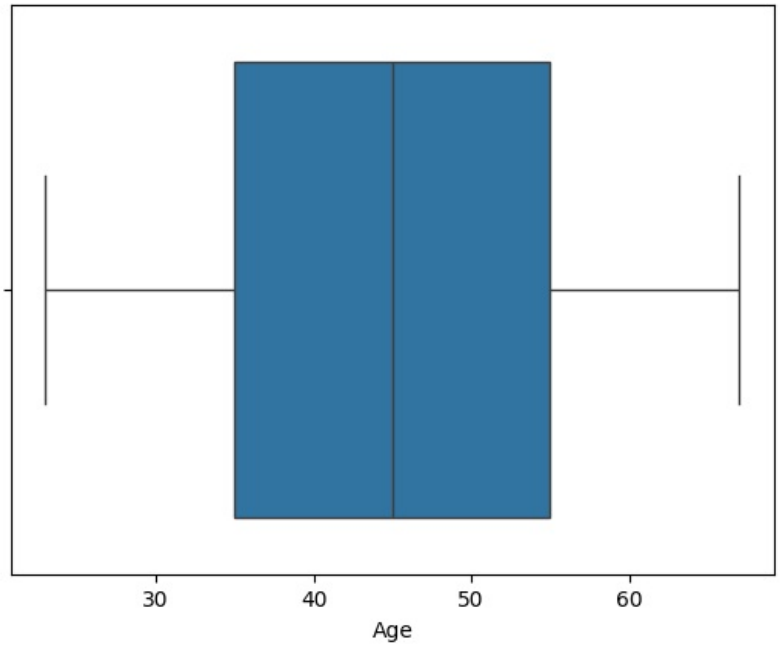
- As seen in the above counterplot, both the graduates(2) and the professionals(3) have a much lower margin in denying the personal loan offer compared to the undergraduates(1).
- However, on the flip side, the graduates and the professionals had almost the same amount of accepted personal loans and the undergrads had half as much.
- This shows that as more advanced the educational level is, the more likely the personal offer will be accepted, and vice versa.

Univariate Analysis

Checking the distribution for numerical columns

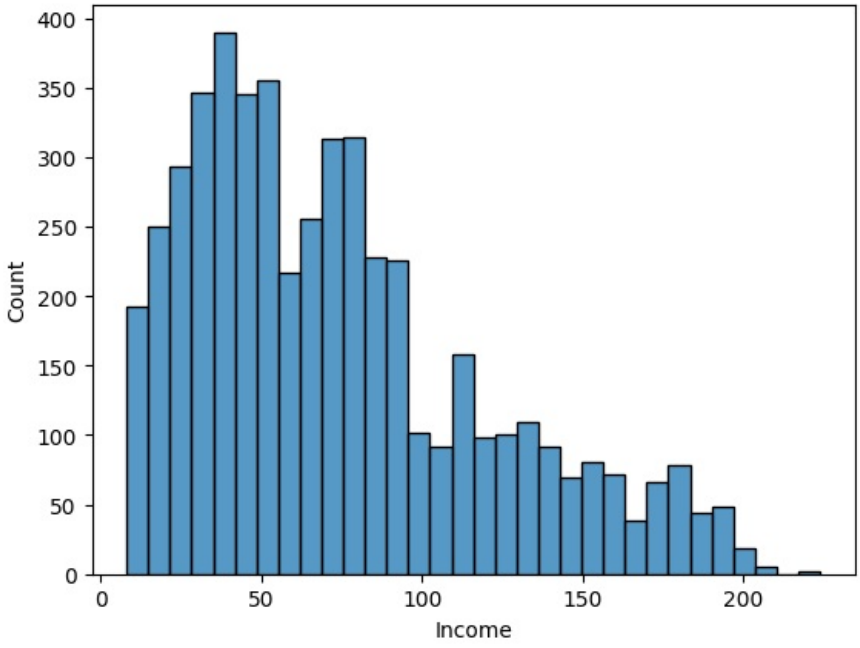
In [62]:
```
# Observations for Age

sns.histplot(data=df,x='Age')
plt.show()
sns.boxplot(data=df,x='Age')
plt.show()
```

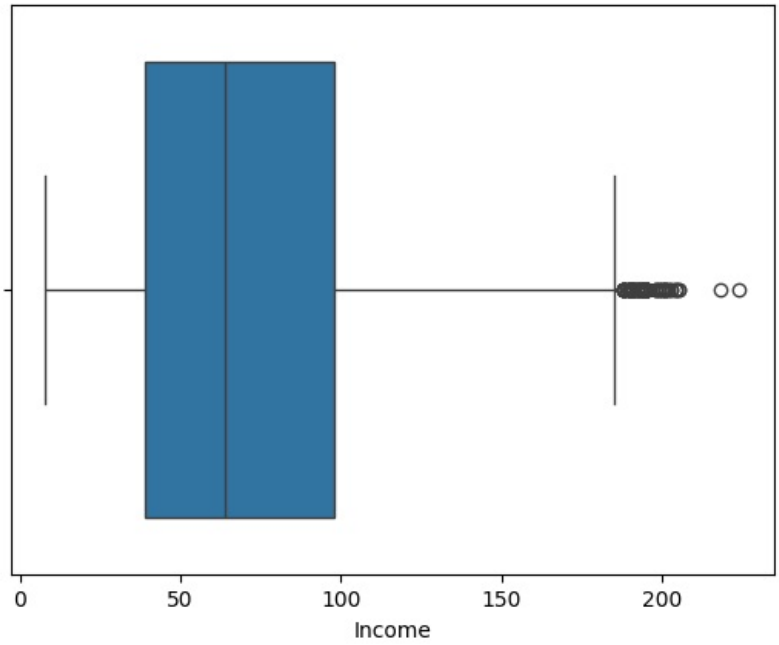




- In the above graphs, the distribution for the ages of the customers is normally distributed. This is a great sign which indicates that there isn't an age bias or a sort of leaning towards a certain demographic present in this data.
- There are also no outliers present in this column.

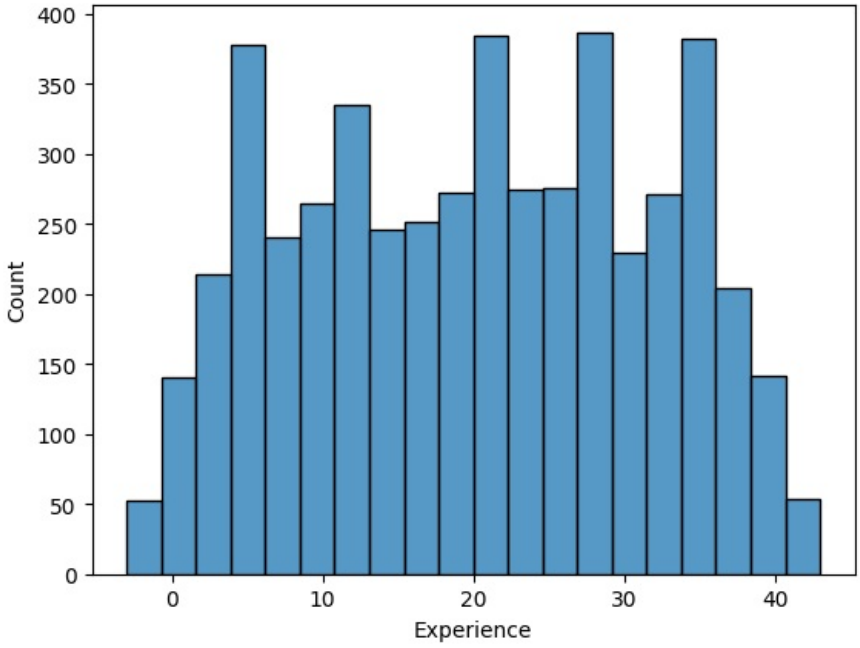
In [63]:
```
# Observations on Income

sns.histplot(data=df,x='Income')
plt.show()
sns.boxplot(data=df,x='Income')
plt.show()
```

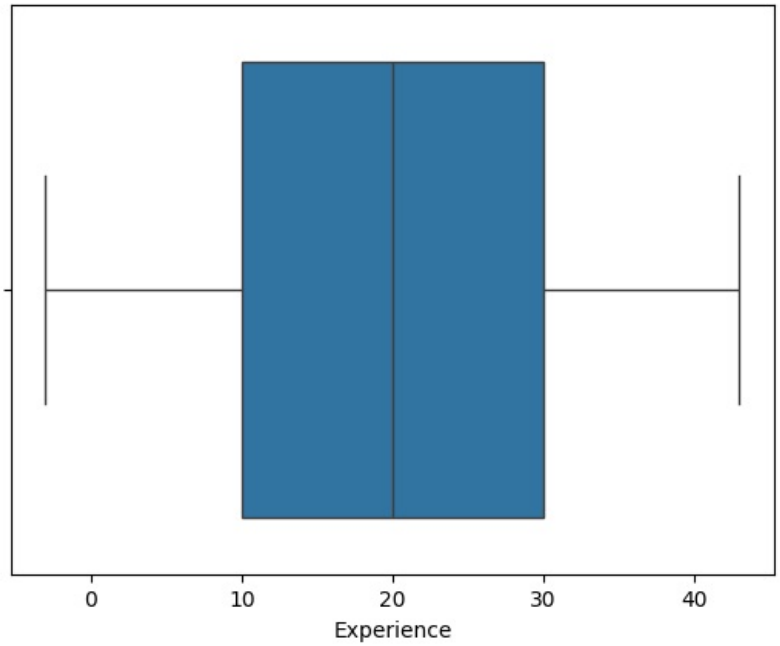




- As seen in the histogram and boxplot above, the annual income of the customers is skewed to the right.
- There are also many outliers present in the column.
- The income being right-skewed also means that the majority of the customers make less than 99,000 dollars, this can show the type of economic/financial situation most customers are in.

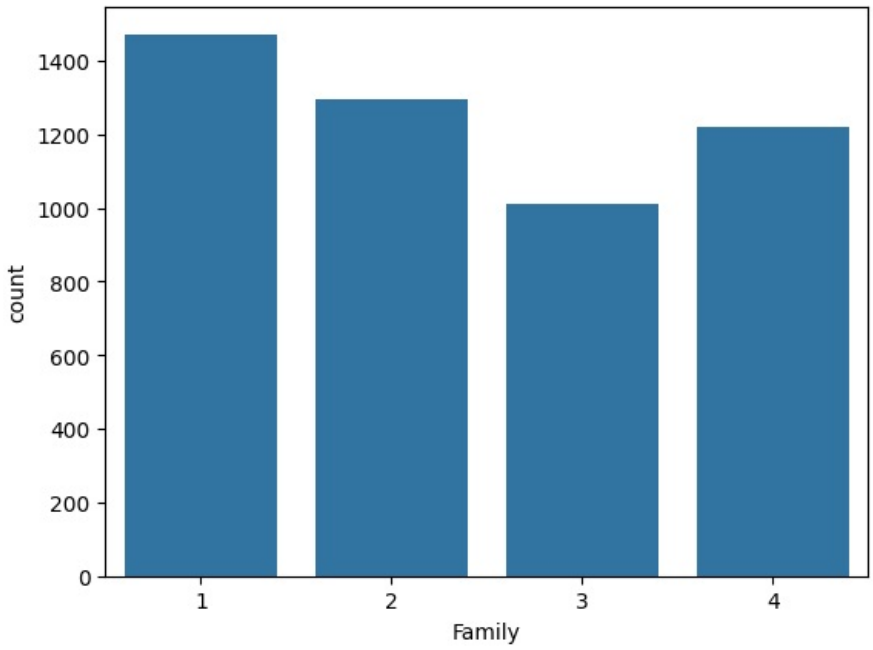
In [64]:
```
# Observations on Experience

sns.histplot(data=df,x='Experience')
plt.show()
sns.boxplot(data=df,x='Experience')
plt.show()
```





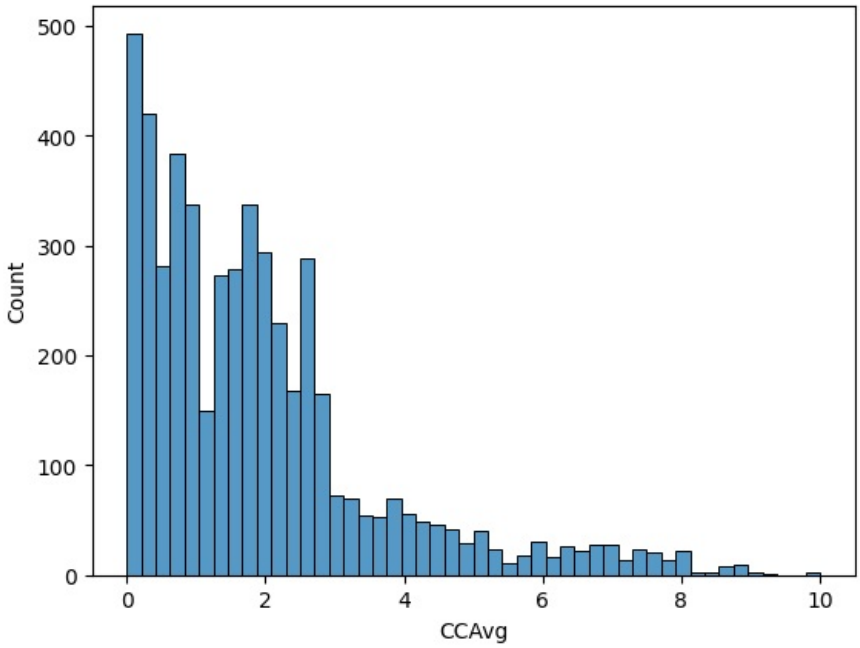
- In the above plots, the data is normally distributed which indicates that the years or professional experience between the customers in the data ranges from 5 to 35 in an even manner.
- The column also doesn't contain any outliers, thus being normally distributed.

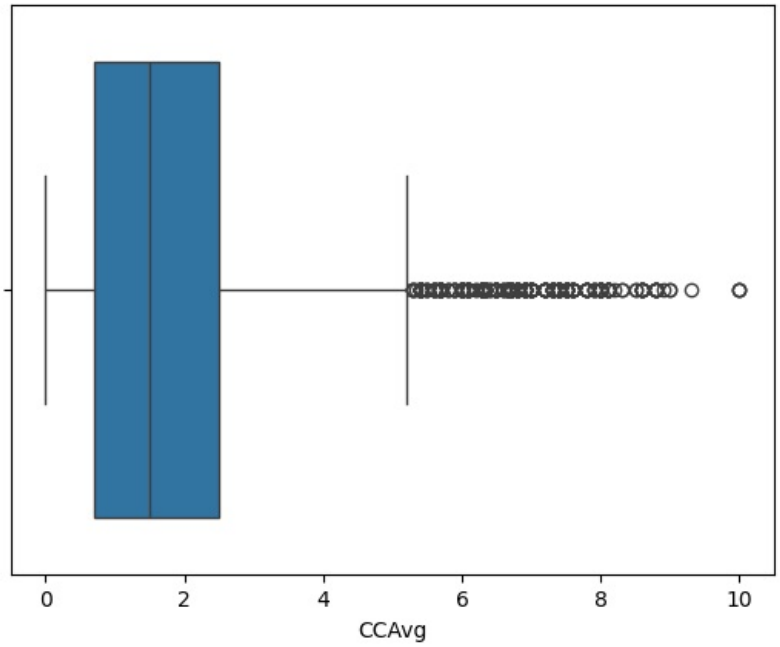
In [65]:
```
# Observations on Family

sns.countplot(data=df,x='Family')
plt.show()
```



- In the counterplot above, the family sizes of the customers are not significantly different from each other.
- The greatest family size being 1 whilst the least family size being 3 indicates that the customers are most prone to smaller families in that demographic.

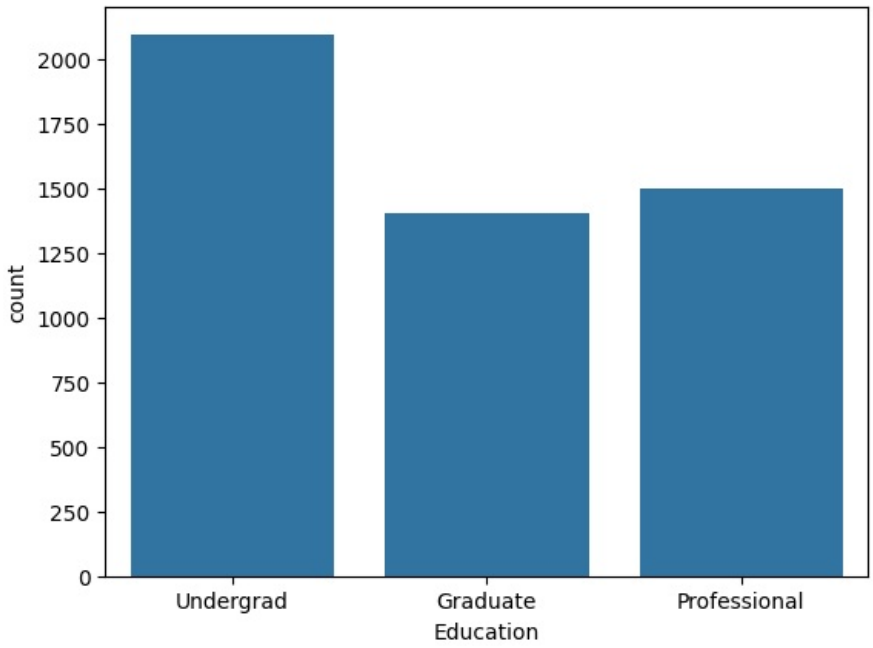
In [66]:
```
# Observation on CCAVg

sns.histplot(data=df,x='CCAvg')
plt.show()
sns.boxplot(data=df,x='CCAvg')
plt.show()
```





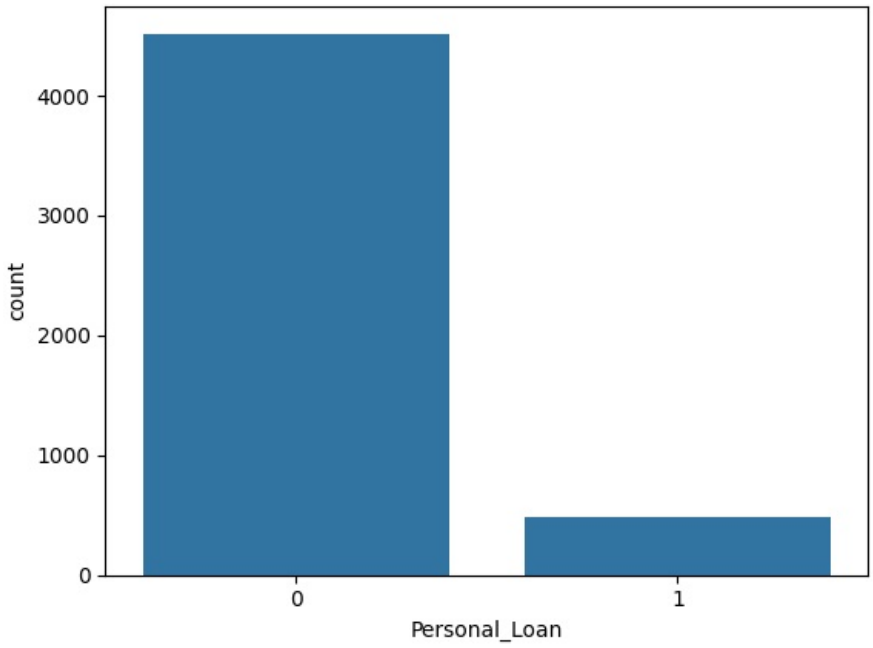
- The average spending on credit cards per month for customers is highly skewed to the right. This indicates that there is a significant amount of customers who do not spend too much on credit cards per month.
- This can either be good or bad, in a good manner if the customers are saving their money wisely and in a bad way if they are not wealthy enough to spend money on certain things.
- There are also many outliers present after 5 thousand dollars spent.

In [67]:
```
# Observation on Education

# To make the depiction easier, convert 1 = Undergrad, 2 = Graduate and 3 = Professional
df['Education'] = df['Education'].replace({1:'Undergrad',2:'Graduate',3:'Professional'})

sns.countplot(data=df,x='Education')
plt.show()
```



- As seen in the counterplot above, there is a good margin for the amount of undergraduates collected in the data compared to graduates and professionals. This can suggest that the demographic in the data is mostly young individuals or those with not as much experience educational-wise.

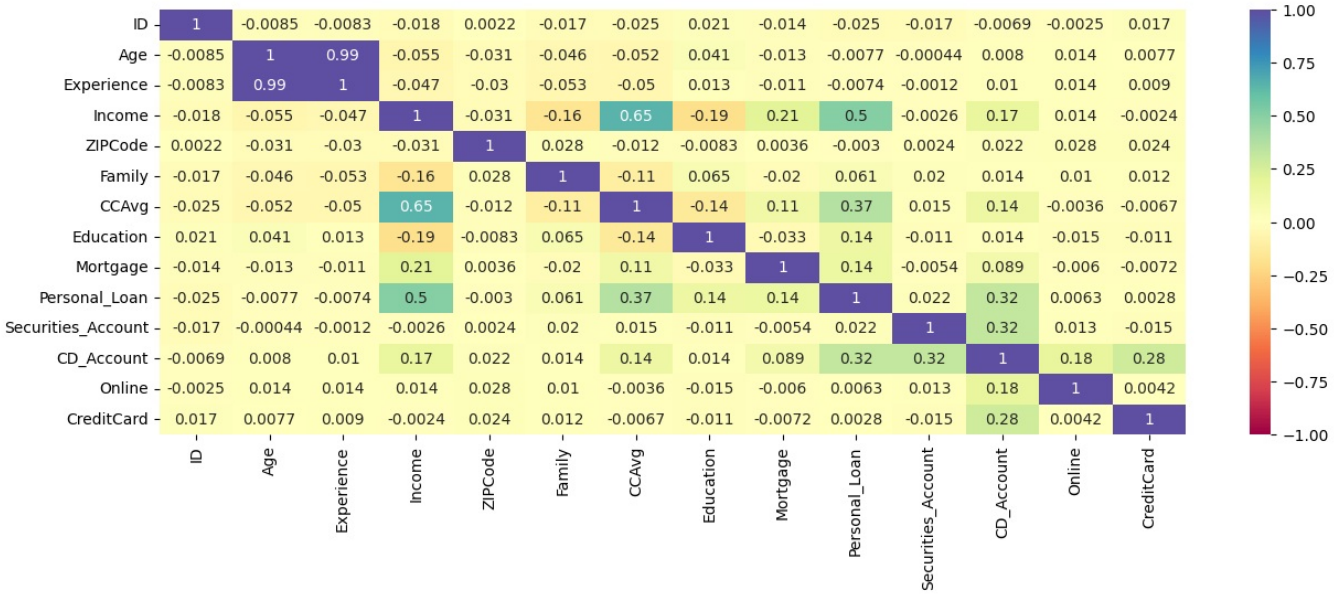
In [68]:
```
# Observation on Personal Loan

sns.countplot(data=df,x='Personal_Loan')
plt.show()
```



- In the above count plot, there is a major difference in the amount of customers who had declined the personal loan offer compared to those who had accepted.
- This can suggest the type of customers who have been offered.

Bivariate Analysis

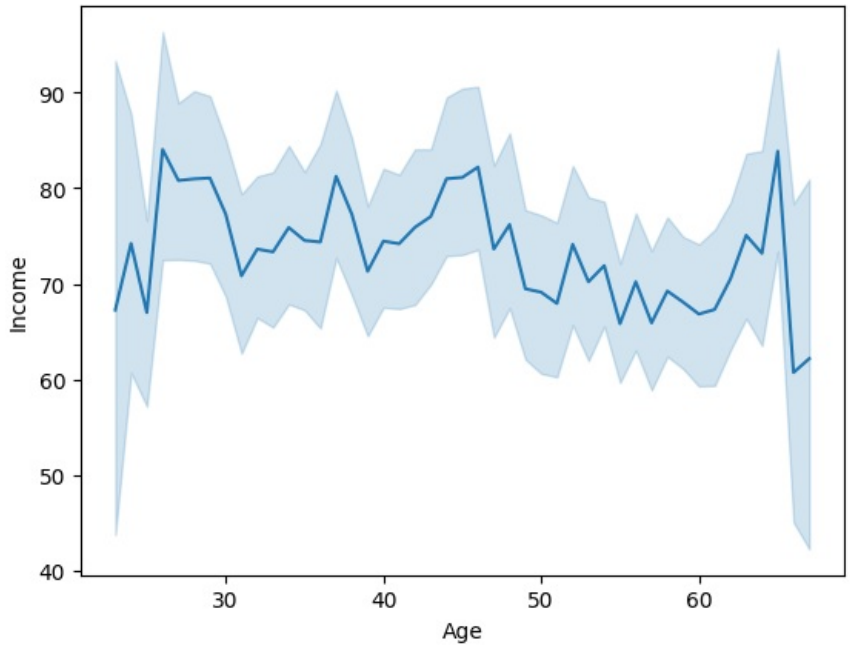
In [69]:
```
# Creating a heat map to visualize the correlations of all the variables in the data

# First convert the Education variables back to its original state
df['Education'] = df['Education'].replace({'Undergrad':1,'Graduate':2,'Professional':3})

plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True,cmap='Spectral',vmin=-1,vmax=1)
plt.show()
```



Observations

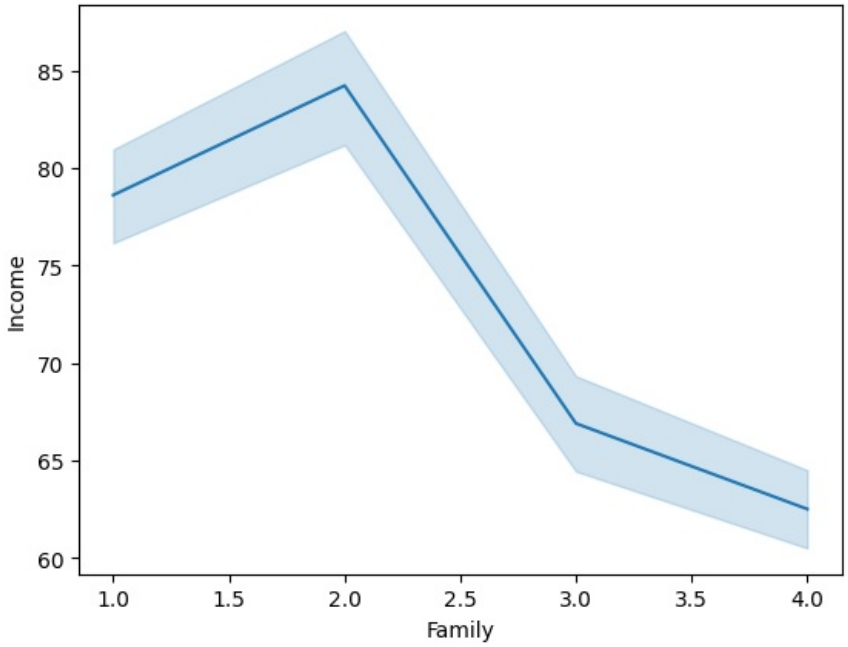
- The Age column displays a very high correlation with the Experience column. This indicates that the higher the age that customer is the more experience they gain, which is an obvious point to put out, but a good one to look at.
- The same could be said for the correlation between Education and Income columns, which have a negative correlation. Meaning the lower the educational level the higher the customer's income is. Additionally, the less the family size the lower the income.
- The correlations between the Zipcode column and the rest of the columns shouldn't be looked at too descriptively due to the column being a unique identifier for certain areas, increasing and decreasing in a zipcode wouldn't cause any impact.

In [70]:
```
# Exploring the correlation between Age and Income

sns.lineplot(data=df,x='Age',y='Income')
plt.show()
```



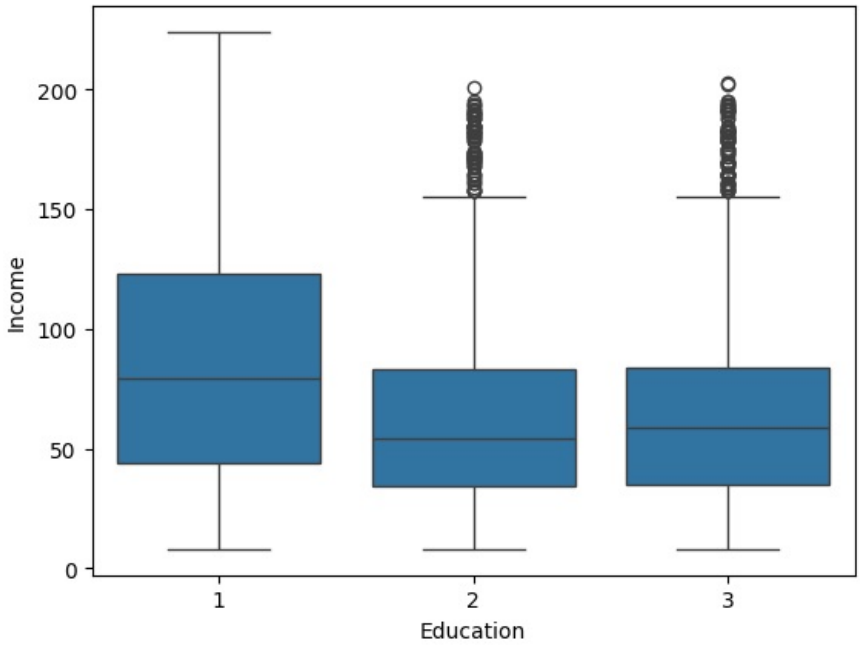
- As seen in the above line plot, there is not correlation between the two variables as there is no gradual increase or decrease in the lines.
- Although from ages 45 to 65 there does seem to be somewhat of a decrease in the customer's annual income. This can suggest that customers at that age do not have high-income jobs.

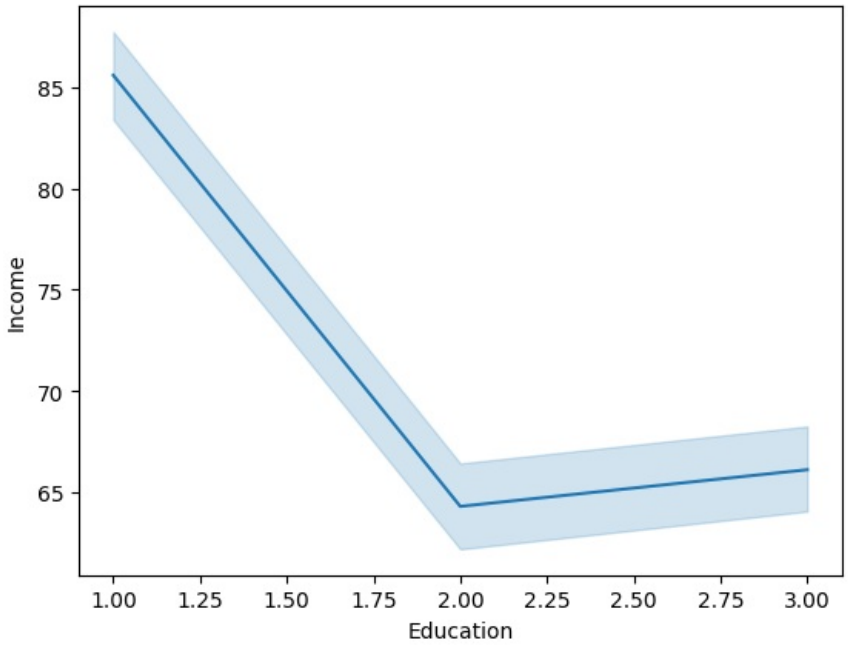
In [71]:
```
# Finding the correlation between Family and Income

sns.lineplot(data=df,x='Family',y='Income')
plt.show()
```



- From the plot above, there is a clear depiction of the income gradually decreasing as the family size increases.
- This can indicate that the bigger the family size is the less their annual income.

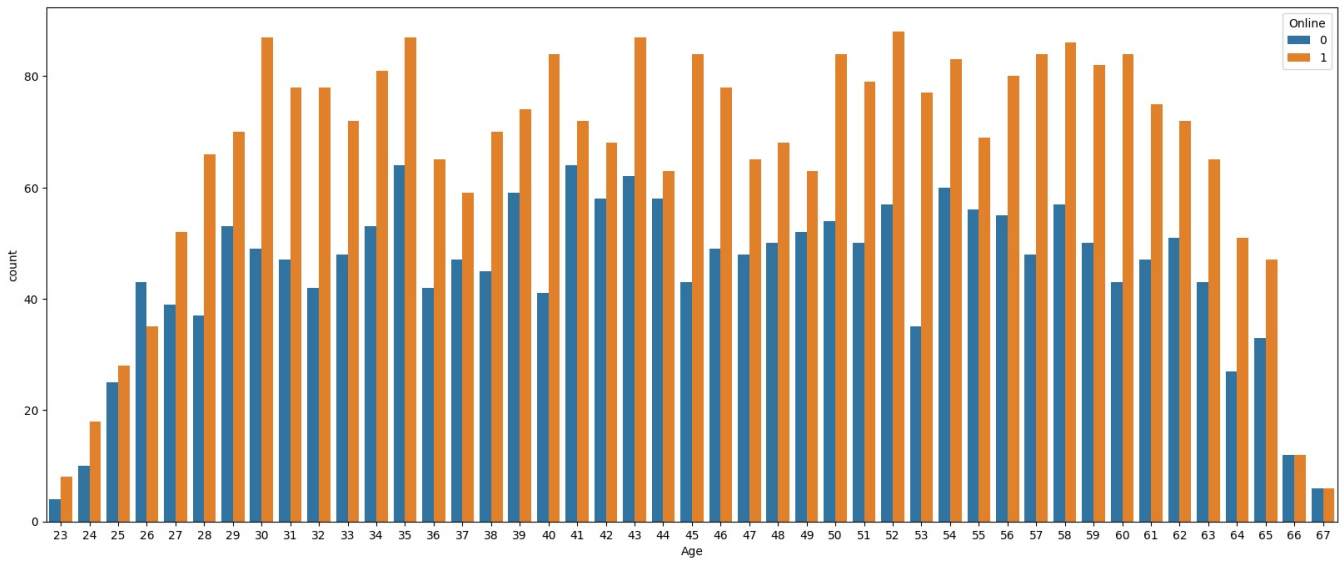
In [72]:
```
# Finding the correlation between Income and Education

# Note: 1 = Undergrad, 2 = Graduate and 3 = Professional

sns.boxplot(data=df,y='Income',x='Education')
plt.show()
sns.lineplot(data=df,y='Income',x='Education')
plt.show()
```





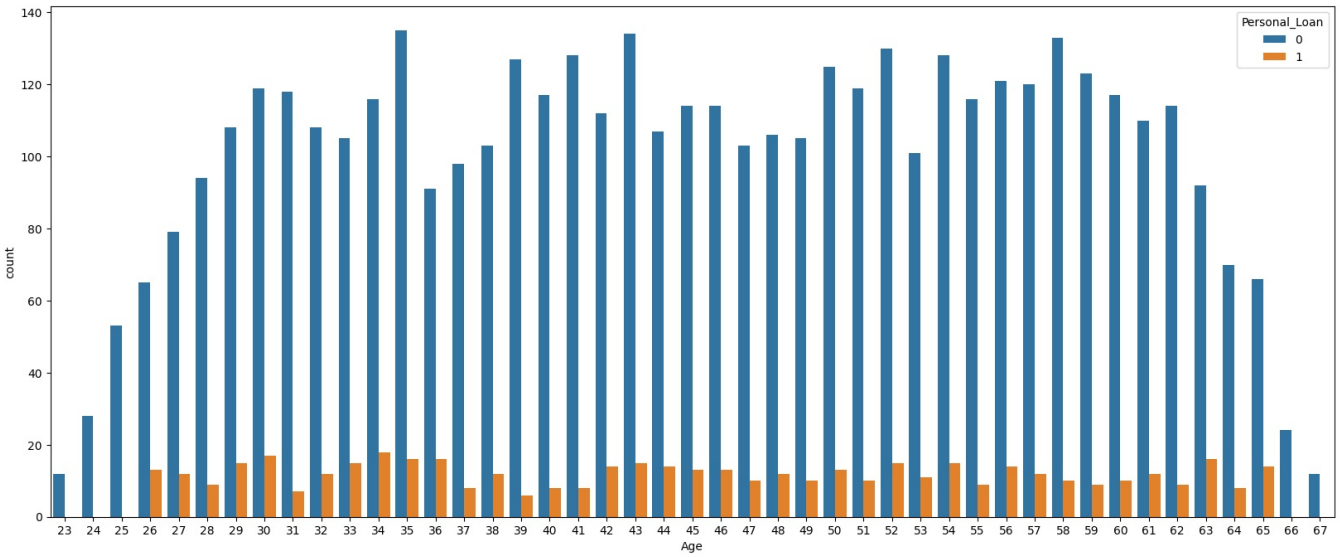
- From the plots above, it is shown that undergraduates have the most amount of income compared to graduates and professionals which is somewhat alarming.
- There are also outliers present in both graduates and professionals which could be given some attention later on.
- The undergradutes containing the most income could suggest an error in the data collection or there could be nothing incorrect about the data collected. Maybe undergrads having a higher annual income than professionals is normal in that particular area in which the data is taken.

In [73]:
```
# Exploring the correlation between Online and Age

plt.figure(figsize=(20,8))
sns.countplot(data=df,x='Age',hue='Online')
plt.show()
```



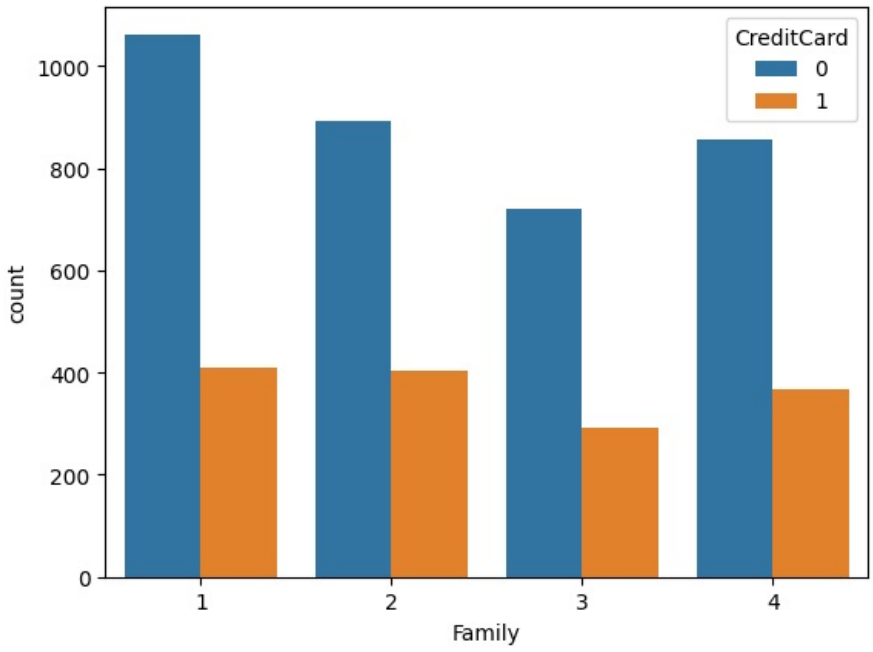
- As seen in the above count plot, the distribution between the customers who use Internet banking facilities and their age is surprisingly normal, I would have believed that the older the customers the less prone they are to use an Internet banking facility, but that is not the case.

In [74]:
```
# Exploring the relationship between Age & Personal Loan

plt.figure(figsize=(20,8))
sns.countplot(data=df,x='Age',hue='Personal_Loan')
plt.show()
```



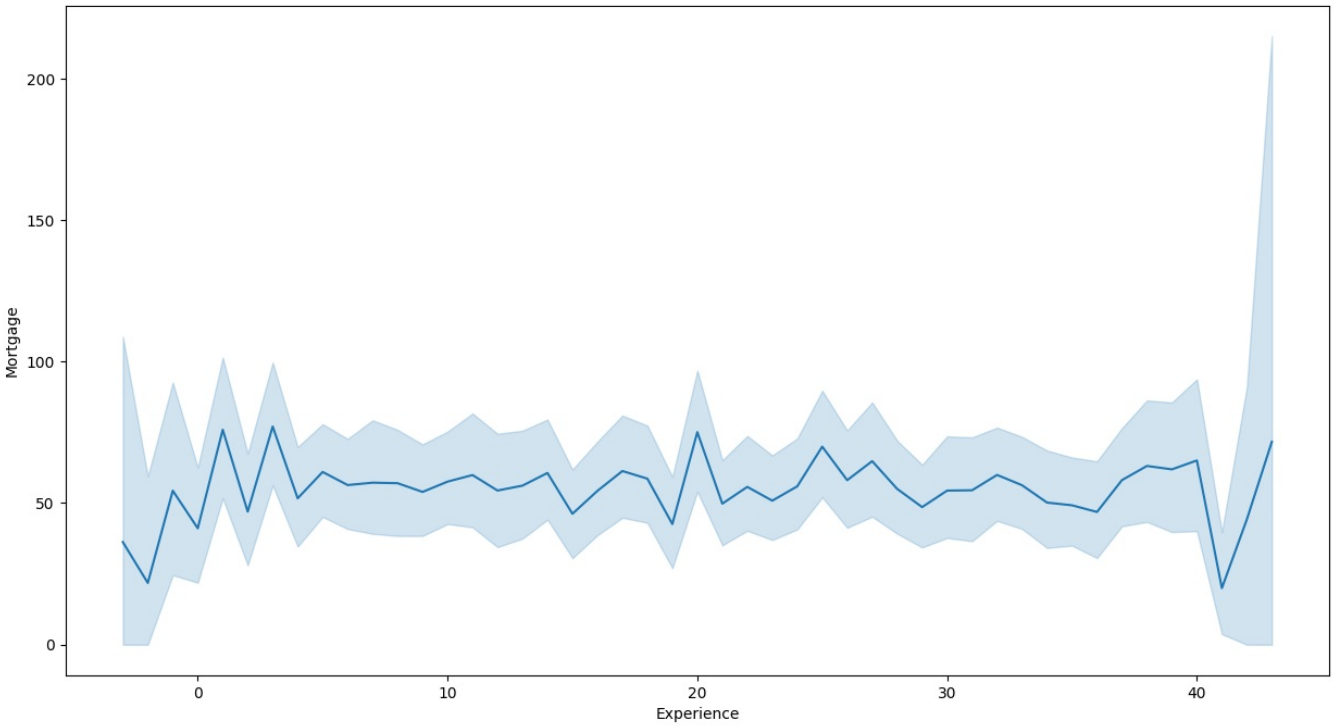
- As seen in the count plot above the data is normally distributed, there is no presence of any skewness or outliers.
- Although, the amount of customers who had accepted a personal loan offer to those who didn't across all ages in the data is quite drastic.

In [75]:
```
# Analyzing the relationship between Family Size and Credit Card Usage

sns.countplot(data=df,x='Family',hue='CreditCard')
plt.show()
```



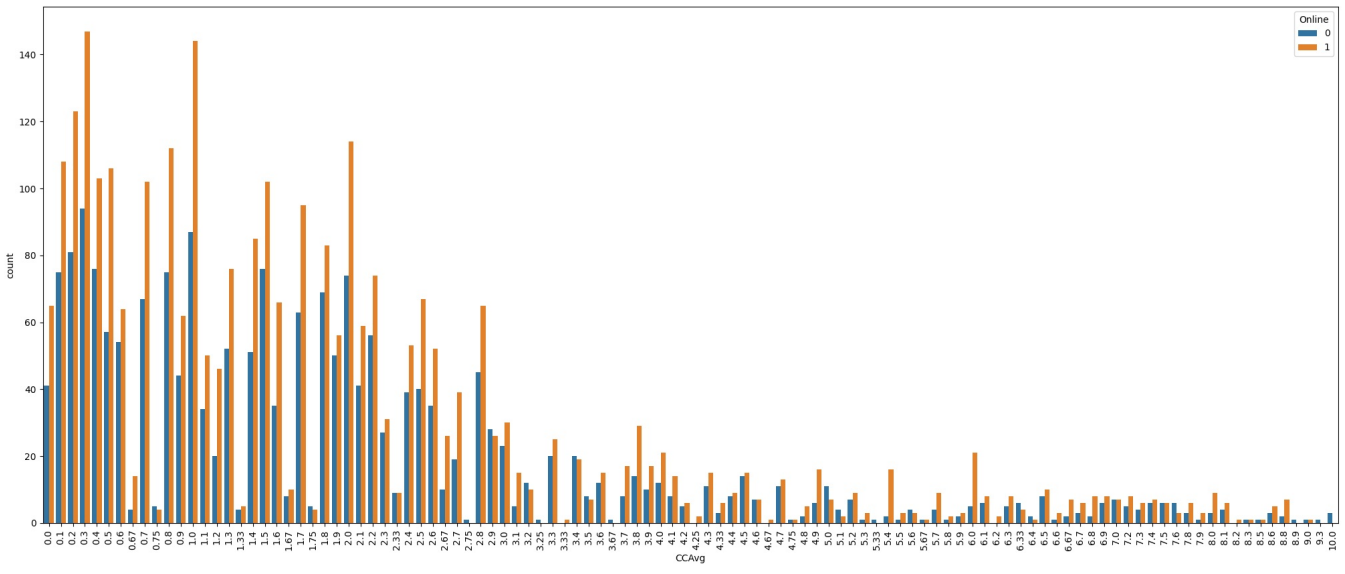
- In the above plot, from family sizes 1 through 4, there is almost a double in the amount of those who do not use a credit card issued by any other bank compared to the families that did.
- This shows that the families that were partaken in the data are loyal to AllLife Bank.

In [76]:
```
# Investigating the realtionship between Experience and Mortgage Value

plt.figure(figsize=(15,8))
sns.lineplot(data=df,x='Experience',y='Mortgage')
plt.show()
```



- From this line plot above, we can observe that there is not any significant change happening as the years of professional experience increase.
- Therefore concluding that the correlation between these two columns is not as notable as expected. This could also be due to not enough houses having mortgages.

In [77]:
```
# Depicting the relationship between CCAvg & Online

plt.figure(figsize=(25,10))
sns.countplot(data=df,x='CCAvg',hue='Online')
plt.xticks(rotation=90) # Rotate x-axis labels by 90 degrees
plt.show()
```



- In the count plot, it seems to be right-skewed meaning that there is a more significant amount of customers who spend less whilst utilizing an internet banking facility.
- Therefore the more the average customer spends on their credit card per month, the less the amount of customers who use the internet banking facility.

Data Preprocessing

- Missing value treatment
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)
- Preparing data for modeling
- Any other preprocessing steps (if needed)

Missing Value Treatment

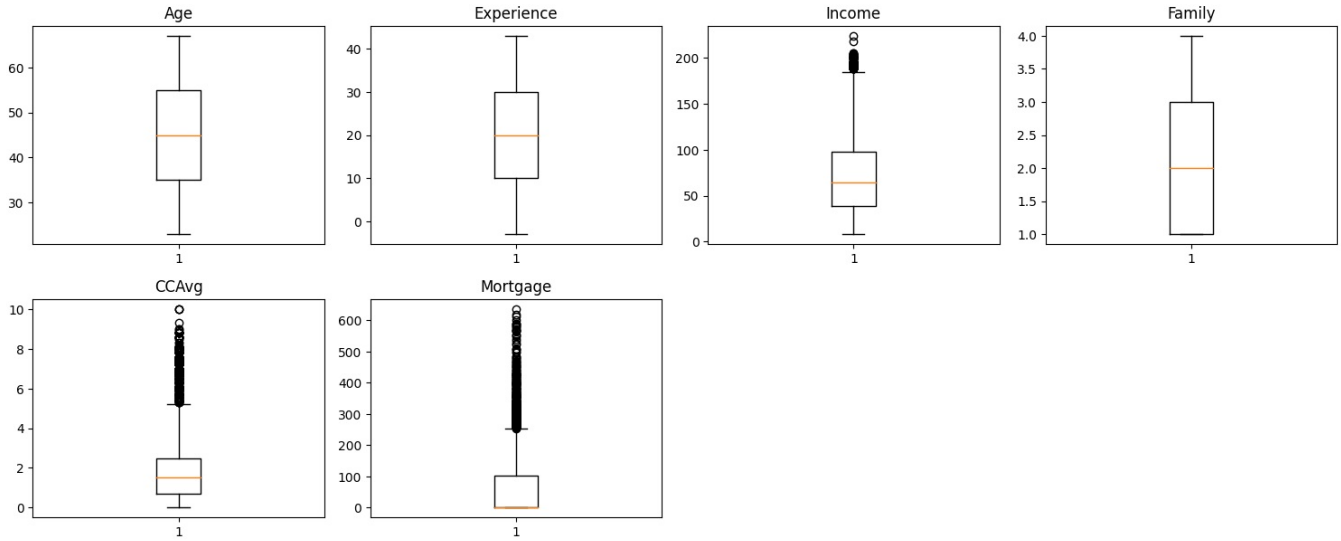
In [78]:
```
# Count the percentage of missing values (if any) in each column

pd.DataFrame({'Count':df.isnull().sum()[df.isnull().sum()>0],'Percentage':(df.isnull().sum()[df.isnull().sum()>
```

Out[78]:

	Count	Percentage

- As seen in the above table, there are no columns with any missing values therefore the table is empty with no rows or columns.

In [79]:
```
# Visualizing the outliers in all the numerical columns present in the data

numeric_columns = ['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Mortgage']
plt.figure(figsize=(15, 12))

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

plt.show()
```



- We can observe that 3 out of the 6 numerical columns contain outliers in them.

Analyze the numerical columns to see if the values are considered outliers or not.

- Income - The outliers in this column can be considered normal data because a customer's annual income can be more than about 180 thousand dollars, that is something that isn't unrealistic.
- CCAvg - The customer's average spending on credit cards per month depends upon various factors. For instance, the type of job that individual is working, obviously the majority of customers have a credit card spending average of less than 3 thousand dollars per month, but again it could be considered as genuine data.
- Mortgage - The value of the customer's house mortgage also varies in many elements as well. The customer's house location, for instance, can contribute to the mortgage price, the prices could also change from time to time. Additionally, the outliers are outliers due to there being a significant amount of customers who do not have mortgages for their houses to those who do.

Therefore, the outliers present in the data are not considered outliers and do not need to be treated.

Data Preperation for Modeling

In [80]:
```
# Looking at the first 5 rows of the data.

df.head()
```

Out[80]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Account	CD_Account	Online	C
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	

- From looking at the data, we can create a dummy variable for the Education column to visualize the classifications for each education level in separate columns.

In [81]:
```
# First convert education level 1 to Undergrad, level 2 to Graduate, level 3 to Professional
df['Education'] = df['Education'].replace({1:'Undergrad',2:'Graduate',3:'Professional'})

# Create dummy variables for the Eduaction column

df = pd.get_dummies(df, columns=['Education'])
df.head()
```

Out[81]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Mortgage	Personal_Loan	Securities_Account	CD_Account	Online	CreditCard	E
0	1	25	1	49	91107	4	1.6	0	0	1	0	0	0	
1	2	45	19	34	90089	3	1.5	0	0	1	0	0	0	
2	3	39	15	11	94720	1	1.0	0	0	0	0	0	0	
3	4	35	9	100	94112	1	2.7	0	0	0	0	0	0	
4	5	35	8	45	91330	4	1.0	0	0	0	0	0	1	

In [82]:
```
# Currently the dummy variables are True and False, we need to make it 1s and 0s.

for col in ["Education_Graduate", "Education_Professional", "Education_Undergrad"]:
  df[col] = df[col].astype(int)
```

In [83]:
```
df.head()
```

Out[83]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Mortgage	Personal_Loan	Securities_Account	CD_Account	Online	CreditCard	E
0	1	25	1	49	91107	4	1.6	0	0	1	0	0	0	
1	2	45	19	34	90089	3	1.5	0	0	1	0	0	0	
2	3	39	15	11	94720	1	1.0	0	0	0	0	0	0	
3	4	35	9	100	94112	1	2.7	0	0	0	0	0	0	
4	5	35	8	45	91330	4	1.0	0	0	0	0	0	1	

In [84]:
```
# Split the Data

X = df.drop(columns=['ID', 'Personal_Loan', 'ZIPCode'])
y = df['Personal_Loan']
```

In [85]:
```
# Split X and y into training and test set in 70:30 ratio

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

In [86]:
```
# Fit Linear Model

import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split # Sklearn package's randomized data splitting function


regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

Out[86]:

▾ LinearRegression

LinearRegression()

In [87]:
```
# Finding the coefficients for each variable and the intercept

for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, regression_model.coef_[idx]))
```

```
The coefficient for Age is -0.005830340667264938
The coefficient for Experience is 0.0061262713120217235
The coefficient for Income is 0.002865613233001863
The coefficient for Family is 0.030120456156665
The coefficient for CCAvg is 0.017078887769683835
The coefficient for Mortgage is 9.202942920009786e-05
The coefficient for Securities_Account is -0.06919883202815111
The coefficient for CD_Account is 0.32522827129942355
The coefficient for Online is -0.02580536206404267
The coefficient for CreditCard is -0.0438562200155962
The coefficient for Education_Graduate is 0.04436230205329969
The coefficient for Education_Professional is 0.0510163082973728
The coefficient for Education_Undergrad is -0.09537861035067265
```

In [88]:
```
# Access the intercept directly as a scalar
intercept = regression_model.intercept_
print("The intercept for the model is {}".format(intercept))
```

```
The intercept for the model is -0.0587430650011478
```

In [89]:
```
# Finding the score (R^2) for in-sample
regression_model.score(X_train, y_train)
```

Out[89]:
```
0.40176895932889156
```

In [90]:
```
# out of sample score (R^2)

regression_model.score(X_test, y_test)
```

Out[90]:
```
0.3818473592515367
```

Model Building

Model Evaluation Criterion

- Since the objective is a binary prediction of whether or not a liability customer will buy personal loans, to understand which customer attributes are more significant in driving purchases, and to identify which segment of customers to target more, we will choose **Accuracy**, **Precision**, **Recall**, **F1 Score** and the **Confusion Matrix** as the evaluation criteria.

Model Building

In [91]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classifi
```

- We will utilize the DecisionTreeClassifier function which is imported above, as well as using default 'gini' criteria to split the data.

In [92]:
```
dTree = DecisionTreeClassifier(criterion = 'gini', random_state=1)
dTree.fit(X_train, y_train)
```

Out[92]:
```
DecisionTreeClassifier(random_state=1)
```

In [93]:
```
# Scoring the Decision Tree

print("Accuracy on the Training Set:",dTree.score(X_train, y_train))
print("Accuracy on the Test Set:    ",dTree.score(X_test, y_test))
```

```
Accuracy on the Training Set: 1.0
Accuracy on the Test Set:     0.98
```

In [94]:
```
# Checking number of positives

y.value_counts(1)
```

Out[94]:
```
Personal_Loan
0    0.904
1    0.096
Name: proportion, dtype: float64
```

- The ratio of positives to negatives is 1 to 9, meaning for every customer who accepts the personal loan, approximately 9 customers

do not accept it.

In [95]:
```
## Function to create confusion matrix
def make_confusion_matrix(model,y_actual,labels=[1, 0]):

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

In [96]:
```
##  Function to calculate recall score
def get_recall_score(model):
    '''
    model : classifier to predict values of X

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

In [97]:
```
# Making predictions

y_pred = dTree.predict(X_test)
```

In [98]:
```
# Evaluating the model

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

In [99]:
```
# Printing the evaluation

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"F1 Score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

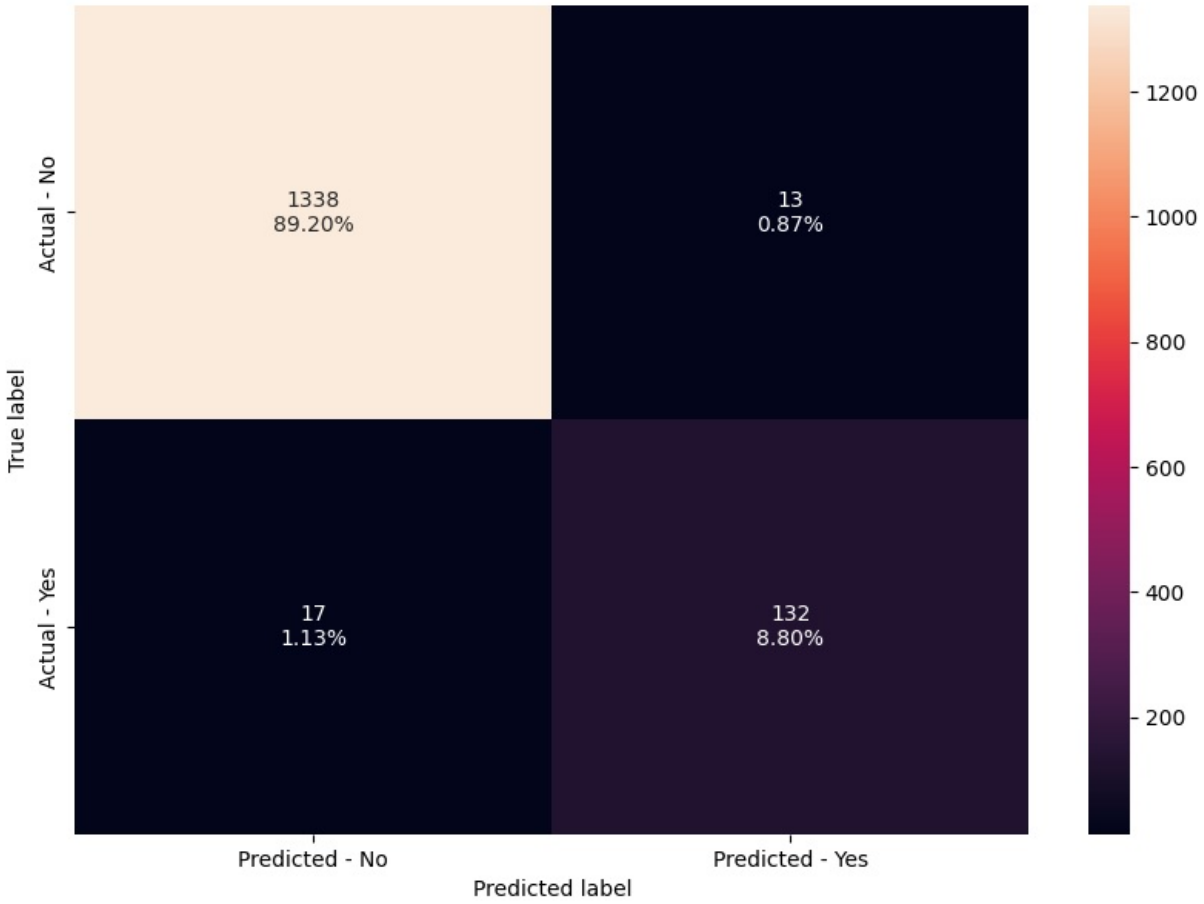
```
Accuracy: 0.98
Precision: 0.9103448275862069
F1 Score: 0.8979591836734694
Confusion Matrix:
[[1338   13]
 [  17  132]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1351
           1       0.91      0.89      0.90       149

    accuracy                           0.98      1500
   macro avg       0.95      0.94      0.94      1500
weighted avg       0.98      0.98      0.98      1500
```

From the print statements, we can observe:

- The accuracy (which measures the overall correctness of the model) resulted at 98% which is a good score and suggests that the model is performing well overall.
- The precision ended up at 91%, meaning that when the model predicts a customer to buy a personal loan, it is correct 91% of the time. This is important for minimizing false positives, which means predicting a customer will buy a loan when they end up not buying.
- For the recall, the score ended up being 89%, this is a crucial score that must be a high enough percentage for identifying as many potential buyers as possible, and to make sure none of them are missed.
- In F1 Score resulted at 90%, which indicates that the model maintains a good trade-off between the precision and the recall, it also shows the model's robustness in identifying the customers who bought the loans.

In [ ]:
```
make_confusion_matrix(dTree,y_test)
```

In [100…
```
make_confusion_matrix(dTree,y_test)
```



In [101…
```
# Retrieving the recall score on the training and the test set
get_recall_score(dTree)
```

```
Recall on training set :  1.0
Recall on test set :  0.8859060402684564
```

From the Confusion Matrix, as seen above, we can observe:

- There are 1338 customers correctly predicted not to buy a loan.
- 13 Customers who had incorrectly predicted to buy a loan.
- 17 Customers who had actually bought a loan but the model incorrectly predicted not to buy a loan.
- There are 132 customers correctly predicted to buy a loan.

In [102…
```
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1351
           1       0.91      0.89      0.90       149

    accuracy                           0.98      1500
   macro avg       0.95      0.94      0.94      1500
weighted avg       0.98      0.98      0.98      1500
```

From the classification report, we can observe that:

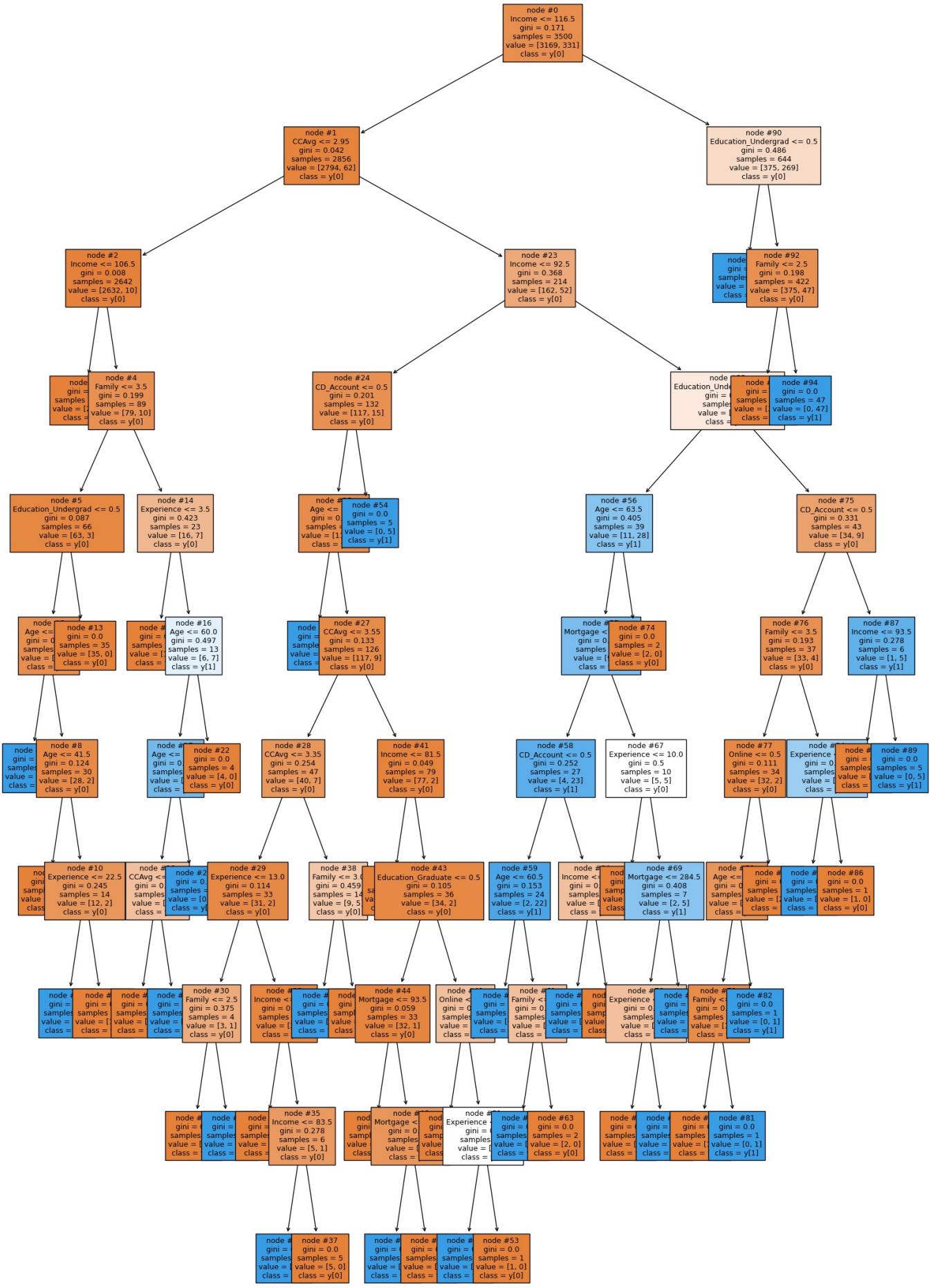
- Class 0 (Non-loan buyers):
  - The precision, recall, and F1 scores are all 0.99, which indicates that the model performs exceptionally well in identifying those customers who did not accept the personal loan offer.
- Class 1 (Loan buyers):
  - the precision score being 0.91, the recall score of 0.89, and the F1 score of 0.90, suggest that the model is very effective in identifying buyers, although there is a slight trade-off between precision and the recall scores.

Visualizng the Decision Tree

In [103…
```
# Printing the customer attributes

cust_attributes = list(X.columns)
print(cust_attributes)
```

```
['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Mortgage', 'Securities_Account', 'CD_Account', 'Online', 'C
reditCard', 'Education_Graduate', 'Education_Professional', 'Education_Undergrad']
```

In [104…]
```
plt.figure(figsize=(20,30))
tree.plot_tree(dTree,feature_names=cust_attributes,filled=True,fontsize=9,node_ids=True,class_names=True)
plt.show()
```



In [105…]
```
# Text report displaying the rules of a decision tree

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

```
|--- Income <= 116.50
|   |--- CCAvg <= 2.95
|   |   |--- Income <= 106.50
|   |   |   |--- weights: [2553.00, 0.00] class: 0
|   |   |--- Income >  106.50
|   |   |   |--- Family <= 3.50
|   |   |   |   |--- Education_Undergrad <= 0.50
|   |   |   |   |   |--- Age <= 28.50
|   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- Age >  28.50
|   |   |   |   |   |   |--- Age <= 41.50
|   |   |   |   |   |   |   |--- weights: [16.00, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  41.50
|   |   |   |   |   |   |   |--- Experience <= 22.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- Experience >  22.50
|   |   |   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |--- Education_Undergrad >  0.50
|   |   |   |   |   |--- weights: [35.00, 0.00] class: 0
|   |   |   |--- Family >  3.50
|   |   |   |   |--- Experience <= 3.50
|   |   |   |   |   |--- weights: [10.00, 0.00] class: 0
|   |   |   |   |--- Experience >  3.50
|   |   |   |   |   |--- Age <= 60.00
|   |   |   |   |   |   |--- Age <= 32.50
|   |   |   |   |   |   |   |--- CCAvg <= 2.35
|   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  2.35
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- Age >  32.50
|   |   |   |   |   |   |   |--- weights: [0.00, 6.00] class: 1
|   |   |   |   |   |--- Age >  60.00
|   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- Income <= 92.50
|   |   |   |--- CD_Account <= 0.50
|   |   |   |   |--- Age <= 26.50
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- Age >  26.50
|   |   |   |   |   |--- CCAvg <= 3.55
|   |   |   |   |   |   |--- CCAvg <= 3.35
|   |   |   |   |   |   |   |--- Experience <= 13.00
|   |   |   |   |   |   |   |   |--- Family <= 2.50
|   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Family >  2.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- Experience >  13.00
|   |   |   |   |   |   |   |   |--- Income <= 82.50
|   |   |   |   |   |   |   |   |   |--- weights: [23.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Income >  82.50
|   |   |   |   |   |   |   |   |   |--- Income <= 83.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- Income >  83.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  3.35
|   |   |   |   |   |   |   |--- Family <= 3.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |--- Family >  3.00
|   |   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  3.55
|   |   |   |   |   |   |--- Income <= 81.50
|   |   |   |   |   |   |   |--- weights: [43.00, 0.00] class: 0
|   |   |   |   |   |   |--- Income >  81.50
|   |   |   |   |   |   |   |--- Education_Graduate <= 0.50
|   |   |   |   |   |   |   |   |--- Mortgage <= 93.50
|   |   |   |   |   |   |   |   |   |--- weights: [26.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Mortgage >  93.50
|   |   |   |   |   |   |   |   |   |--- Mortgage <= 104.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- Mortgage >  104.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Education_Graduate >  0.50
|   |   |   |   |   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Online >  0.50
|   |   |   |   |   |   |   |   |   |--- Experience <= 29.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- Experience >  29.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- CD_Account >  0.50
|   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |--- Income >  92.50
|   |   |   |--- Education_Undergrad <= 0.50
|   |   |   |   |--- Age <= 63.50
|   |   |   |   |   |--- Mortgage <= 172.00
|   |   |   |   |   |   |--- CD_Account <= 0.50
|   |   |   |   |   |   |   |--- Age <= 60.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 21.00] class: 1
|   |   |   |   |   |   |   |--- Age >  60.50
```

```
|   |   |   |   |   |   |   |   |--- Family <= 2.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- Family >  2.50
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- CD_Account >  0.50
|   |   |   |   |   |   |   |   |--- Income <= 98.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- Income >  98.50
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- Mortgage >  172.00
|   |   |   |   |   |   |   |--- Experience <= 10.00
|   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  10.00
|   |   |   |   |   |   |   |   |--- Mortgage <= 284.50
|   |   |   |   |   |   |   |   |   |--- Experience <= 25.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Experience >  25.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- Mortgage >  284.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |--- Age >  63.50
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |--- Education_Undergrad >  0.50
|   |   |   |   |   |--- CD_Account <= 0.50
|   |   |   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |   |   |   |--- Age <= 55.00
|   |   |   |   |   |   |   |   |   |--- Family <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Family >  2.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- Age >  55.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- Online >  0.50
|   |   |   |   |   |   |   |   |--- weights: [20.00, 0.00] class: 0
|   |   |   |   |   |   |--- Family >  3.50
|   |   |   |   |   |   |   |--- Experience <= 21.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- Experience >  21.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- CD_Account >  0.50
|   |   |   |   |   |   |--- Income <= 93.50
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- Income >  93.50
|   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|--- Income >  116.50
|   |--- Education_Undergrad <= 0.50
|   |   |--- weights: [0.00, 222.00] class: 1
|   |--- Education_Undergrad >  0.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [375.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [0.00, 47.00] class: 1
```

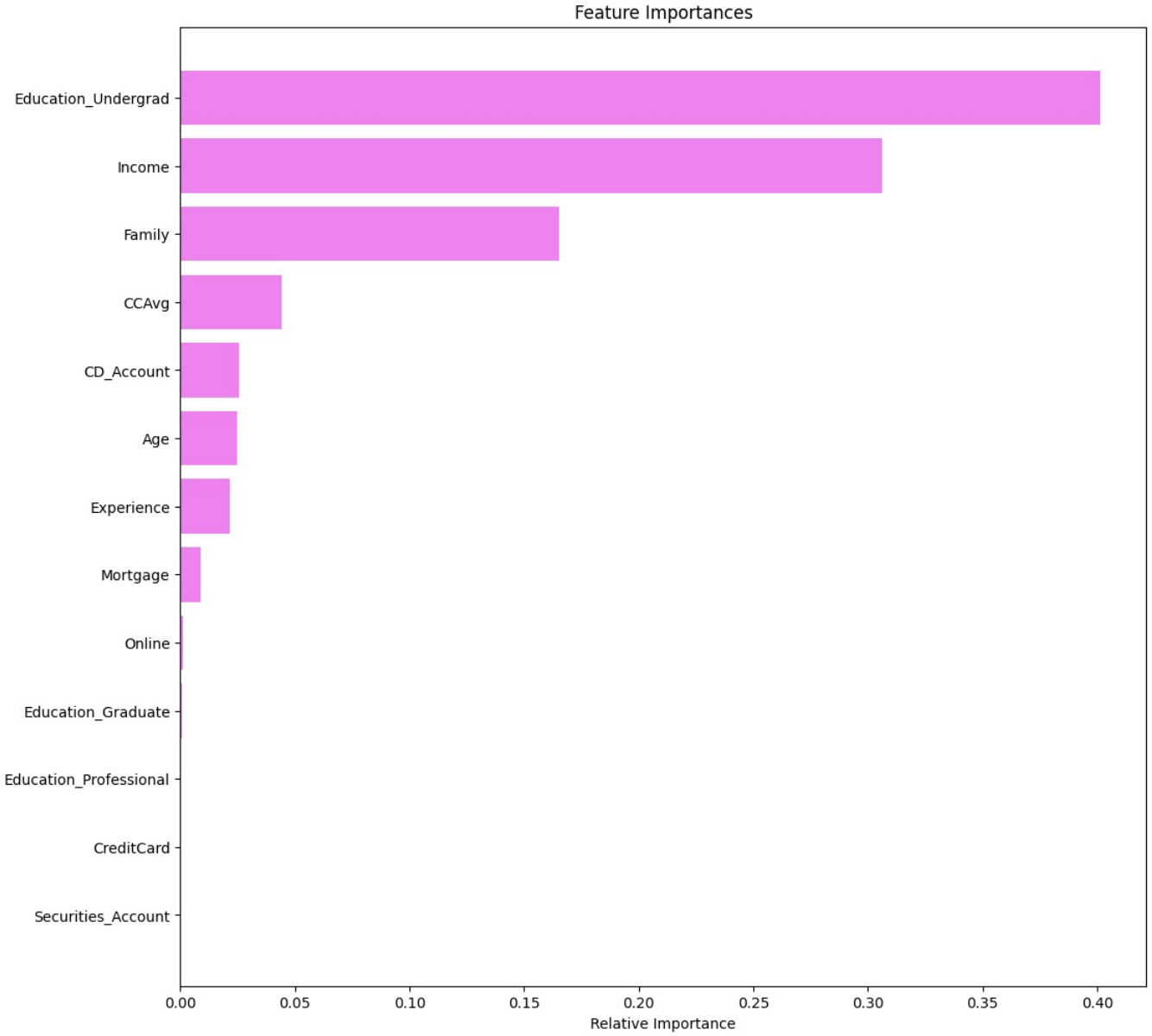
- As seen in the above textual representation of the decision tree, some could view this as a better method of easily understanding the information in a decision tree rather than looking at the cramped-up visual depiction.

In [106…
```
# Text version for the important features in the decision tree

print (pd.DataFrame(dTree.feature_importances_, columns = ["Importance"], index = X_train.columns).sort_values(
```

```
                      Importance
Education_Undergrad     0.401465
Income                  0.306244
Family                  0.165104
CCAvg                   0.044408
CD_Account              0.025711
Age                     0.024890
Experience              0.021425
Mortgage                0.008792
Online                  0.001117
Education_Graduate      0.000843
Securities_Account      0.000000
CreditCard              0.000000
Education_Professional  0.000000
```

In [107…
```
# Graphical figure of the important features

importances = dTree.feature_importances_
indices = np.argsort(importances)

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

```
plt.show()
```



As you can see from the bar graph above, the Undergrad Education attribute is the most important variable for predicting a liability customer will buy personal loans. Following that, the customer's annual income and their family size are the runner-ups on the relative importance.

Model Performance Improvement

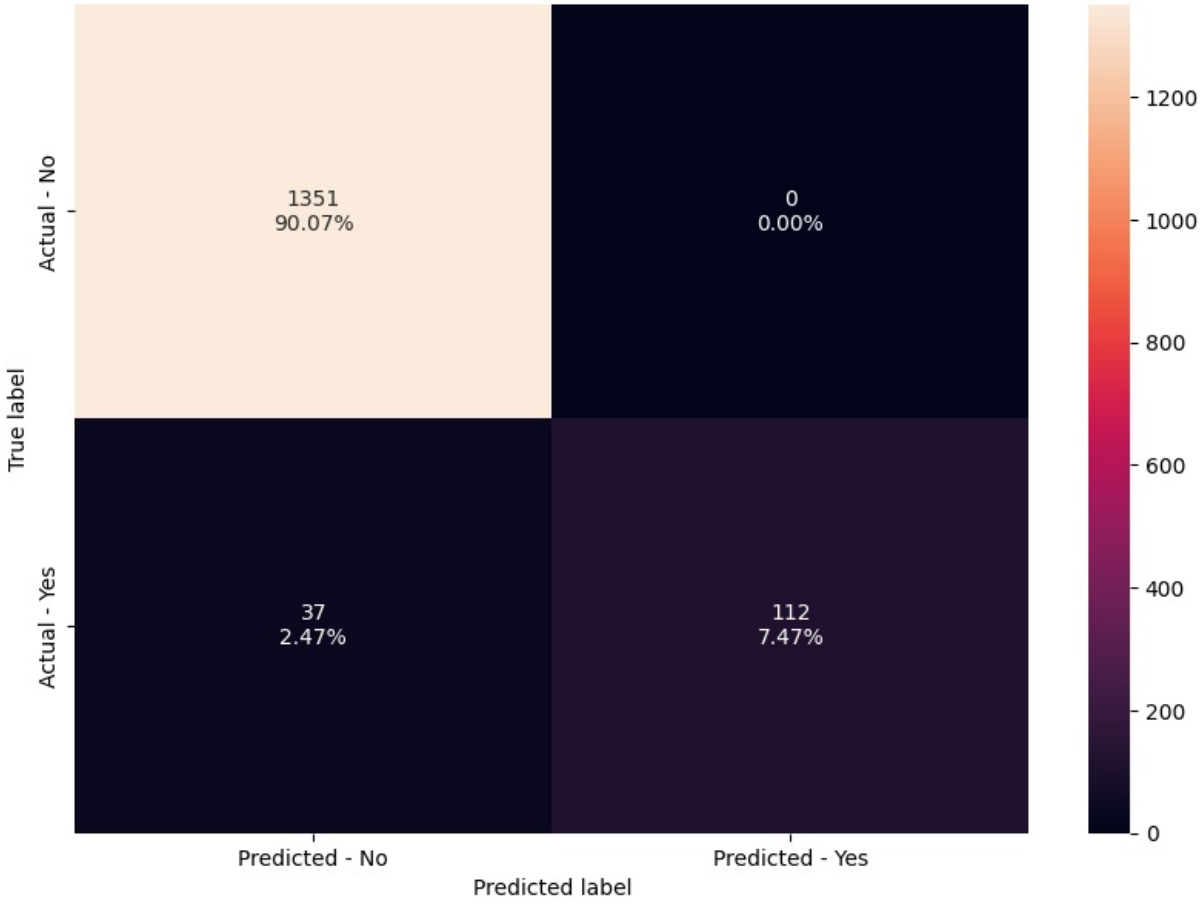
Pre-Pruning

Attempting to reduce over fitting by limiting the max_depth of the tree to 3.

In [108…
```
dTree1 = DecisionTreeClassifier(criterion = 'gini',max_depth=3,random_state=1)
dTree1.fit(X_train, y_train)
```

Out[108]:
```
DecisionTreeClassifier(max_depth=3, random_state=1)
```

In [109…
```
make_confusion_matrix(dTree1, y_test)
```



In [110…

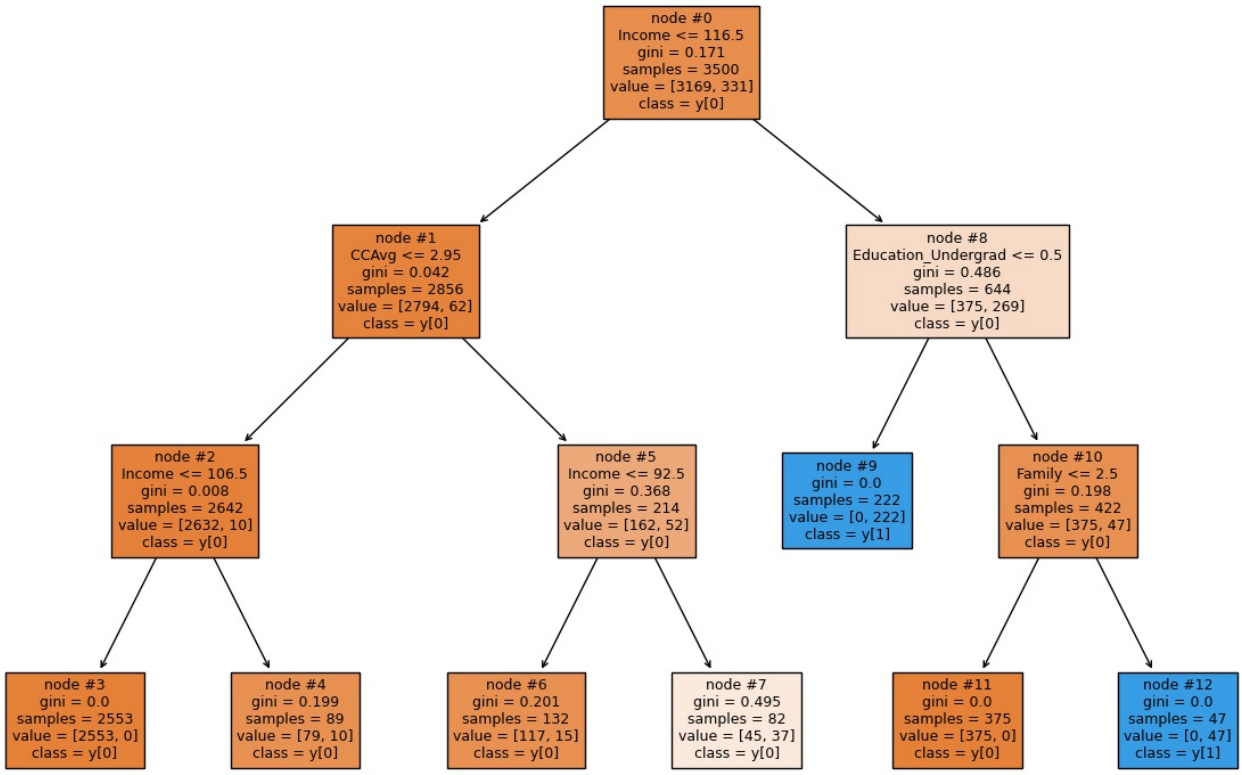
```
# Accuracy report on train and test set
print("Accuracy on training set : ",dTree1.score(X_train, y_train))
print("Accuracy on test set : ",dTree1.score(X_test, y_test))

# Recall scores on train and test set
get_recall_score(dTree1)
```

```
Accuracy on training set :  0.9822857142857143
Accuracy on test set :  0.9753333333333334
Recall on training set :  0.8126888217522659
Recall on test set :  0.7516778523489933
```

Since reducing the max-depth to 3, the recall score on the training set reduced from 1 to 0.81. However, this is not an improvement, because the model before the reduction was balanced, now it is towards the underfitting scale.

In [111…

```
# Decision Tree Graphic

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

tree.plot_tree(dTree1,feature_names=cust_attributes,filled=True,fontsize=9,node_ids=True,class_names=True)
plt.show()
```



The decision tree, compared to before is now readable, but the recall score on the test set did not improve with the reduction of the depth.

In [112…
```
# Textual representation showing the rules of a decision tree

print(tree.export_text(dTree1,feature_names=cust_attributes,show_weights=True))
```

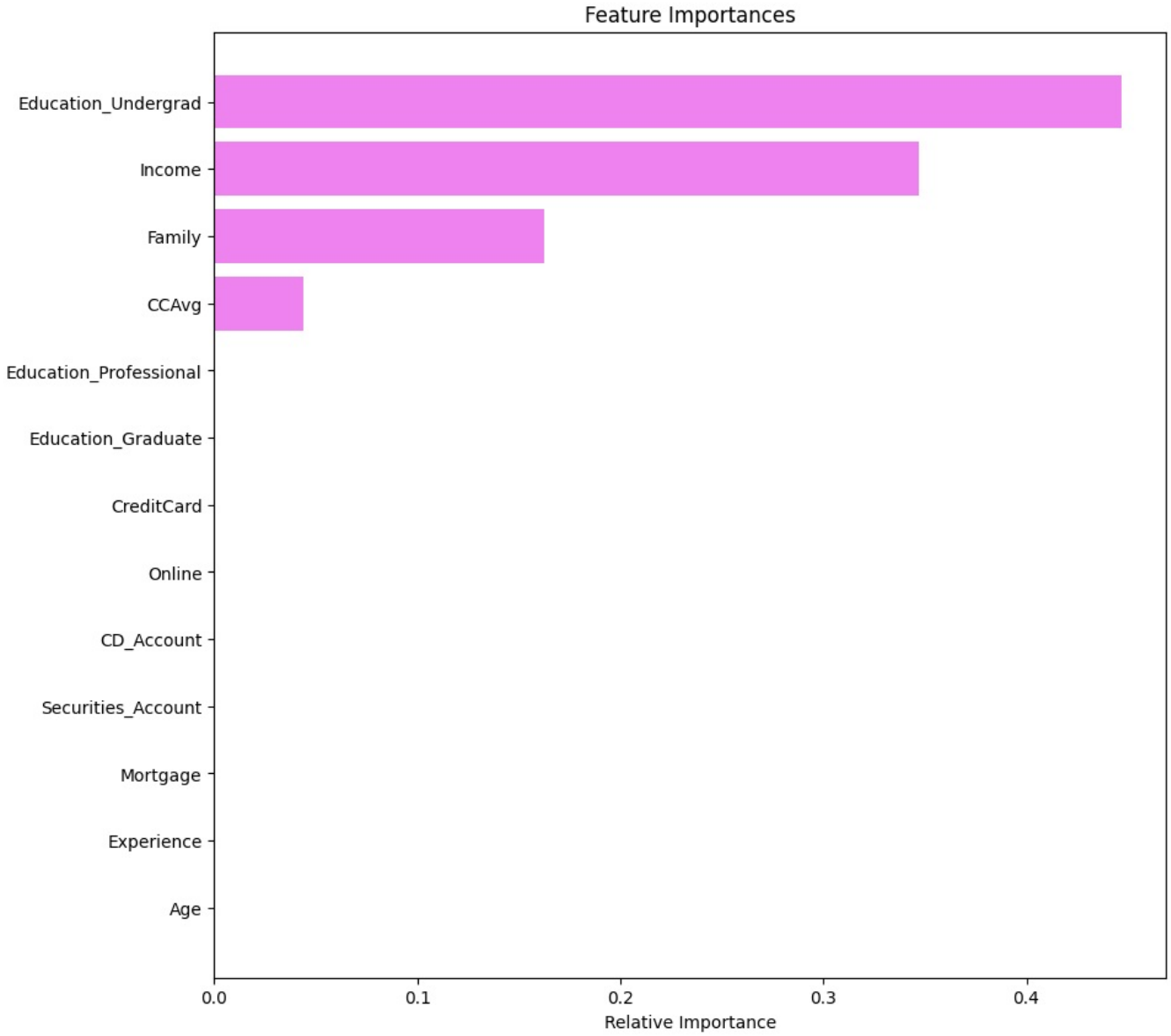
```
|--- Income <= 116.50
|   |--- CCAvg <= 2.95
|   |   |--- Income <= 106.50
|   |   |   |--- weights: [2553.00, 0.00] class: 0
|   |   |--- Income >  106.50
|   |   |   |--- weights: [79.00, 10.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- Income <= 92.50
|   |   |   |--- weights: [117.00, 15.00] class: 0
|   |   |--- Income >  92.50
|   |   |   |--- weights: [45.00, 37.00] class: 0
|--- Income >  116.50
|   |--- Education_Undergrad <= 0.50
|   |   |--- weights: [0.00, 222.00] class: 1
|   |--- Education_Undergrad >  0.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [375.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [0.00, 47.00] class: 1
```

In [113…
```
# Chart displaying the importance of the customer attributes

print (pd.DataFrame(dTree1.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by = '
```

```
                         Imp
Education_Undergrad     0.446593
Income                  0.346997
Family                  0.162372
CCAvg                   0.044038
Age                     0.000000
Experience              0.000000
Mortgage                0.000000
Securities_Account      0.000000
CD_Account              0.000000
Online                  0.000000
CreditCard              0.000000
Education_Graduate      0.000000
Education_Professional  0.000000
```

In [114…
```
# Graphical figure of the important customer attributes

importances = dTree1.feature_importances_
indices = np.argsort(importances)
```

```
plt.figure(figsize=(10,10))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [cust_attributes[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



The customer attributes that were on the top 3 in the previous model are the same in this reduced max depth model as well. This can show how setting max_depth to 3 is not enough to improve the model. We will instead use the grid search method.

Using GridSearch for Hyperparameter Tuning

In [115…
```
from sklearn.model_selection import GridSearchCV
```

In [116…
```
# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1)

# Grid of parameters to choose from
## add from article
parameters = {'max_depth': np.arange(1,10),
              'min_samples_leaf': [1, 2, 5, 7, 10,15,20],
              'max_leaf_nodes' : [2, 3, 5, 10],
              'min_impurity_decrease': [0.001,0.01,0.1]
             }

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

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

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

# Fit the best algorithm to the data.
```

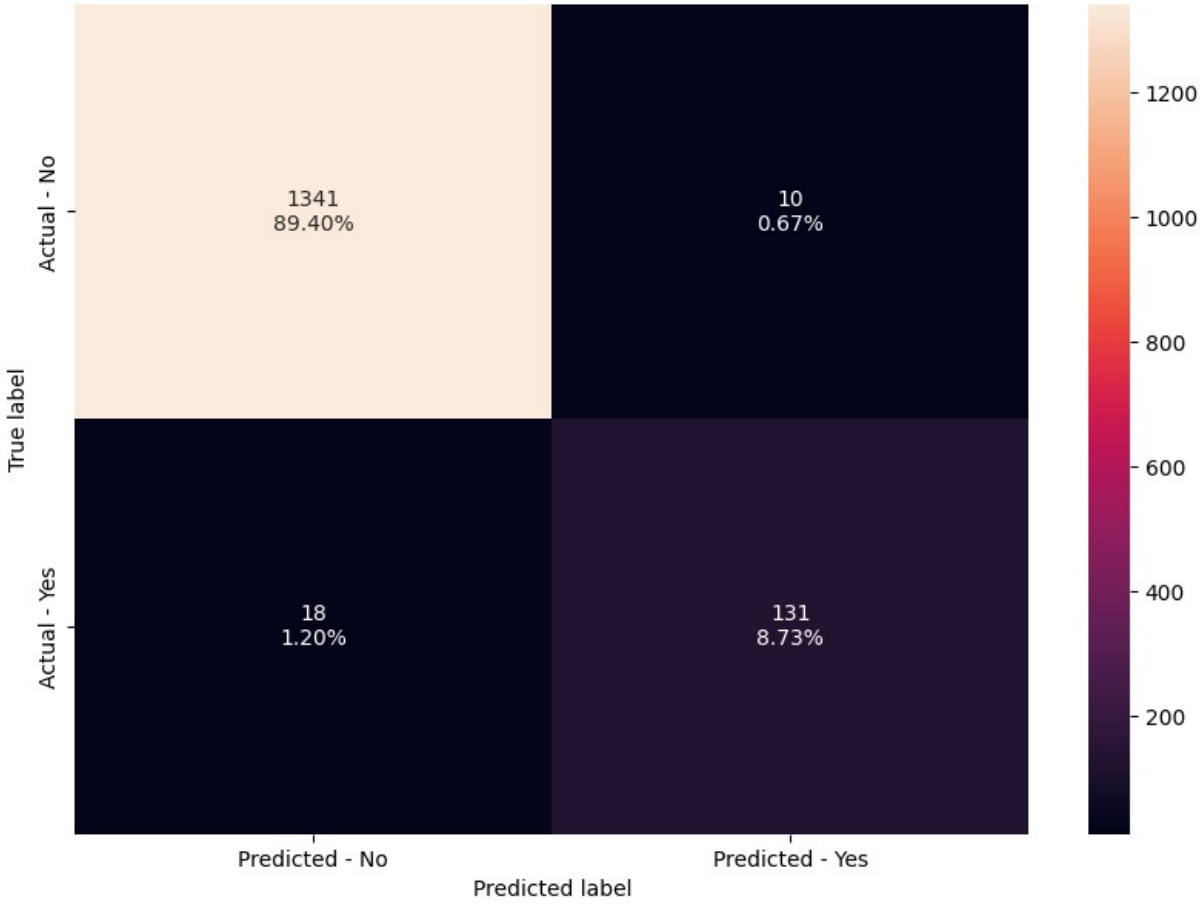
```
estimator.fit(X_train, y_train)
```

Out[116]:

```
DecisionTreeClassifier(max_depth=5, max_leaf_nodes=10,
                       min_impurity_decrease=0.001, random_state=1)
```

In [117…

```
# Confustion Matrix with the tuned hyperparameter decision tree just created

make_confusion_matrix(estimator,y_test)
```



In [118…

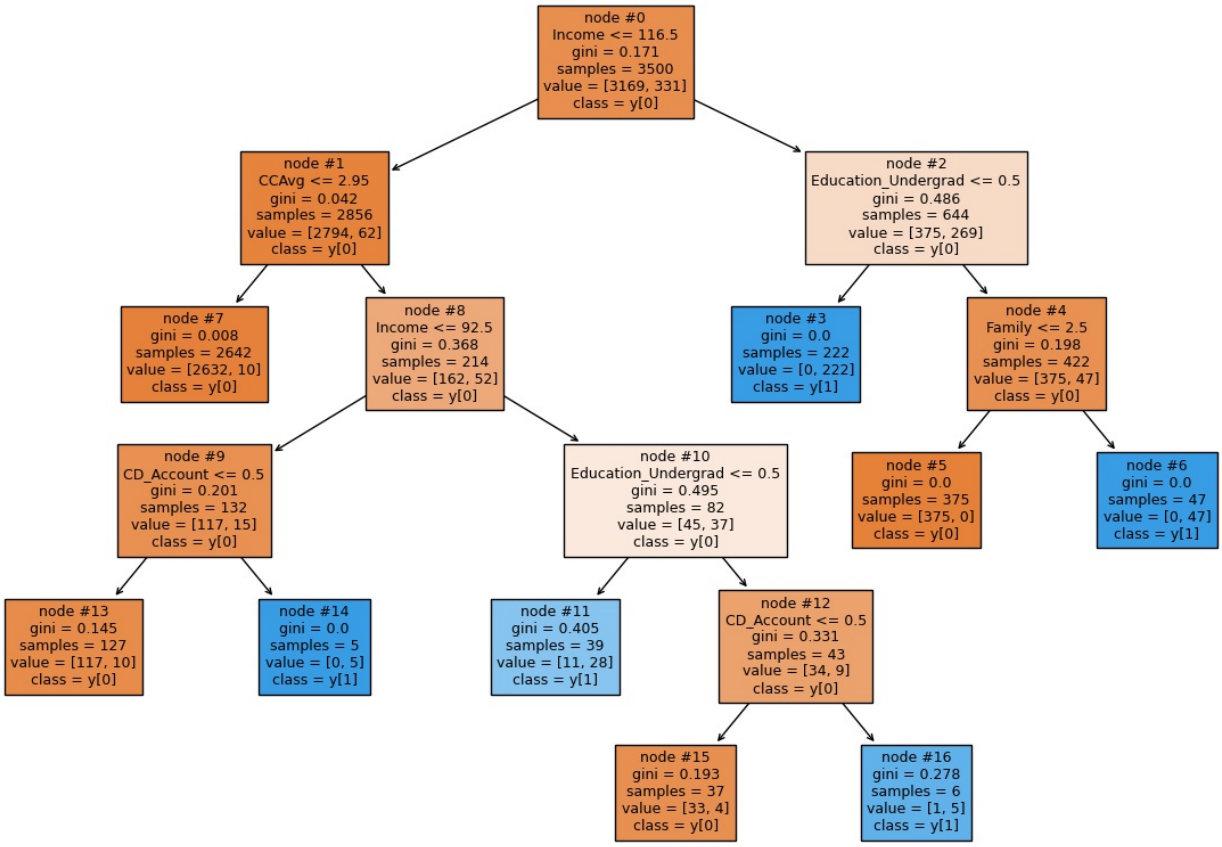
```
# Accuracy scores on train and test set
print("Accuracy on training set : ",estimator.score(X_train, y_train))
print("Accuracy on test set : ",estimator.score(X_test, y_test))

# Recall scores on train and test set
get_recall_score(estimator)
```

```
Accuracy on training set :  0.9897142857142858
Accuracy on test set :  0.9813333333333333
Recall on training set :  0.9274924471299094
Recall on test set :  0.8791946308724832
```

- The recall score has not improved from the original model.
- Although compared to the model before this with a recall score on the training set being 0.93 and test set 0.88, it has improved by a significant amount. *With that being said, the model is now on the scale of being balanced.

In [119…

```
# Plotting the Decision Tree of the hyperparameter model

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

tree.plot_tree(estimator,feature_names=cust_attributes,filled=True,fontsize=9,node_ids=True,class_names=True)
plt.show()
```



In [120…

```
# Textual report displaying the rules of the decision tree

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

```
|--- Income <= 116.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [2632.00, 10.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- Income <= 92.50
|   |   |   |--- CD_Account <= 0.50
|   |   |   |   |--- weights: [117.00, 10.00] class: 0
|   |   |   |--- CD_Account >  0.50
|   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |--- Income >  92.50
|   |   |   |--- Education_Undergrad <= 0.50
|   |   |   |   |--- weights: [11.00, 28.00] class: 1
|   |   |   |--- Education_Undergrad >  0.50
|   |   |   |   |--- CD_Account <= 0.50
|   |   |   |   |   |--- weights: [33.00, 4.00] class: 0
|   |   |   |   |--- CD_Account >  0.50
|   |   |   |   |   |--- weights: [1.00, 5.00] class: 1
|--- Income >  116.50
|   |--- Education_Undergrad <= 0.50
|   |   |--- weights: [0.00, 222.00] class: 1
|   |--- Education_Undergrad >  0.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [375.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [0.00, 47.00] class: 1
```

In [121…

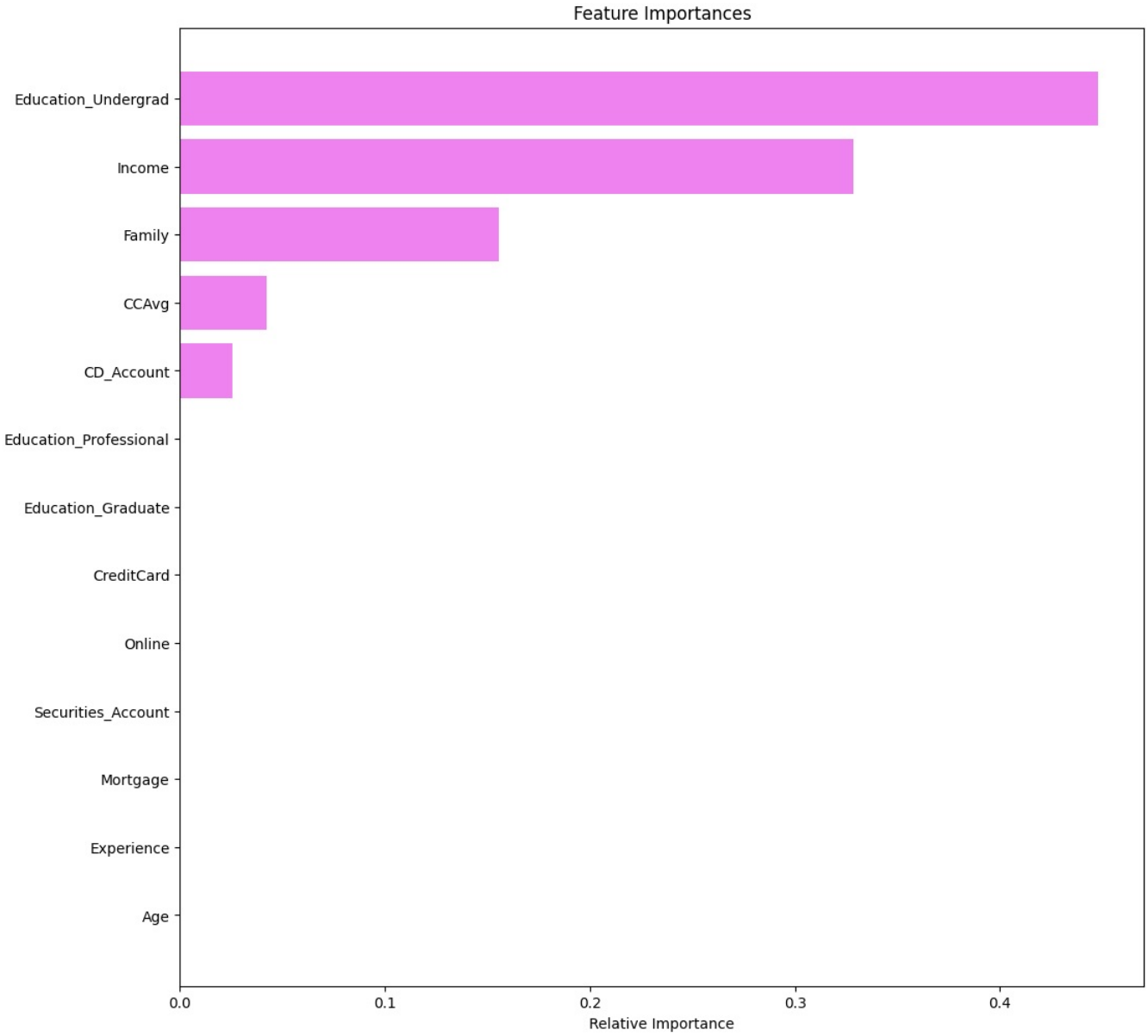
```
# Ranking the importance of customer attributes  in the tree building

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

```
                         Imp
Education_Undergrad     0.447999
Income                  0.328713
Family                  0.155711
CCAvg                   0.042231
CD_Account              0.025345
Age                     0.000000
Experience              0.000000
Mortgage                0.000000
Securities_Account      0.000000
Online                  0.000000
CreditCard              0.000000
Education_Graduate      0.000000
Education_Professional  0.000000
```

In [122…

```
importances = estimator.feature_importances_
```

```
indices = np.argsort(importances)

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



- Just like the previous two models, the hierarchy of important customer attributes is the same with undergrad education level being the most important, and income & family coming after.

Post-Pruning (Cost Complexity Pruning)

- Post-pruning might provide better results than pre-pruning because their is a high probability that some of the hyperparameters might have been abandoned.

In [123…
```
# Total impurity of leaves vs effective alphas of pruned tree

clf = DecisionTreeClassifier(random_state=1)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

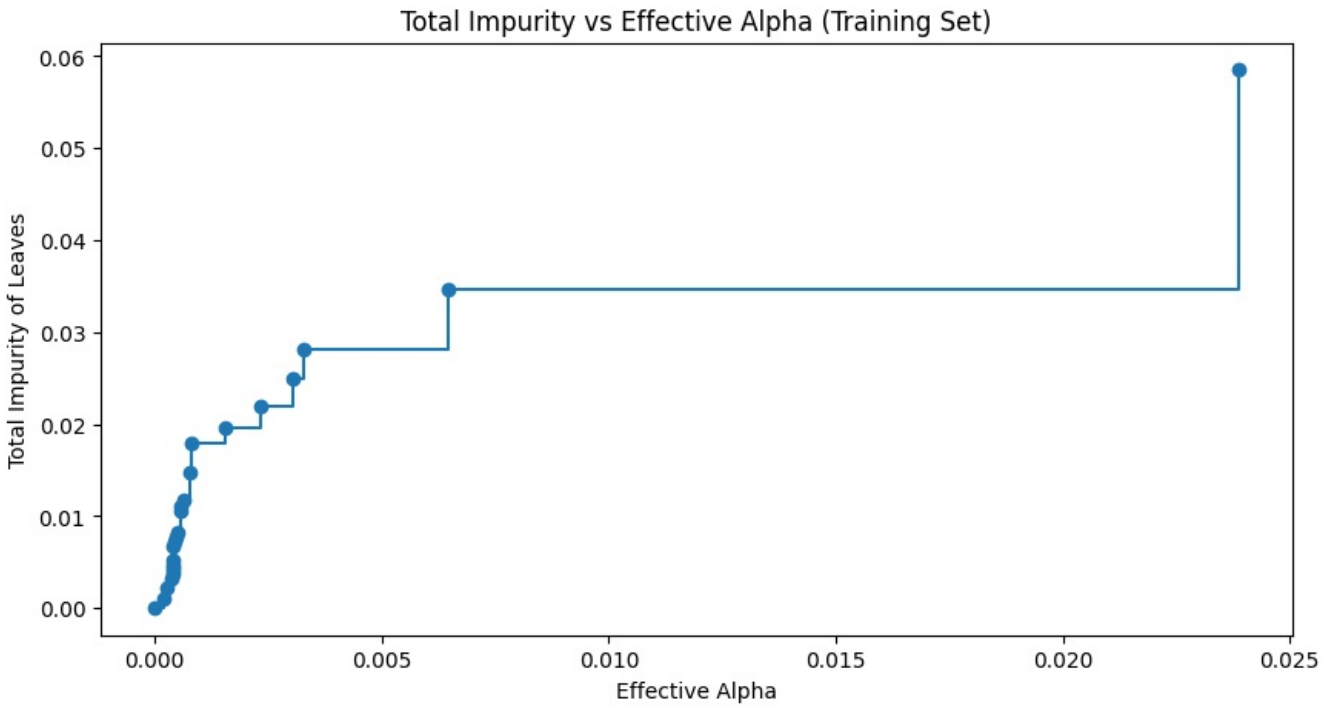
In [124…
```
# Showcasing table of the alphas with their impuritiy measures
pd.DataFrame(path)
```

Out[124]:

	ccp_alphas	impurities
0	0.000000	0.000000
1	0.000186	0.001114
2	0.000268	0.002188
3	0.000359	0.003263
4	0.000381	0.003644
5	0.000381	0.004025
6	0.000381	0.004406
7	0.000381	0.004787
8	0.000381	0.005168
9	0.000409	0.006804
10	0.000435	0.007240
11	0.000476	0.007716
12	0.000508	0.008224
13	0.000578	0.010537
14	0.000582	0.011119
15	0.000621	0.011740
16	0.000769	0.014817
17	0.000792	0.017985
18	0.001552	0.019536
19	0.002333	0.021869
20	0.003024	0.024893
21	0.003294	0.028187
22	0.006473	0.034659
23	0.023866	0.058525
24	0.056365	0.171255

In [125…

```
# Graphing the table created above

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("Effective Alpha")
ax.set_ylabel("Total Impurity of Leaves")
ax.set_title("Total Impurity vs Effective Alpha (Training Set)")
plt.show()
```



Use the effective alphas above, to create a decision tree. The last value in ccp_alphas is the alpha value that prunes the whole tree, leaving the tree with one node left.

In [126…

```
# Code to create a decision tree & state the number of nodes in the tree
```

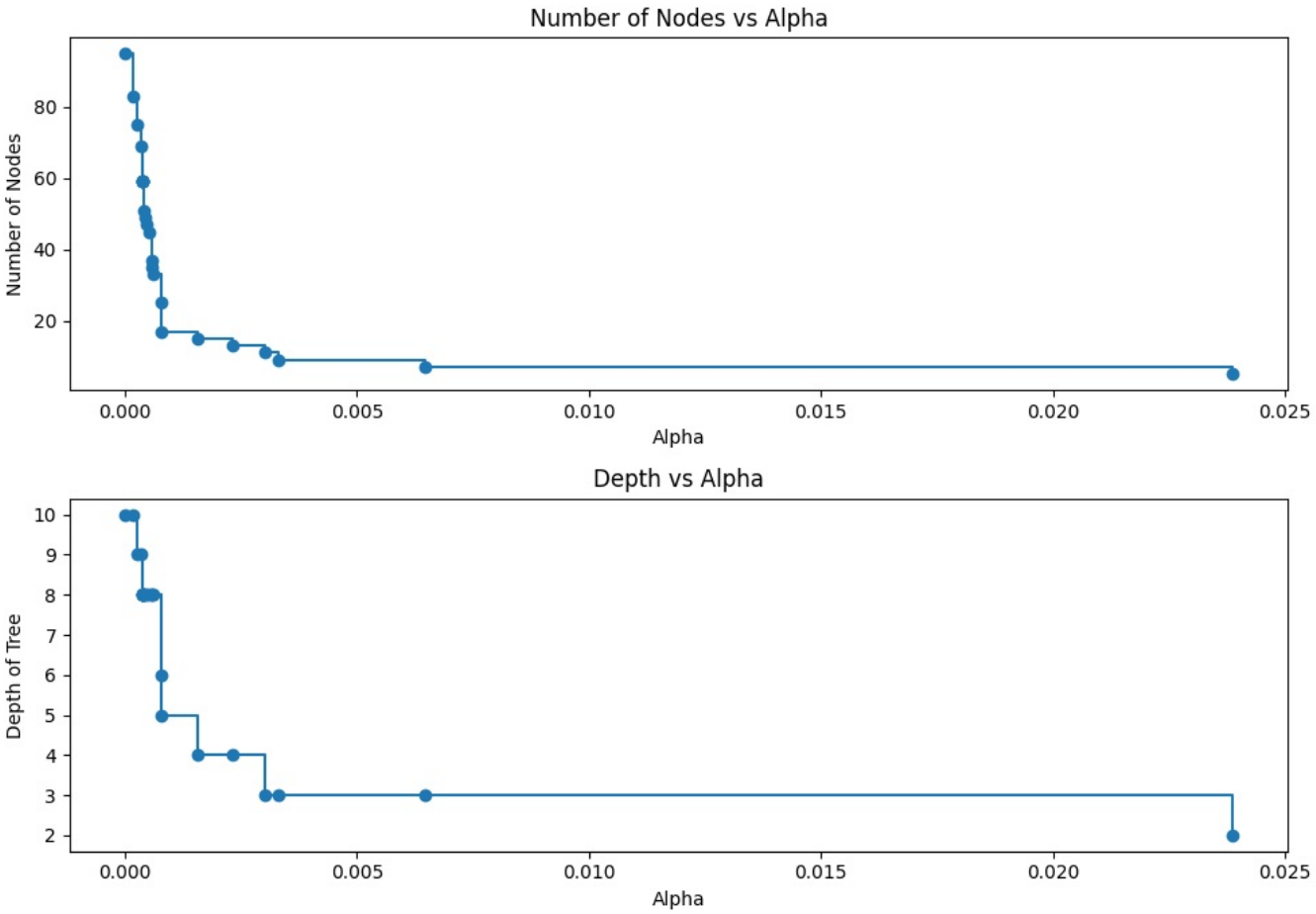
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("The number of nodes in the tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

```
The number of nodes in the tree is: 1 with ccp_alpha: 0.056364969335601575
```

In what remains, we remove the final element in clfs & ccp_alphas, due to it being the trivial tree with only one node.

In [127…

```
# Displaying the comparison between the number of nodes and the tree depth

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

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



- As seen in the two graphs above, when the number of nodes and the tree depth decreases, the alpha increases in size.

Accuracy vs Alpha (Training & Test Sets)

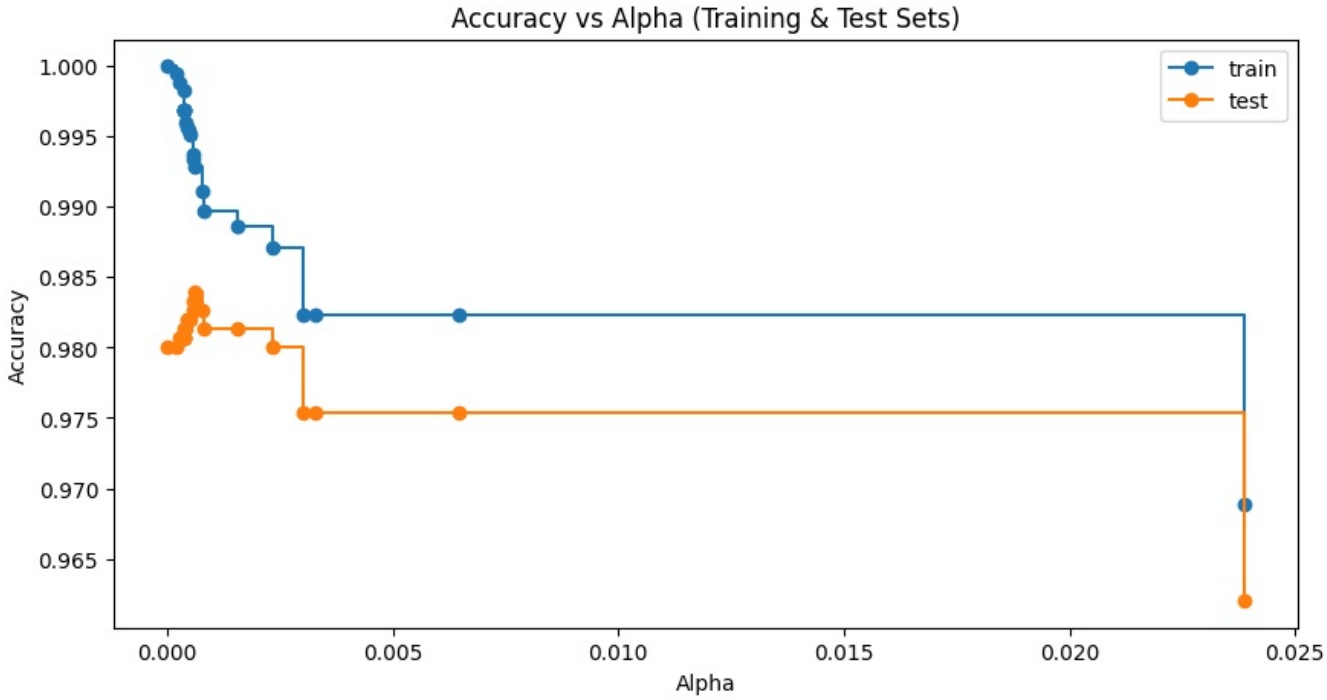
In [128…

```
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 [129…

```
fig, ax = plt.subplots(figsize=(10,5))
ax.set_xlabel("Alpha")
ax.set_ylabel("Accuracy")
ax.set_title("Accuracy vs Alpha (Training & Test Sets)")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
```

```
plt.show()
```



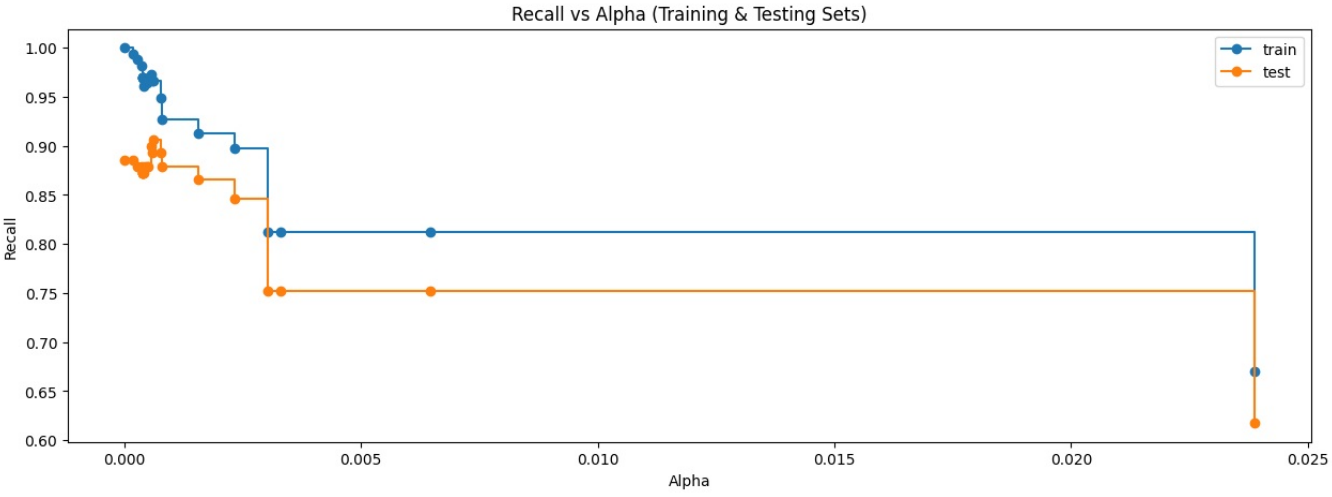
- As seen in the graph above, the training and test sets have a similar pattern in which they decrease as the alpha increases.
- Since accuracy did not result well, recall will be the next option for further testing.

Creating Training & Testing Sets for Recall

In [130…
```
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 [131…
```
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 [132…
```
fig, ax = plt.subplots(figsize=(15,5))
ax.set_xlabel("Alpha")
ax.set_ylabel("Recall")
ax.set_title("Recall vs Alpha (Training & Testing Sets)")
ax.plot(ccp_alphas, recall_train, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, recall_test, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```

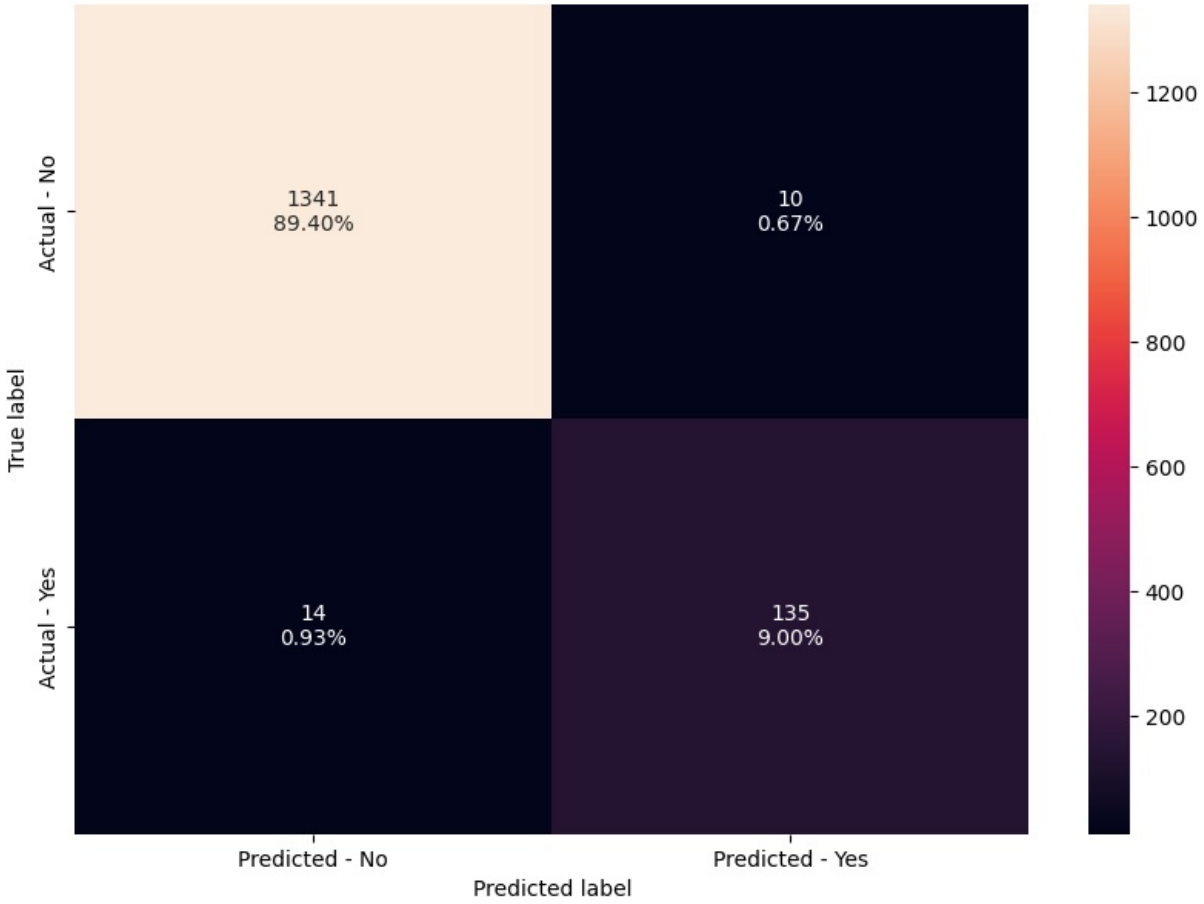


- Just like the Acurracy vs Alpha, the training and testing recall scores follow the same pattern with the training scoring being slightly better in the end.

In [133…
```
# Creating a model where we get highest train and test recall score from the graph above

index_best_model = np.argmax(recall_test)
best_model = clfs[index_best_model]
print(best_model)
```

```
DecisionTreeClassifier(ccp_alpha=0.0006209286209286216, random_state=1)
```

In [134…
```
# Constructing a decision tree with the model just created above

make_confusion_matrix(best_model,y_test)
```



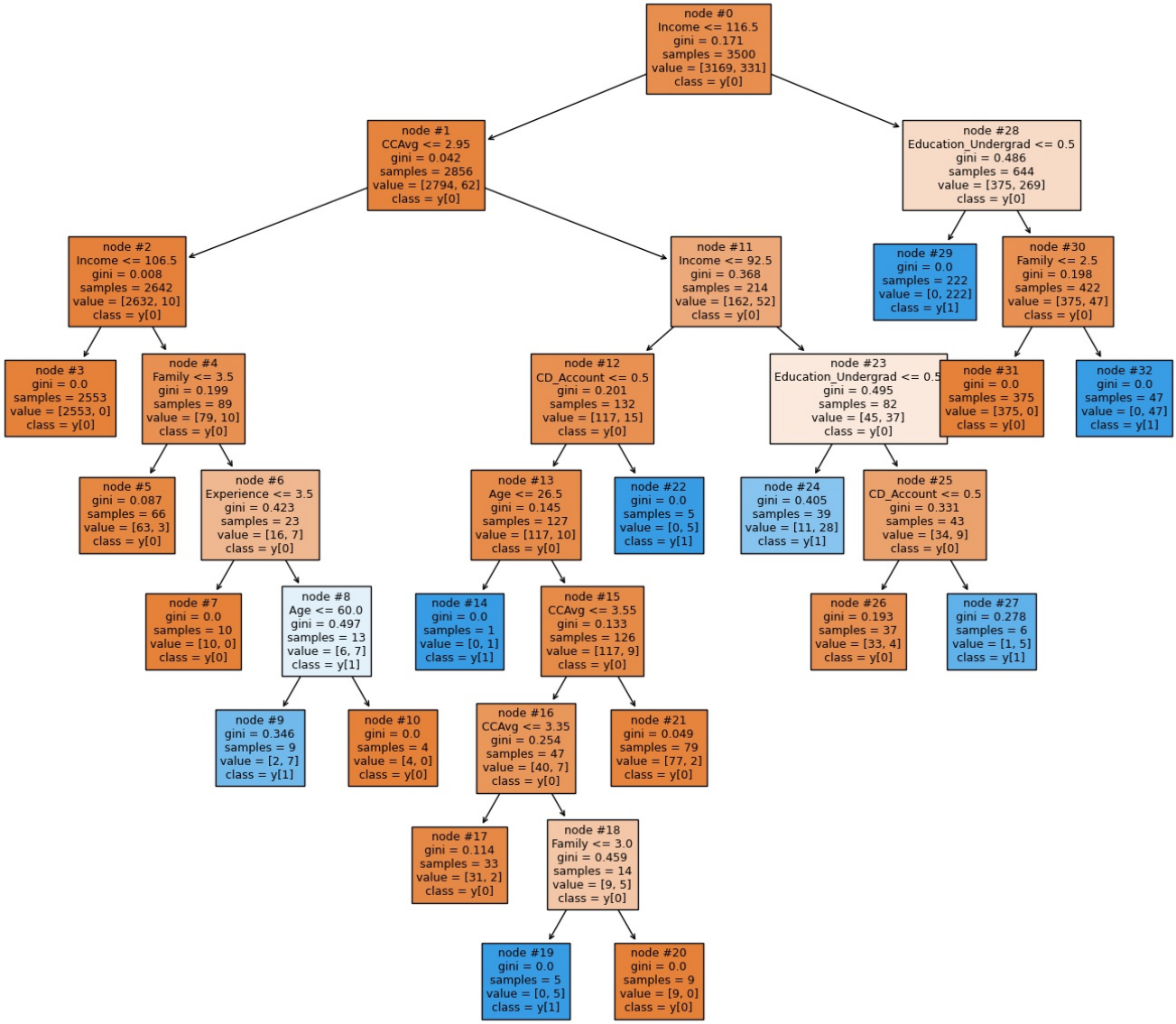
In [135…
```
# Gathering the recall scores on both the training and the testing set

get_recall_score(best_model)
```

```
Recall on training set :  0.9667673716012085
Recall on test set :  0.9060402684563759
```

- Compared to all the previous models created, the post-pruning has resulted in providing the best scores for both the training and the testing sets.

Visualizing the Decision Tree of the Post-Pruned Model

In [136…
```
plt.figure(figsize=(17,15))

tree.plot_tree(best_model,feature_names=cust_attributes,filled=True,fontsize=9,node_ids=True,class_names=True)
plt.show()
```



- The decision tree above appears to be very complex in size.

In [137…

```
# Textual report of the decision tree

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

```
|--- Income <= 116.50
|   |--- CCAvg <= 2.95
|   |   |--- Income <= 106.50
|   |   |   |--- weights: [2553.00, 0.00] class: 0
|   |   |--- Income >  106.50
|   |   |   |--- Family <= 3.50
|   |   |   |   |--- weights: [63.00, 3.00] class: 0
|   |   |   |--- Family >  3.50
|   |   |   |   |--- Experience <= 3.50
|   |   |   |   |   |--- weights: [10.00, 0.00] class: 0
|   |   |   |   |--- Experience >  3.50
|   |   |   |   |   |--- Age <= 60.00
|   |   |   |   |   |   |--- weights: [2.00, 7.00] class: 1
|   |   |   |   |   |--- Age >  60.00
|   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- Income <= 92.50
|   |   |   |--- CD_Account <= 0.50
|   |   |   |   |--- Age <= 26.50
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- Age >  26.50
|   |   |   |   |   |--- CCAvg <= 3.55
|   |   |   |   |   |   |--- CCAvg <= 3.35
|   |   |   |   |   |   |   |--- weights: [31.00, 2.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  3.35
|   |   |   |   |   |   |   |--- Family <= 3.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |--- Family >  3.00
|   |   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  3.55
|   |   |   |   |   |   |--- weights: [77.00, 2.00] class: 0
|   |   |   |--- CD_Account >  0.50
|   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |--- Income >  92.50
|   |   |   |--- Education_Undergrad <= 0.50
|   |   |   |   |--- weights: [11.00, 28.00] class: 1
|   |   |   |--- Education_Undergrad >  0.50
|   |   |   |   |--- CD_Account <= 0.50
|   |   |   |   |   |--- weights: [33.00, 4.00] class: 0
|   |   |   |   |--- CD_Account >  0.50
|   |   |   |   |   |--- weights: [1.00, 5.00] class: 1
|--- Income >  116.50
|   |--- Education_Undergrad <= 0.50
|   |   |--- weights: [0.00, 222.00] class: 1
|   |--- Education_Undergrad >  0.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [375.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [0.00, 47.00] class: 1
```

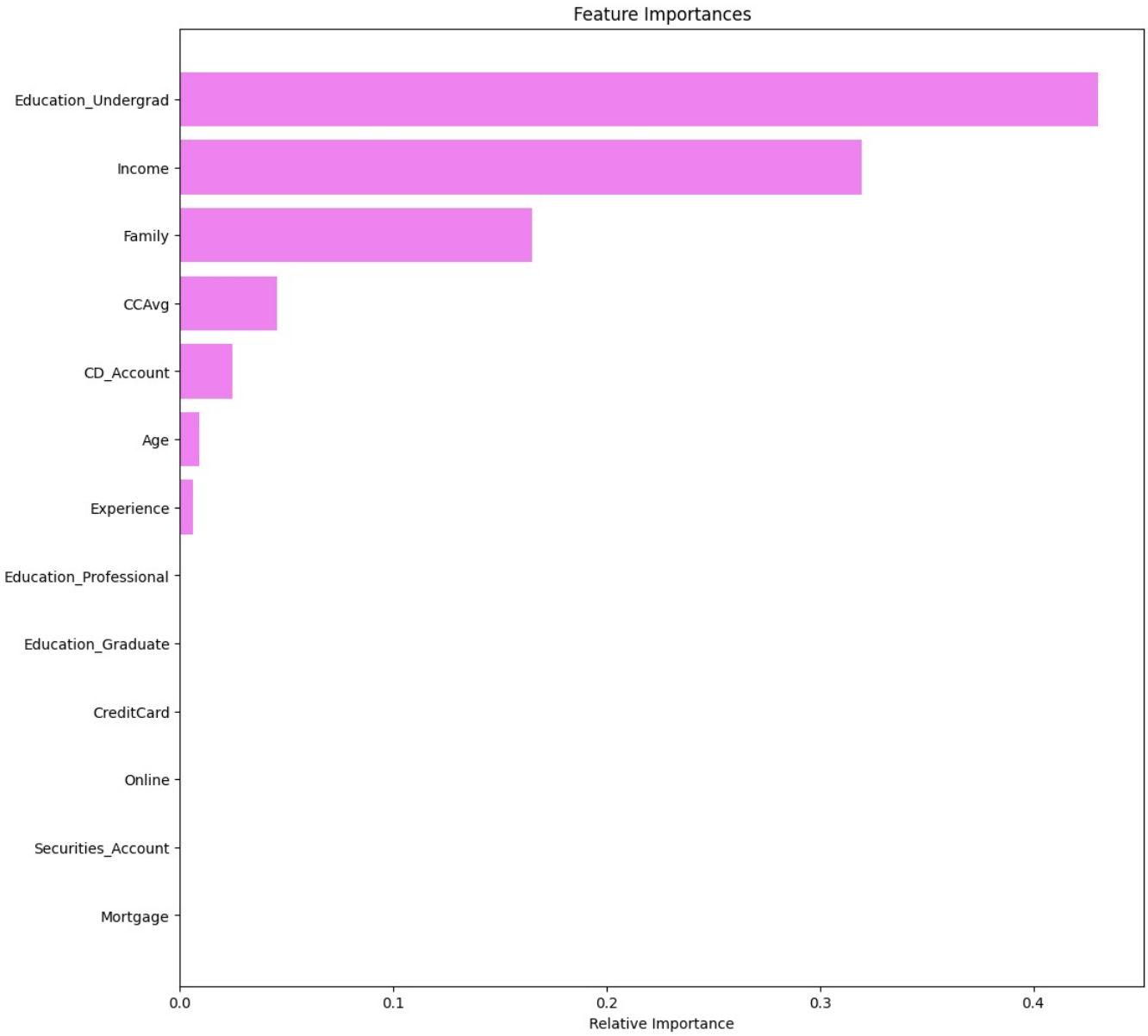
In [138…

```
# Table for the importance of the customer attributes

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

```
                         Imp
Education_Undergrad     0.430462
Income                  0.319735
Family                  0.165225
CCAvg                   0.045288
CD_Account              0.024353
Age                     0.009066
Experience              0.005871
Mortgage                0.000000
Securities_Account      0.000000
Online                  0.000000
CreditCard              0.000000
Education_Graduate      0.000000
Education_Professional  0.000000
```

In [139…

```
# Graphical representation of the important customer attributes

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

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



- Just like the previous models created, the undergrad education level resulted in being the most important attribute following the customer's annual income and their family size.

Model Comparison and Final Model Selection

In [140…

```
# Chart to display the recall scores of all the models that were tested and trained

model_comparison = pd.DataFrame({'Model':['Initial Decision Tree','Pre-Pruned Decision Tree','Decision Tree wit
                                          'Decision Tree with Post-Pruning'], 'Train Recall Score':[1,0.81,0.93,
model_comparison
```

Out[140]:

	Model	Train Recall Score	Test Recall Score
0	Initial Decision Tree	1.00	0.88
1	Pre-Pruned Decision Tree	0.81	0.75
2	Decision Tree with Hyperparameter Tuning	0.93	0.88
3	Decision Tree with Post-Pruning	0.97	0.91

To conclude, the model that ended up performing the best was the post-pruned decision tree for the testing set, with a score of 0.91.

Actionable Insights and Business Recommendations

- The post-pruning model, which turned out to have the best recall scores, can be used to predict whether a liability customer will buy personal loans or not correctly 91% of the time.
- The undergrad education level, the annual income of the customer, and the customer's family size were the most important customer

attributes in predicting whether a customer would buy personal loans.

- From the post-pruned decision tree, if the customer's annual income is less than or equal to 116 thousand dollars, if there is half the amount of undergrad education level for customers, and if the family size is less than or equal to 2.5, then the customer will likely not choose to buy a personal loan.
    - Thus, it is vital for AllLife Bank to keep an eye on these values to detect a customer not purchasing a personal loan.

What recommendations would you suggest to the bank?

- Some recommendations I would suggest would be:
- more attributes listed for the customer for the model to better predict a personal loan purchase. For instance, some attributes that should be added are:
- Where the customer is located.
- What type of house they are living in, such as an apartment, townhome, and a normal house?
- The boolean attributes do not have any importance as seen in the feature graphs above, such as 'Securities_Account', 'CD_Account', and 'Online'. Therefore the bank should be considering improvements in those attributes.
