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

1. ID: Customer ID
2. Age: Customer's age in completed years
3. Experience: #years of professional experience
4. Income: Annual income of the customer (in thousand dollars)
5. ZIP Code: Home Address ZIP code.
6. Family: the Family size of the customer
7. CCAvg: Average spending on credit cards per month (in thousand dollars)
8. Education: Education Level. 1: Undergrad; 2: Graduate;3: Advanced/Professional
9. Mortgage: Value of house mortgage if any. (in thousand dollars)
10. Personal_Loan: Did this customer accept the personal loan offered in the last campaign? (0: No, 1: Yes)
11. Securities_Account: Does the customer have securities account with the bank? (0: No, 1: Yes)
12. CD_Account: Does the customer have a certificate of deposit (CD) account with the bank? (0: No, 1: Yes)
13. Online: Do customers use internet banking facilities? (0: No, 1: Yes)
14. CreditCard: Does the customer use a credit card issued by any other Bank (excluding All life Bank)? (0: No, 1: Yes)

Importing the necessary Libraries

In [1]:

```
# to load and manipulate data
import pandas as pd
import numpy as np

# to visualize data
import matplotlib.pyplot as plt
import seaborn as sns

# to split data into training and test sets
from sklearn.model_selection import train_test_split

# to build decision tree model
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# to tune different models
from sklearn.model_selection import GridSearchCV

# to compute classification metrics
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    recall_score,
    precision_score,
    f1_score,
)

# Lets Load the data now
```

In [2]:

```
from google.colab import drive
 drive.mount('/content/drive')
```

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

In [3]:

```
# loading data into a pandas dataframe
path = '/content/drive/My Drive/python/Loan_Modelling.csv'
loan_modelling = pd.read_csv(path)

# creating a copy of the data
data = loan_modelling.copy()
```

Data Overview

In [4]:

```
data.head(5)
```

Out[4]:

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

In [5]:

```
data.tail(5)
```

Out[5]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Account	CD_Accoun
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

Checking the shape of the data

In [6]:

```
data.shape
```

Out[6]:

```
(5000, 14)
```

Observations: The data has 5000 rows and 14 columns

In [7]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   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: All the columns are numeric and hence no need for hot encoding later. There are no categorical columns. Also, there are no null values

In [8]:

```
data.describe(include="all")
```

Out[8]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000
mean	2500.500000	45.338400	20.104600	73.774200	93169.257000	2.396400	1.937938	1.881000	56.498800	0
std	1443.520003	11.463166	11.467954	46.033729	1759.455086	1.147663	1.747659	0.839869	101.713802	0
min	1.000000	23.000000	-3.000000	8.000000	90005.000000	1.000000	0.000000	1.000000	0.000000	0
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1

Observations:

1. The mean age is 45.3 and 75% of the data are 55 and below.
2. The mean family size is more than 2 and 75% of the family have 3 in them.
3. 75% of the people are advanced in their education.
4. 75% of the people make around 98K.
5. A very small % in expereince have a -ve in it, but since its less than 1% it can be ignored.

In [9]:

```
data.isnull().sum()
```

Out[9]:

	0
ID	0
Age	0
Experience	0
Income	0
ZIPCode	0
Family	0
CCAvg	0
Education	0
Mortgage	0
Personal_Loan	0
Securities_Account	0
CD_Account	0
Online	0
CreditCard	0

dtype: int64

In [10]:

```
data.duplicated().sum()
```

Out[10]:

```
np.int64(0)
```

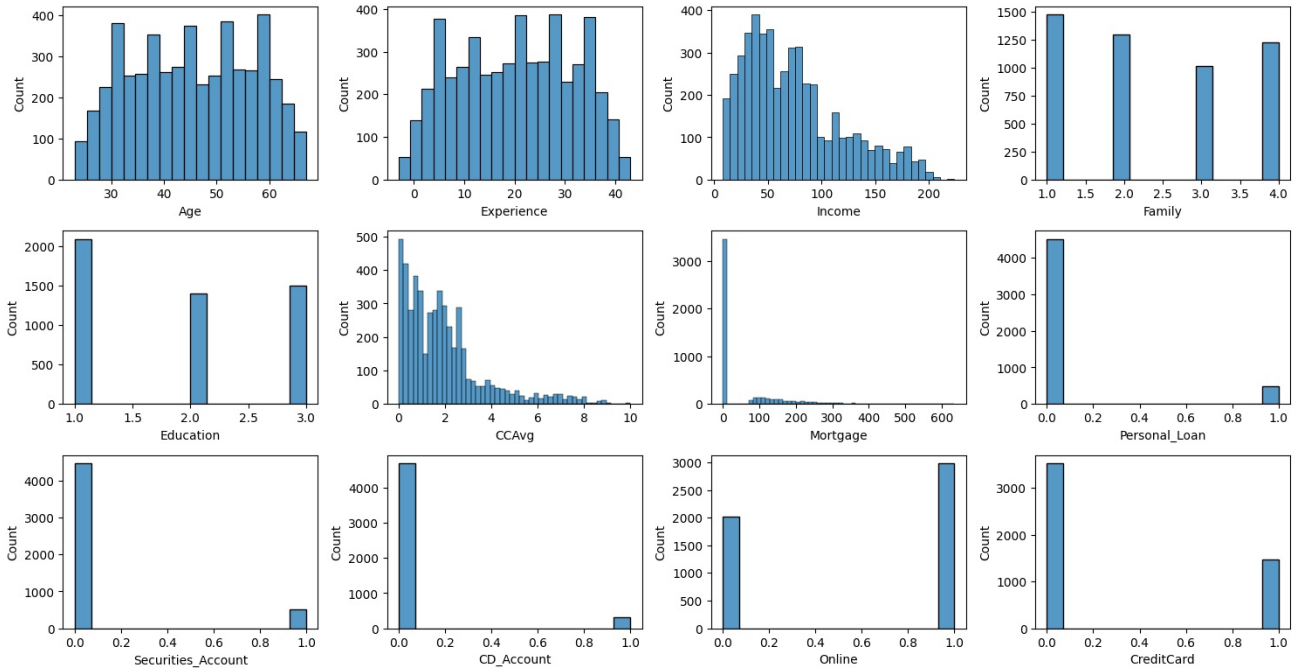
Observations: There are no duplicated values nor are there any values with nulls.

Exploratory Data analysis

Univariate Analysis

In [11]:

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

# defining the list of numerical features to plot
num_features = ['Age', 'Experience', 'Income' , 'Family', 'Education' , 'CCAvg', 'Mortgage', 'Personal_Loan' , 'S
ecurities_Account' , 'CD_Account', 'Online', 'CreditCard']

# plotting the histogram for each numerical feature
for i, feature in enumerate(num_features):
    plt.subplot(4, 4, i+1)    # assign a subplot in the main plot
    sns.histplot(data=data, x=feature)    # plot the histogram

plt.tight_layout();   # to add spacing between plots
```

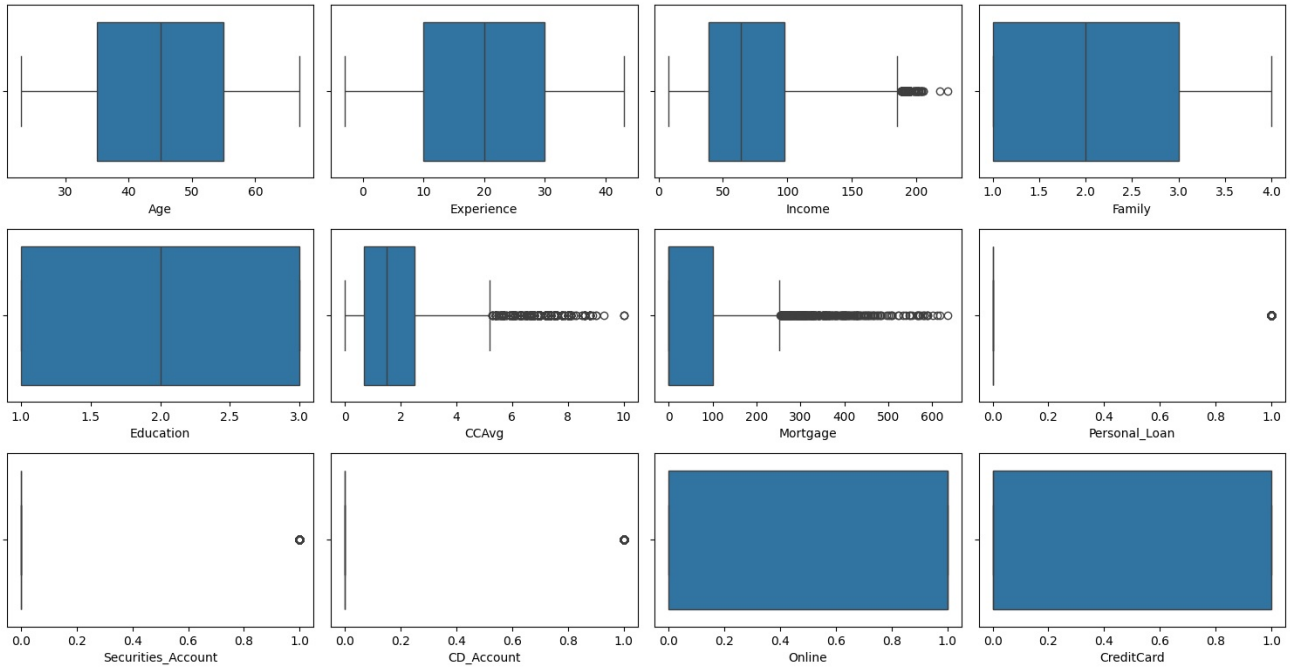


Observations:

1. Income and CCAvg exhibit a right skewed distribution and are most probably very important in our analysis. The average spending on the credit card decreases on the x axis.
2. The age seems to be pretty well distributed and during initial analysis hence doesnt seem to a big factor on the decison making. (This suggests a even distribution around age for approvals).
3. The mortgage is not available on the vast majority of the approved ones so, that cannot be a factor in future approvals. Higher mortage can only be afforded by people with higher income, but since those values are not available (for Mortage), we are not able to strongly say if this variable will be important in the marketing campaigns or not, yet.
4. Online, Securities Account also seem to be unlikely factors since they seem to be farily distributed.
5. The experience has some negative values and is also pretty well distributed. Even distribution suggests that people with a particular experience level dont seem to have the loan approved.
6. Family seems to be well distributed with 1, 2, 3 and 4 pretty evenly distributed and feels less probable as being a factor on loan approvals.

In [12]:

```
# defining the figure size
plt.figure(figsize=(15, 10))

# plotting the boxplot for each numerical feature
for i, feature in enumerate(num_features):
    plt.subplot(4, 4, i+1)    # assign a subplot in the main plot
    sns.boxplot(data=data, x=feature)    # plot the histogram

plt.tight_layout();    # to add spacing between plots
```



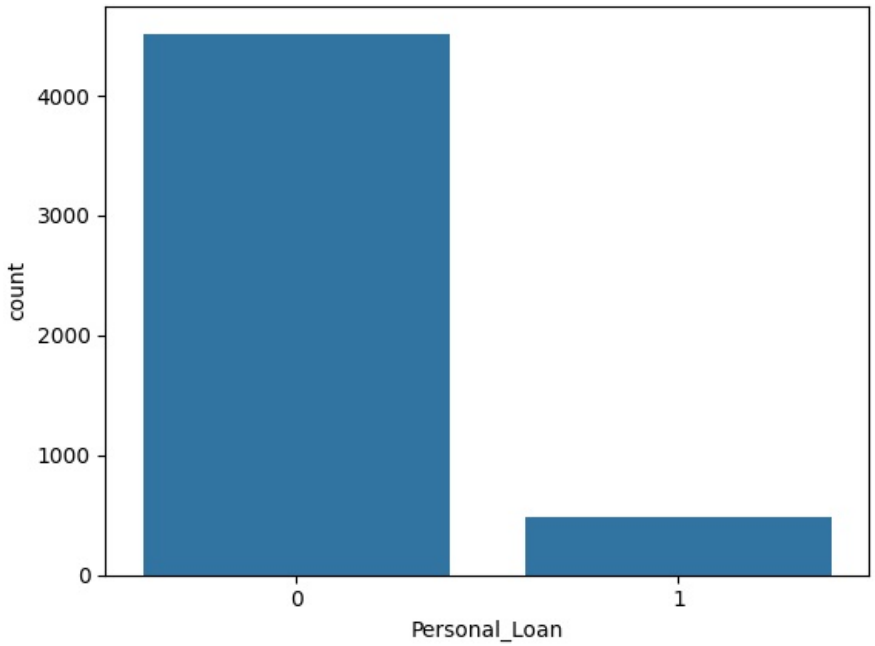
Observations:

1. There are outliers in CCAvg, Mortgage and income and no whiskers for Education, Online and CreditCard.
2. This could mean that CCAvg, Income could be factors in the decision making.

In [13]:

```
# checking the distribution of the categories in Approval
print(100*data['Personal_Loan'].value_counts(normalize=True), '\n')

# plotting the count plot for Approval
sns.countplot(data=data, x='Personal_Loan');
```

```
Personal_Loan
0    90.4
1     9.6
Name: proportion, dtype: float64
```



Approximately 10% have been approved. So we have to targe the rest 90%

In [14]:

```
# Calculate the percentage of people with an online account whose personal loan was approved.
online_account_holders = data[data['Online'] == 1]
approved_loans_online = online_account_holders[online_account_holders['Personal_Loan'] == 1]
percentage = (len(approved_loans_online) / len(online_account_holders)) * 100
print(f"Percentage of people with an online account whose personal loan was approved: {percentage:.2f}%")
```

```
Percentage of people with an online account whose personal loan was approved: 9.75%
```

In [15]:

```
# Calculate the percentage of people with a credit card whose personal loan was approved.
credit_card_holders = data[data['CreditCard'] == 1]
approved_loans_credit_card = credit_card_holders[credit_card_holders['Personal_Loan'] == 1]
print(f'Count of people with credit card: {len(credit_card_holders)}')

percentage = (len(approved_loans_credit_card) / len(credit_card_holders)) * 100
print(f"Percentage of people with a credit card whose personal loan was approved: {percentage:.2f}%")

#CCAvg, , CD_Account, Income, Mortgage are the variables that we probably should care about...
```

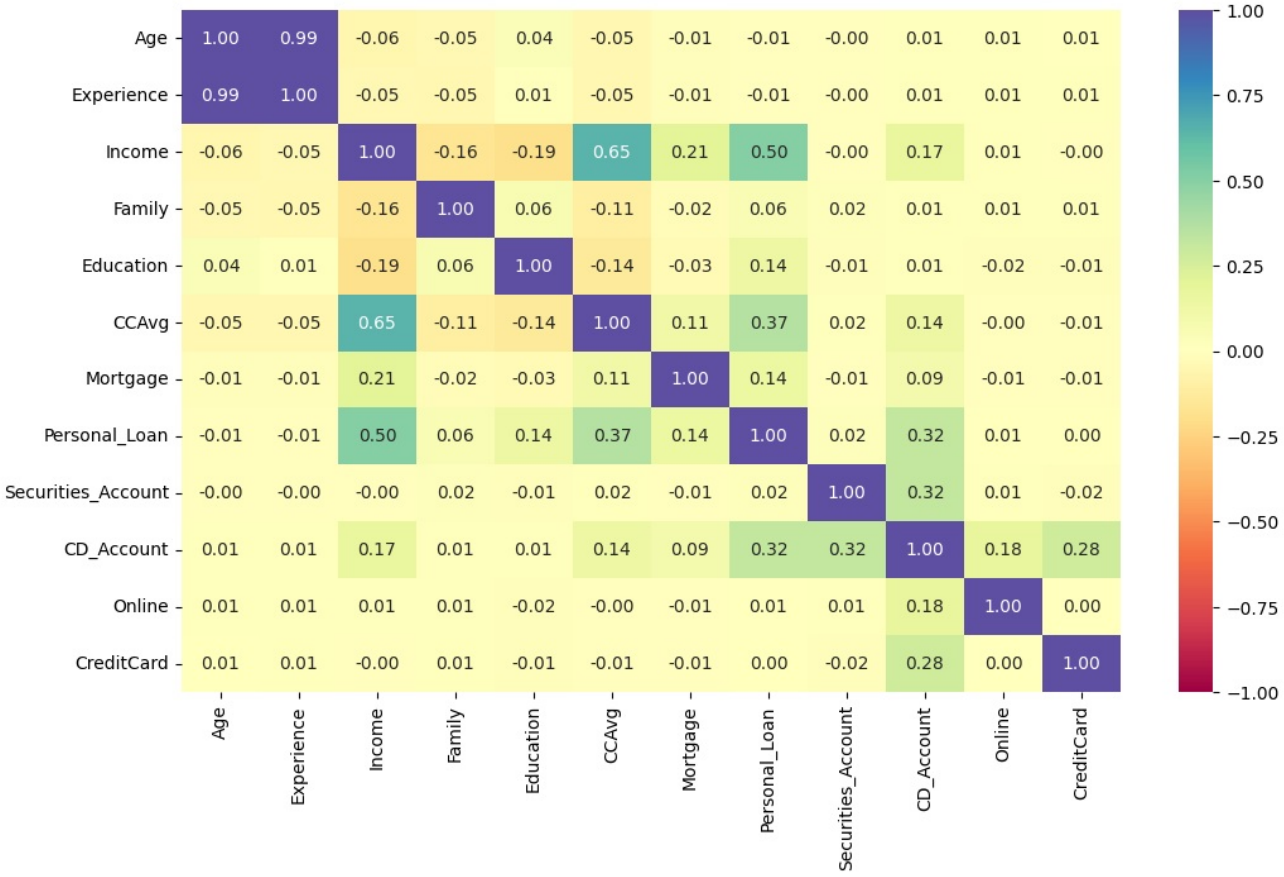
```
Count of people with credit card: 1470
Percentage of people with a credit card whose personal loan was approved: 9.73%
```

Observation: Amongst the approved folks, a mere 10% had a credit card or had an online presense. So, I dont find anything here to target them.

Multivariate Analysis

In [16]:

```
# defining the size of the plot
plt.figure(figsize=(12, 7))

# plotting the heatmap for correlation
sns.heatmap(
    data[num_features].corr(),annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
);
```



Observations: The important features from the bivariable analysis are:

1) CD Account, Income and CCAvg primarily have a stronger co relation to personal loan.

2) Mortage and Education seem to be relevant here also. The coorelation is positive.

3) Family is also positively corelated.

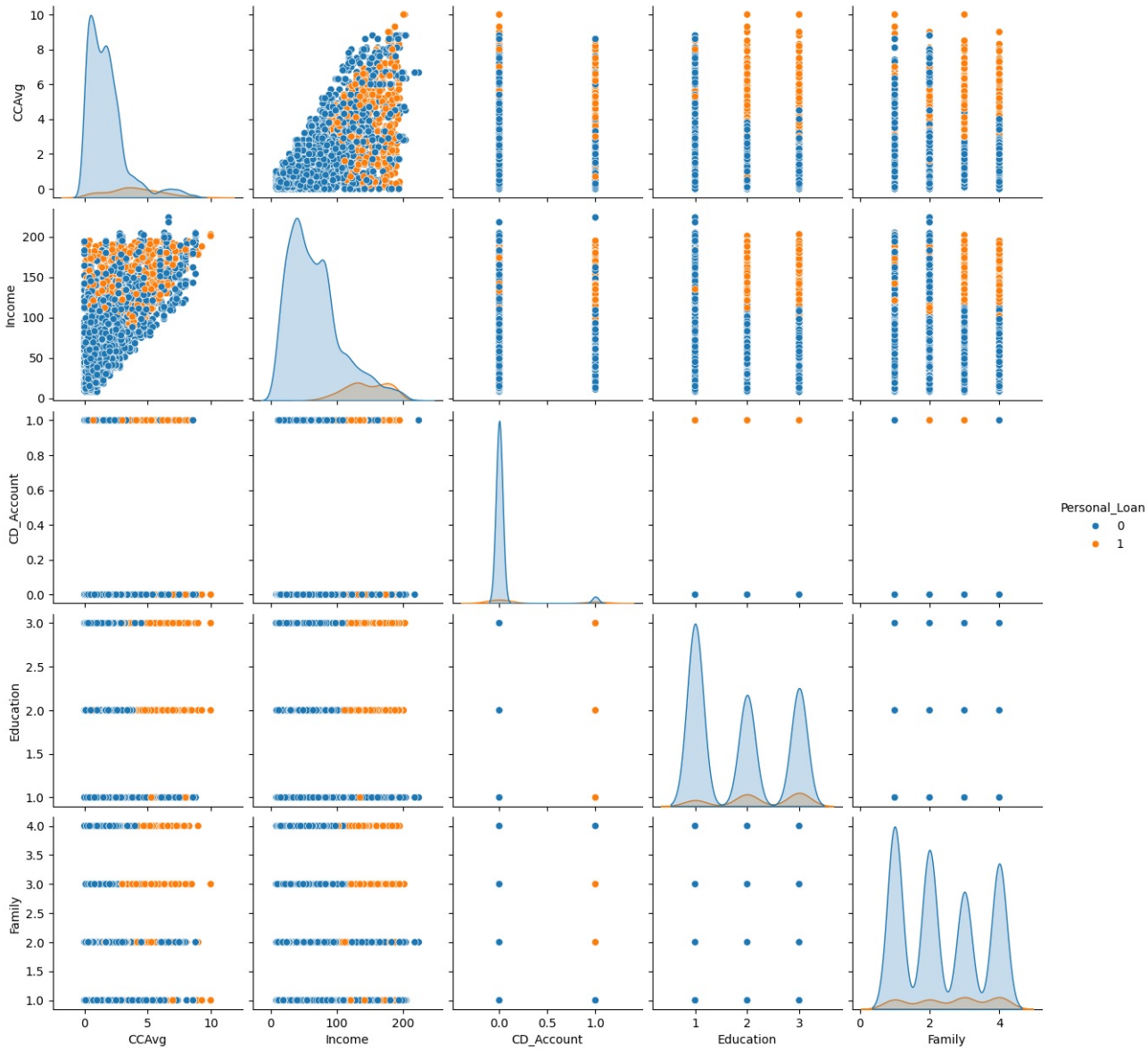
4) Age and experience are highly corelated to each other, but not to personal loan.

5) Credit Card, Online and security Account have a low co relation to personal loan approval(s).

In [17]:

```
# Scatter plot matrix.
# the new features is a refined version of the features after the heat map and EDA.
num_new_features = ['CCAvg', 'Income', 'CD_Account', 'Education', 'Family']

plt.figure(figsize=(12, 8))
sns.pairplot(data, vars=num_new_features, hue='Personal_Loan', diag_kind='kde');
```

```
<Figure size 1200x800 with 0 Axes>
```



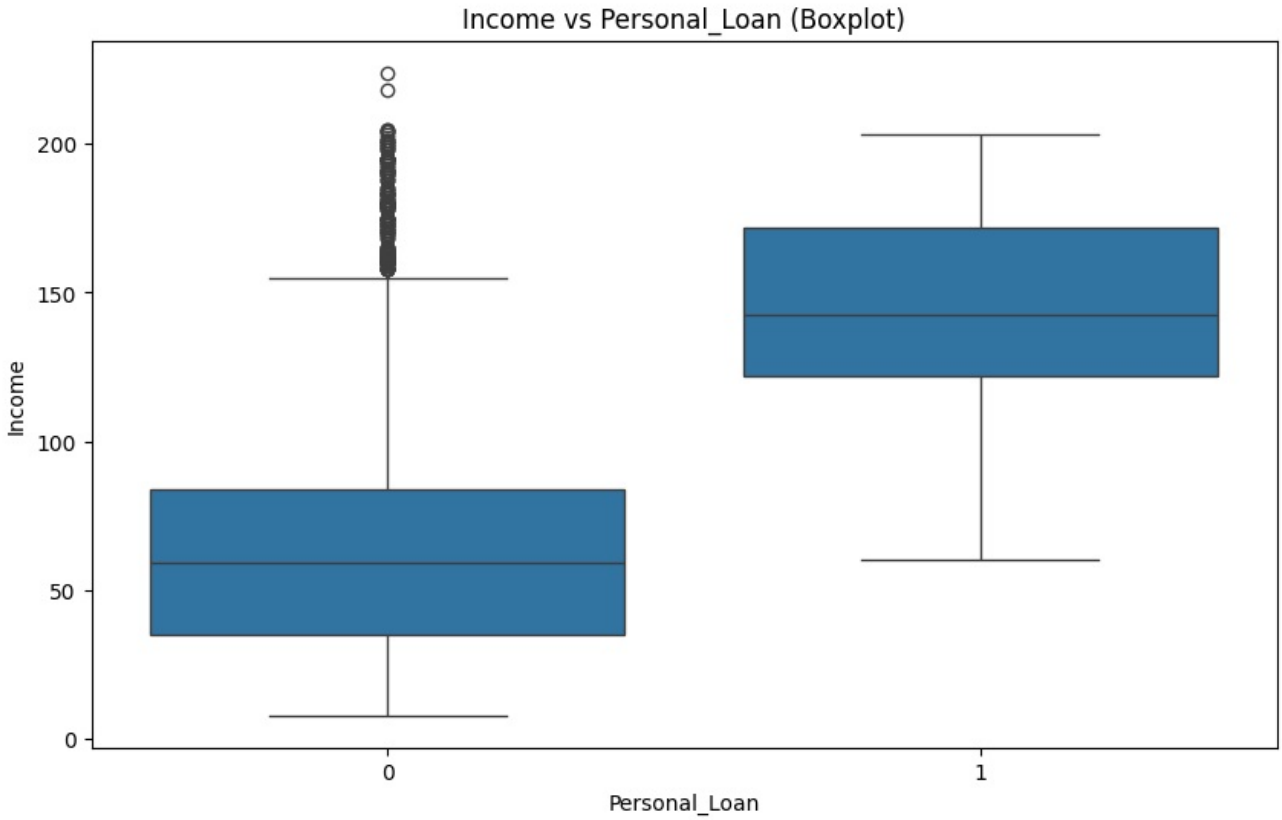
Below are the observations that I have:

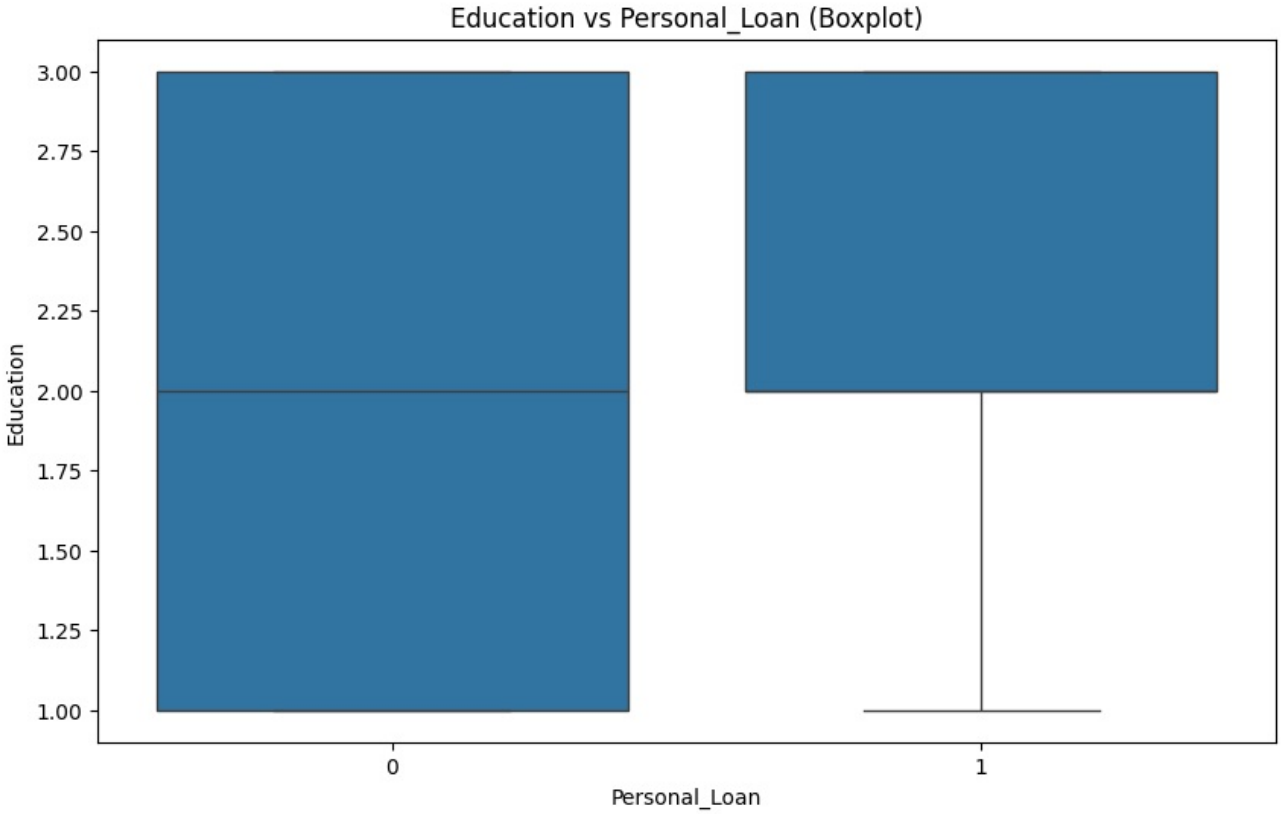
1. People with higher income have a higher chance of their personal loan being approved.
2. Lot of the prior approvals have a online CD account. So this is going to be a factor in future approvals too.
3. Lot of the people in the higher income also have a higher education. So it looks like that is going to play as a factor also.

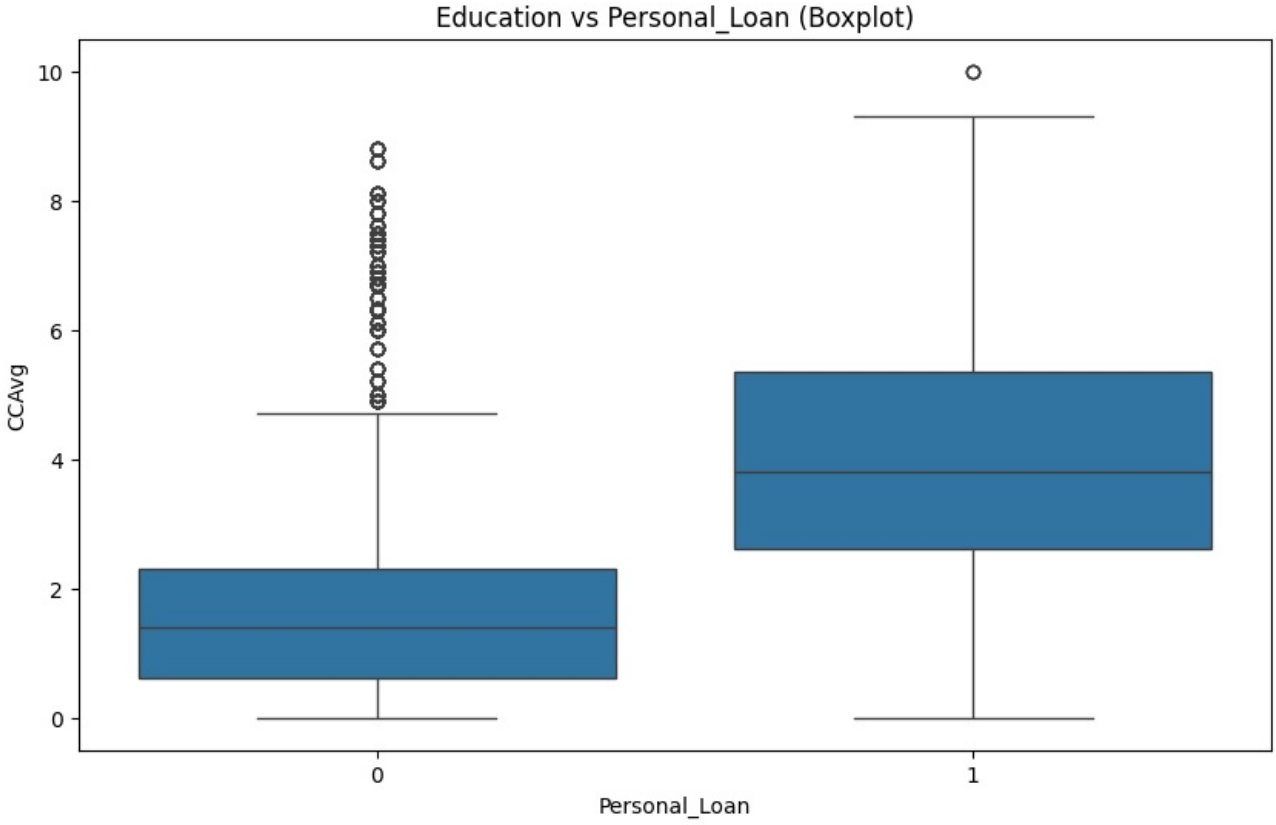
Therefore the features that are of prime importance looks to be a) Income, b) CCAvg and c) Education.

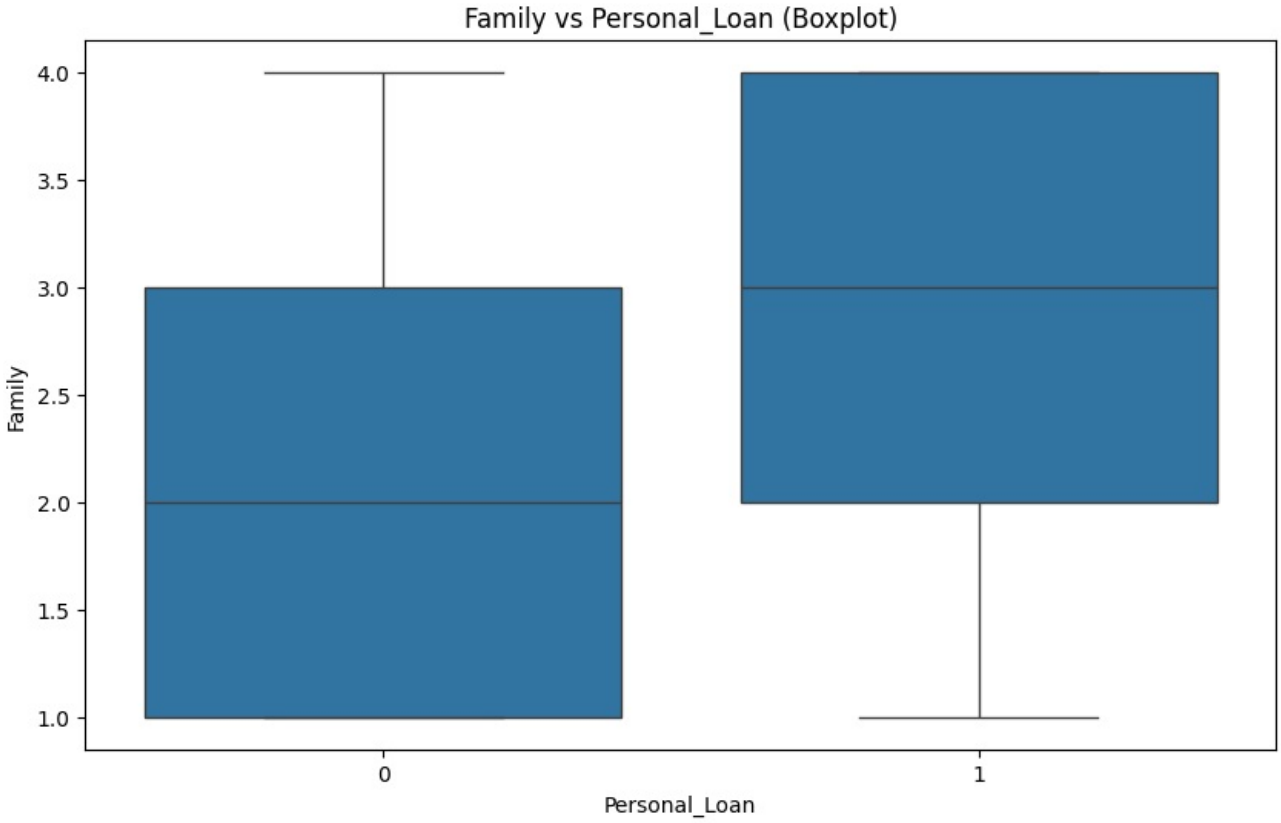
In [18]:

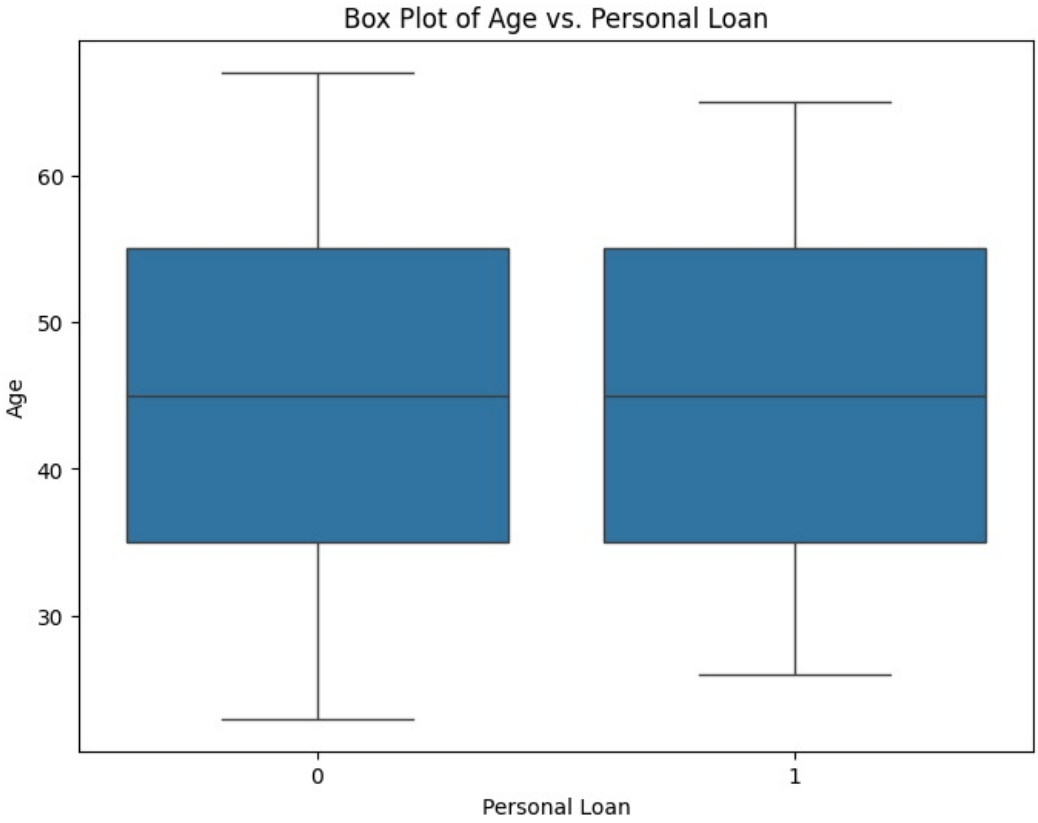
```
# Income vs Approved (boxplot)
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Personal_Loan', y='Income')
plt.title('Income vs Personal_Loan (Boxplot)')
plt.show()

# Education vs Approved (boxplot)
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Personal_Loan', y='Education')
plt.title('Education vs Personal_Loan (Boxplot)')
plt.show()

# CCAvg vs Approved (boxplot)
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Personal_Loan', y='CCAvg')
plt.title('Education vs Personal_Loan (Boxplot)')
plt.show()

# Family vs Approved (boxplot)
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Personal_Loan', y='Family')
plt.title('Family vs Personal_Loan (Boxplot)')
plt.show()

#Box Plot of Age vs. Personal Loan
plt.figure(figsize=(8, 6))
sns.boxplot(x='Personal_Loan', y='Age', data=data)
plt.title('Box Plot of Age vs. Personal Loan')
plt.xlabel('Personal Loan')
plt.ylabel('Age')
plt.show()
```




Education vs Personal_Loan (Boxplot)
Education
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
1
Personal_Loan


Education vs Personal_Loan (Boxplot)
CCAvg
10
8
6
4
2
0
0
1
Personal_Loan





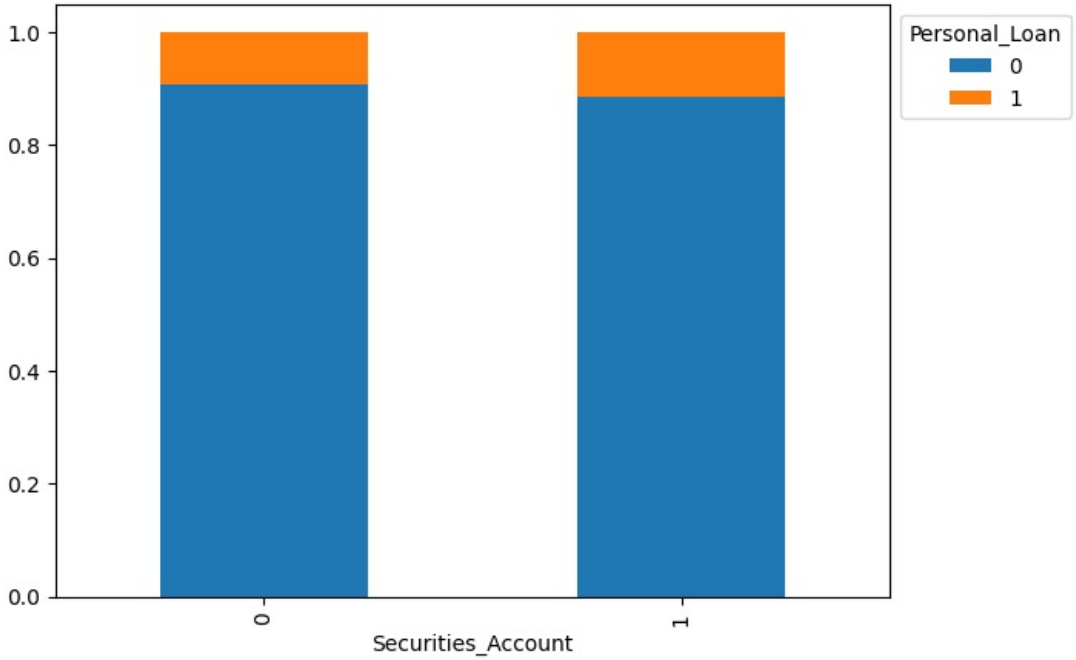
Observation:

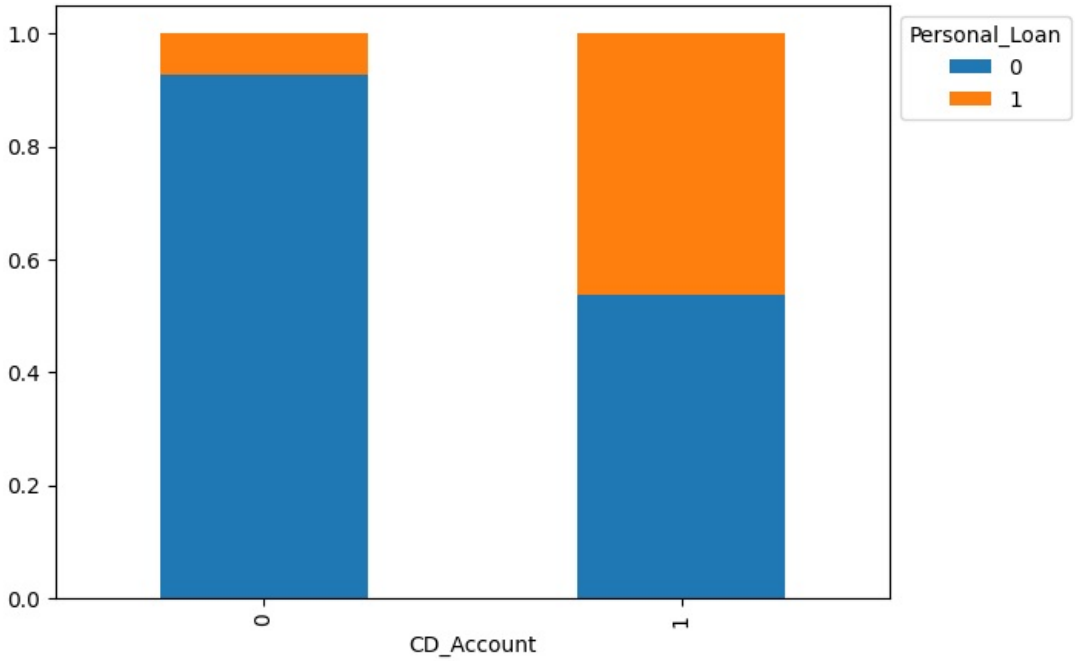
1. EDUCATION: Just as I mentioned on the previous anaysis, people with at least education at a gradudate level or advanced level seem to be good candidates for approval.
2. INCOME: Also, people with higher income also qualify to be probable candidates.
3. FAMILY: The approval is also dependant on fanly size. For single people the chance of approval is low.
4. CCAVG: The box plot clearly shows prior approvals for people with higher CCAvg.

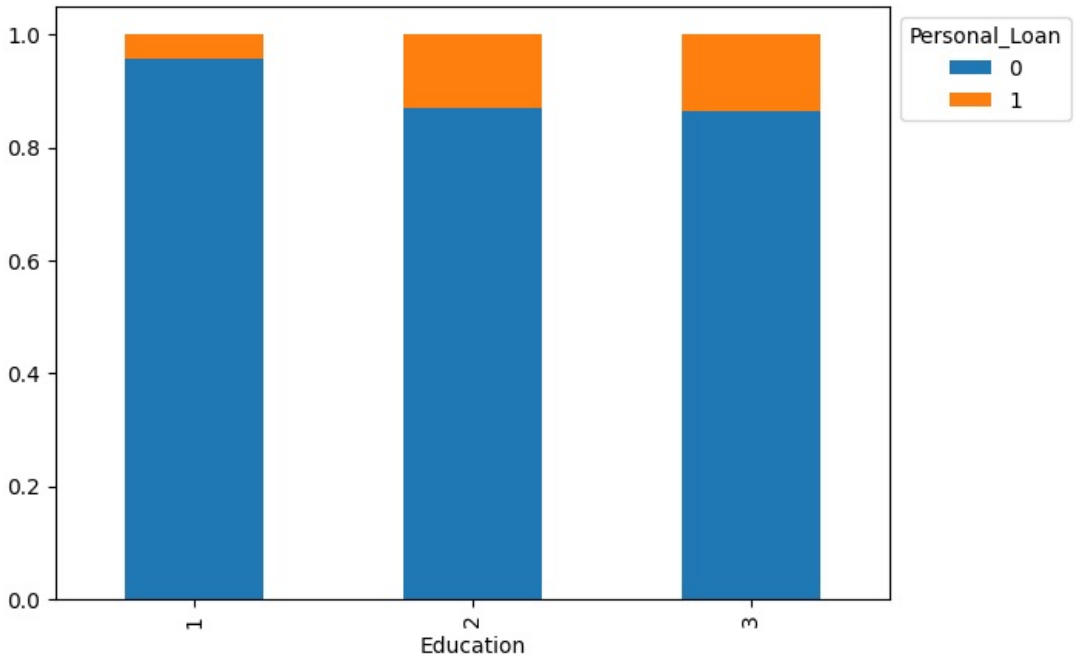
Mortgage, Experience and Age dont seem to be factor in the decision making. (Why? - For prior loan approvals we can see in the box plots that the age is exactly the same for approved and not approved, indicating that age is not a factor. Now, since Experience is very corelated to age, it would be the same for experience also. Mortgage is 0 for the vast amount of people, so it was not used for prior approvals)

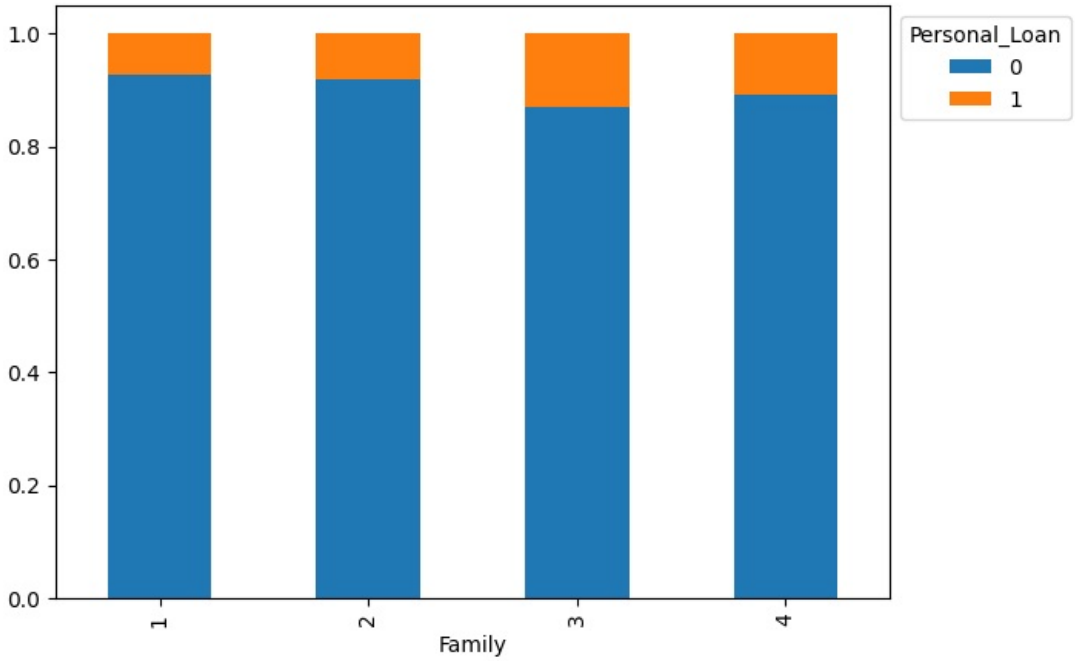
In [19]:

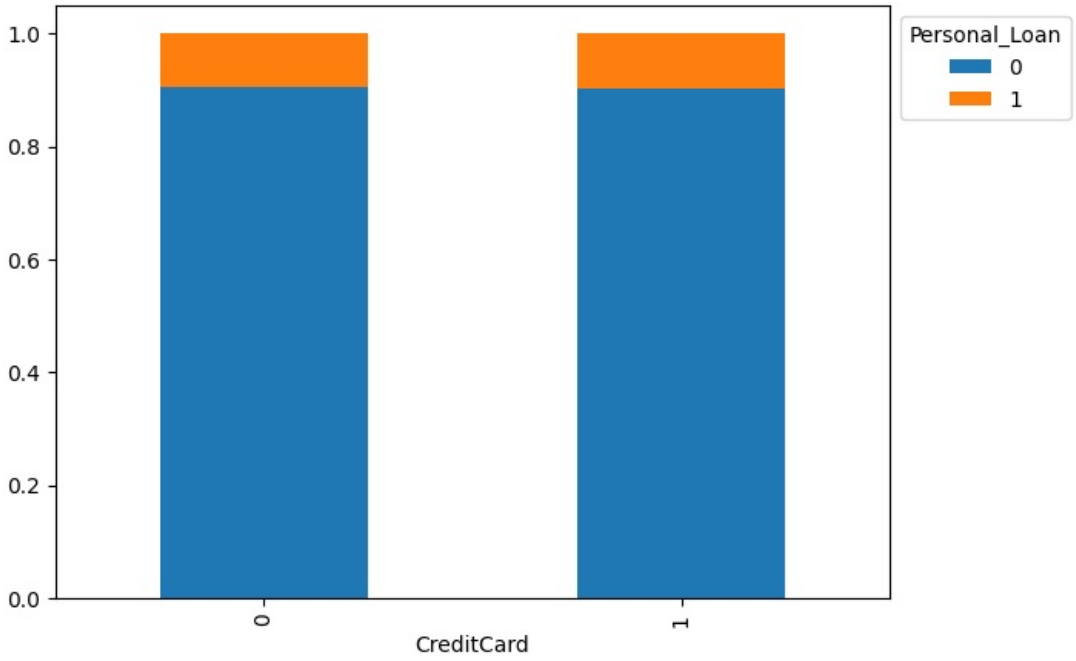
```
# creating a crosstab (come back!!!)
num_features = ['Securities_Account', 'CD_Account', 'Education', 'Family', 'CreditCard']

for feature in num_features:
  tab = pd.crosstab(
    data[feature],
    data['Personal_Loan'],
    normalize='index'    # normalizing by dividing each row by its row total
  )
    # Plot the stacked bar chart
  tab.plot(kind='bar', stacked=True, figsize=(7, 5))    # creating a stacked bar chart from the normalized crosst
ab
  plt.xlabel(feature)
  plt.legend(loc='upper left', bbox_to_anchor=(1, 1), title='Personal_Loan');    # adding a legend for the 'Appro
val' column
```






Personal_Loan
0
1
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
Education


Personal_Loan
0
1
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
Family


Personal_Loan
0
1
1.0
0.8
0.6
0.4
0.2
0.0
0
1
CreditCard

Observations for the cross tabs (above):

1. Prior approvals for security account and the ones that awere not approved seem to tbe visually the same.
2. Prior approvals for family seem to be slightly higher for non-single people
3. Approvals seem to be better for people with CD Account
4. Approvals seem to be higher for people with higher education.

Observations on the other questions:

1. What is the distribution of mortgage attribute? Are there any noticeable patterns or outliers in the distribution?

Answer: Mortgage value is 0 for a significant part of the dataset. Higher mortage indicates higher income and hence the likeliness of being approved.

1. How many customers have credit cards?
   Answer : 1470
2. What are the attributes that have a strong correlation with the target attribute (personal loan)?
   Answer: Income, CD_Account, CCAvg seem to have a strong corelation. Mortgage, Family and education have a positive corelation, but not strong.
3. How does a customer's interest in purchasing a loan vary with their age?
   Answer: The interest in purchasing a loan doesnt seem to vary with age. Per the box plot (age vs Personal_Loan), they seem to be the exact for approved vs not approved.
4. How does a customer's interest in purchasing a loan vary with their education?
   Answer: The prior approvals for a personal loan seem to be more probable for people with at least a graduate degree. (Cross tabs above have only 4.43% of undergraduates with loan approved)

Prepare the data for modelling

In [20]:

```
#Lets start preparing the data

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

# splitting the data in an 80:20 ratio for train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_state=42)
# stratify ensures that the training and test sets have a similar distribution of the response variable


print("Shape of training set:", X_train.shape)
print("Shape of test set:", X_test.shape, '\n')
print("Percentage of classes in training set:")
print(100*y_train.value_counts(normalize=True), '\n')
print("Percentage of classes in test set:")
print(100*y_test.value_counts(normalize=True))
```

```
Shape of training set: (4000, 13)
Shape of test set: (1000, 13)

Percentage of classes in training set:
Personal_Loan
0    90.4
1     9.6
Name: proportion, dtype: float64

Percentage of classes in test set:
Personal_Loan
0    90.4
1     9.6
Name: proportion, dtype: float64
```

Build the model

In [21]:

```
# creating an instance of the decision tree model
dtree1 = DecisionTreeClassifier(random_state=42)    # random_state sets a seed value and enables reproducibility

# fitting the model to the training data
dtree1.fit(X_train, y_train)
```

Out[21]:

```
DecisionTreeClassifier(random_state=42)
```

Model Evaluation

In [22]:

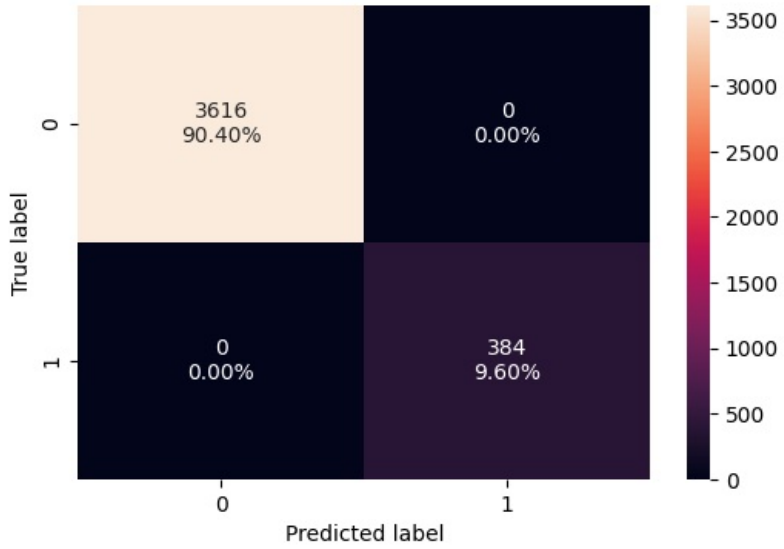
```
# defining a function to compute different metrics to check performance of a classification model built using skl
earn
def model_performance_classification(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    return df_perf
```

In [23]:

```
def plot_confusion_matrix(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    # Predict the target values using the provided model and predictors
    y_pred = model.predict(predictors)

    # Compute the confusion matrix comparing the true target values with the predicted values
    cm = confusion_matrix(target, y_pred)

    # Create labels for each cell in the confusion matrix with both count and percentage
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)    # reshaping to a matrix

    # Set the figure size for the plot
    plt.figure(figsize=(6, 4))

    # Plot the confusion matrix as a heatmap with the labels
    sns.heatmap(cm, annot=labels, fmt="")

    # Add a label to the y-axis
    plt.ylabel("True label")

    # Add a label to the x-axis
    plt.xlabel("Predicted label")
```

In [24]:

```
plot_confusion_matrix(dtree1, X_train, y_train)
```



In [25]:

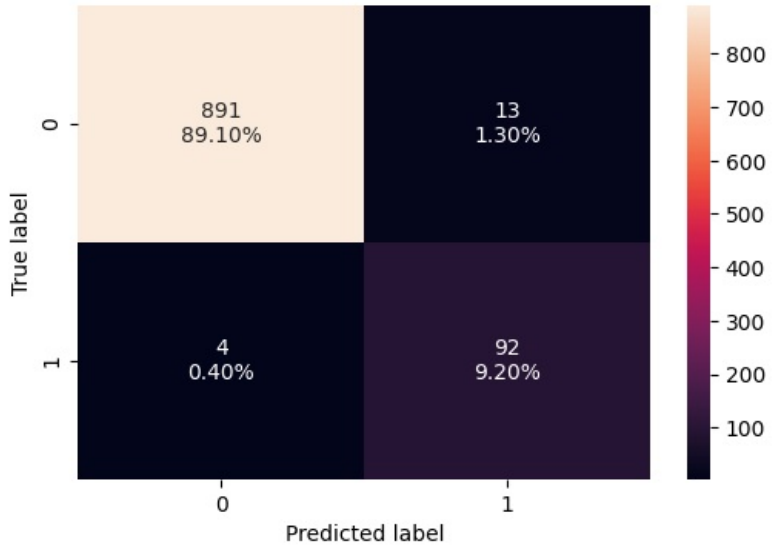
```
dtree1_train_perf = model_performance_classification(
    dtree1, X_train, y_train
)
dtree1_train_perf
```

Out[25]:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

In [26]:

```
plot_confusion_matrix(dtree1, X_test, y_test)
```



In [27]:

```
dtree1_test_perf = model_performance_classification(
    dtree1, X_test, y_test
)
dtree1_test_perf
```

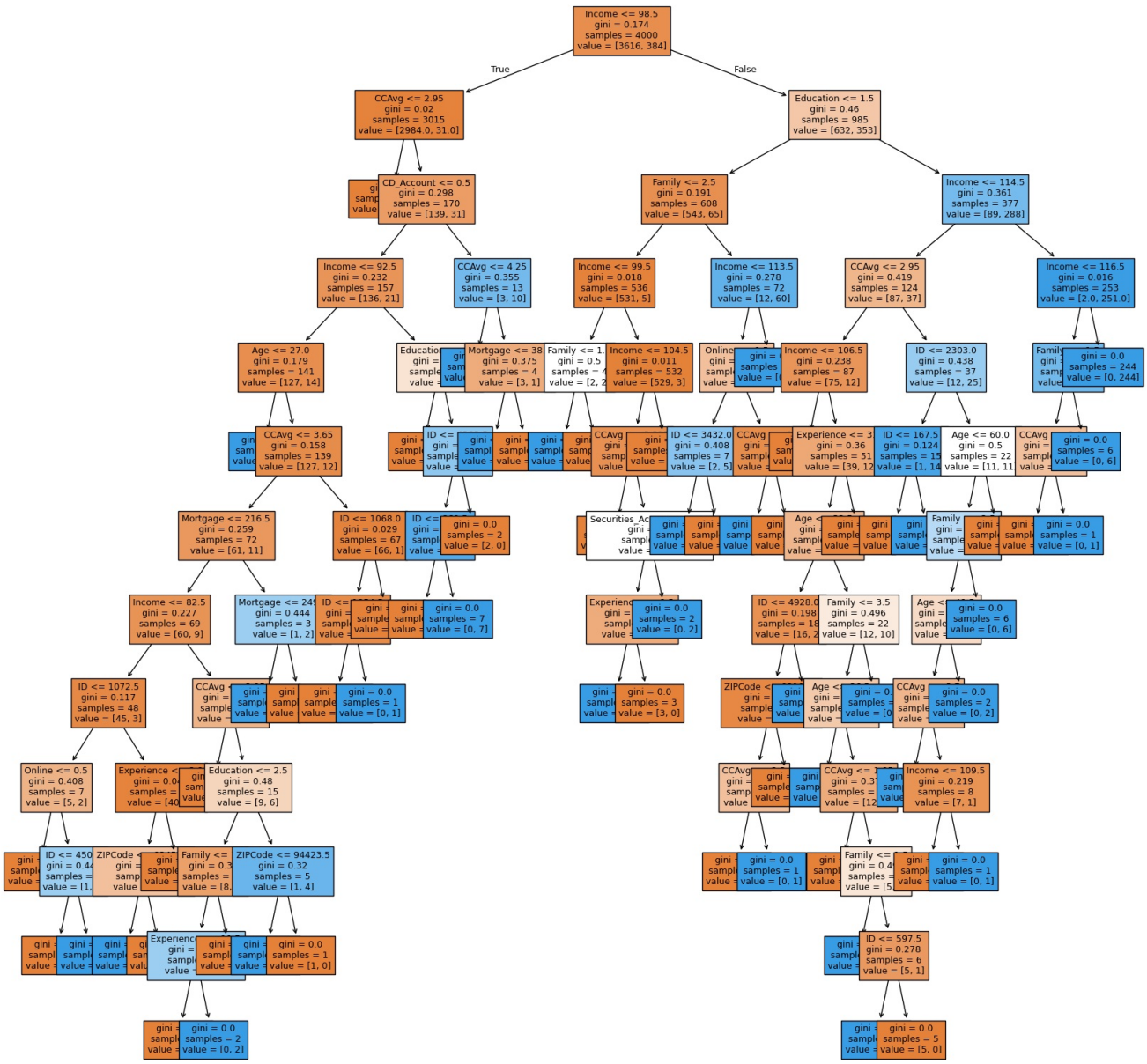
Out[27]:

	Accuracy	Recall	Precision	F1
0	0.983	0.958333	0.87619	0.915423

Observations: The training set is perfect, but the test test is lower. The precision is 0.87. This clearly means that the model is overfitting. This is not good. Now, lets visualize the decision tree

In [28]:

```
# list of feature names in X_train
feature_names = list(X_train.columns)

# set the figure size for the plot
plt.figure(figsize=(20, 20))

# plotting the decision tree
out = tree.plot_tree(
    dtree1,                         # decision tree classifier model
    feature_names=feature_names,    # list of feature names (columns) in the dataset
    filled=True,                    # fill the nodes with colors based on class
    fontsize=9,                     # font size for the node text
    node_ids=False,                 # do not show the ID of each node
    class_names=None,               # whether or not to display class names
)

# add arrows to the decision tree splits if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")    # set arrow color to black
        arrow.set_linewidth(1)          # set arrow linewidth to 1

# displaying the plot
plt.show()
```



In [29]:

```
# printing a text report showing the rules of a decision tree
print(
    tree.export_text(
        dtree1,    # specify the model
        feature_names=feature_names,    # specify the feature names
        show_weights=True    # specify whether or not to show the weights associated with the model
    )
)
```

```
|--- Income <= 98.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [2845.00, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- CD_Account <= 0.50
|   |   |   |--- Income <= 92.50
|   |   |   |   |--- Age <= 27.00
|   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |--- Age >  27.00
|   |   |   |   |   |--- CCAvg <= 3.65
|   |   |   |   |   |   |--- Mortgage <= 216.50
|   |   |   |   |   |   |   |--- Income <= 82.50
|   |   |   |   |   |   |   |   |--- ID <= 1072.50
|   |   |   |   |   |   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Online >  0.50
|   |   |   |   |   |   |   |   |   |   |--- ID <= 450.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- ID >  450.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- ID >  1072.50
|   |   |   |   |   |   |   |   |   |--- Experience <= 8.00
|   |   |   |   |   |   |   |   |   |   |--- ZIPCode <= 92453.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- ZIPCode >  92453.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Experience >  8.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [38.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Income >  82.50
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |   |   |   |   |--- Education <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- Family <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |--- Family >  1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Education >  2.50
|   |   |   |   |   |   |   |   |--- ZIPCode <= 94423.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |   |   |   |--- ZIPCode >  94423.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- Mortgage >  216.50
|   |   |   |   |   |   |   |--- Mortgage <= 249.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- Mortgage >  249.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  3.65
|   |   |   |   |   |   |--- ID <= 1068.00
|   |   |   |   |   |   |   |--- ID <= 1054.50
|   |   |   |   |   |   |   |   |--- weights: [13.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- ID >  1054.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- ID >  1068.00
|   |   |   |   |   |   |   |--- weights: [53.00, 0.00] class: 0
|   |   |   |--- Income >  92.50
|   |   |   |   |--- Education <= 1.50
|   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |   |--- Education >  1.50
|   |   |   |   |   |--- ID <= 4282.50
|   |   |   |   |   |   |--- ID <= 860.50
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- ID >  860.50
|   |   |   |   |   |   |   |--- weights: [0.00, 7.00] class: 1
|   |   |   |   |   |--- ID >  4282.50
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |--- CD_Account >  0.50
|   |   |   |--- CCAvg <= 4.25
|   |   |   |   |--- weights: [0.00, 9.00] class: 1
|   |   |   |--- CCAvg >  4.25
|   |   |   |   |--- Mortgage <= 38.00
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
```

```
|   |   |   |   |--- Mortgage >  38.00
|   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|--- Income >  98.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- Income <= 99.50
|   |   |   |   |--- Family <= 1.50
|   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |--- Family >  1.50
|   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |--- Income >  99.50
|   |   |   |   |--- Income <= 104.50
|   |   |   |   |   |--- CCAvg <= 3.31
|   |   |   |   |   |   |--- weights: [20.00, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  3.31
|   |   |   |   |   |   |--- Securities_Account <= 0.50
|   |   |   |   |   |   |   |--- Experience <= 8.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- Experience >  8.50
|   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |--- Securities_Account >  0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |--- Income >  104.50
|   |   |   |   |   |--- weights: [506.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- Income <= 113.50
|   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |--- ID <= 3432.00
|   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |--- ID >  3432.00
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |--- Online >  0.50
|   |   |   |   |   |--- CCAvg <= 0.65
|   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- CCAvg >  0.65
|   |   |   |   |   |   |--- weights: [10.00, 0.00] class: 0
|   |   |   |--- Income >  113.50
|   |   |   |   |--- weights: [0.00, 54.00] class: 1
|   |--- Education >  1.50
|   |   |--- Income <= 114.50
|   |   |   |--- CCAvg <= 2.95
|   |   |   |   |--- Income <= 106.50
|   |   |   |   |   |--- weights: [36.00, 0.00] class: 0
|   |   |   |   |--- Income >  106.50
|   |   |   |   |   |--- Experience <= 31.50
|   |   |   |   |   |   |--- Age <= 33.50
|   |   |   |   |   |   |   |--- ID <= 4928.00
|   |   |   |   |   |   |   |   |--- ZIPCode <= 92109.50
|   |   |   |   |   |   |   |   |   |--- CCAvg <= 2.30
|   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- CCAvg >  2.30
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- ZIPCode >  92109.50
|   |   |   |   |   |   |   |   |   |--- weights: [14.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- ID >  4928.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- Age >  33.50
|   |   |   |   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |   |   |   |--- Age <= 36.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |--- Age >  36.50
|   |   |   |   |   |   |   |   |   |--- CCAvg <= 1.05
|   |   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- CCAvg >  1.05
|   |   |   |   |   |   |   |   |   |   |--- Family <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- Family >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |--- Family >  3.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |--- Experience >  31.50
|   |   |   |   |   |   |--- weights: [11.00, 0.00] class: 0
|   |   |   |--- CCAvg >  2.95
|   |   |   |   |--- ID <= 2303.00
|   |   |   |   |   |--- ID <= 167.50
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- ID >  167.50
|   |   |   |   |   |   |--- weights: [0.00, 14.00] class: 1
|   |   |   |   |--- ID >  2303.00
|   |   |   |   |   |--- Age <= 60.00
|   |   |   |   |   |   |--- Family <= 2.50
|   |   |   |   |   |   |   |--- Age <= 49.50
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.70
```

```
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- CCAvg >  3.70
|   |   |   |   |   |   |   |   |   |--- Income <= 109.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- Income >  109.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- Age >  49.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- Family >  2.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 6.00] class: 1
|   |   |   |   |   |   |--- Age >  60.00
|   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |--- Income >  114.50
|   |   |   |   |--- Income <= 116.50
|   |   |   |   |   |--- Family <= 1.50
|   |   |   |   |   |   |--- CCAvg <= 1.10
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  1.10
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- Family >  1.50
|   |   |   |   |   |   |--- weights: [0.00, 6.00] class: 1
|   |   |   |   |--- Income >  116.50
|   |   |   |   |   |--- weights: [0.00, 244.00] class: 1
```

Decision Tree (Pre-pruning)

In [30]:

```
# define the parameters of the tree to iterate over
max_depth_values = np.arange(2, 11, 2)
max_leaf_nodes_values = np.arange(10, 51, 10)
min_samples_split_values = np.arange(10, 51, 10)

# initialize variables to store the best model and its performance
best_estimator = None
best_score_diff = float('inf')

# iterate over all combinations of the specified parameter values
for max_depth in max_depth_values:
    for max_leaf_nodes in max_leaf_nodes_values:
        for min_samples_split in min_samples_split_values:

            # initialize the tree with the current set of parameters
            estimator = DecisionTreeClassifier(
                max_depth=max_depth,
                max_leaf_nodes=max_leaf_nodes,
                min_samples_split=min_samples_split,
                random_state=42
            )

            # fit the model to the training data
            estimator.fit(X_train, y_train)

            # make predictions on the training and test sets
            y_train_pred = estimator.predict(X_train)
            y_test_pred = estimator.predict(X_test)

            # calculate F1 scores for training and test sets
            train_f1_score = f1_score(y_train, y_train_pred)
            test_f1_score = f1_score(y_test, y_test_pred)

            # calculate the absolute difference between training and test F1 scores
            score_diff = abs(train_f1_score - test_f1_score)

            # update the best estimator and best score if the current one has a smaller score difference
            if score_diff < best_score_diff:
                best_score_diff = score_diff
                best_estimator = estimator
```

In [31]:

```
# creating an instance of the best model
dtree2 = best_estimator

# fitting the best model to the training data
dtree2.fit(X_train, y_train)
```

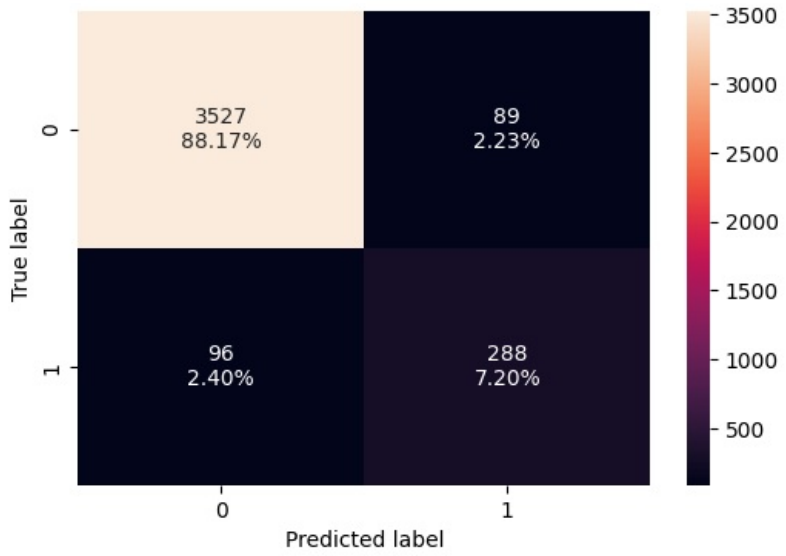
Out[31]:

```
DecisionTreeClassifier(max_depth=np.int64(2), max_leaf_nodes=np.int64(10),
                       min_samples_split=np.int64(10), random_state=42)
```

Lets evaluate the pruned Model

In [32]:

```
plot_confusion_matrix(dtree2, X_train, y_train)
```



In [33]:

```
dtree2_train_perf = model_performance_classification(
    dtree2, X_train, y_train
)
dtree2_train_perf
```

Out[33]:

	Accuracy	Recall	Precision	F1
0	0.95375	0.75	0.763926	0.756899

In [34]:

```
dtree2_test_perf = model_performance_classification(
    dtree2, X_test, y_test
)
dtree2_test_perf
```

Out[34]:

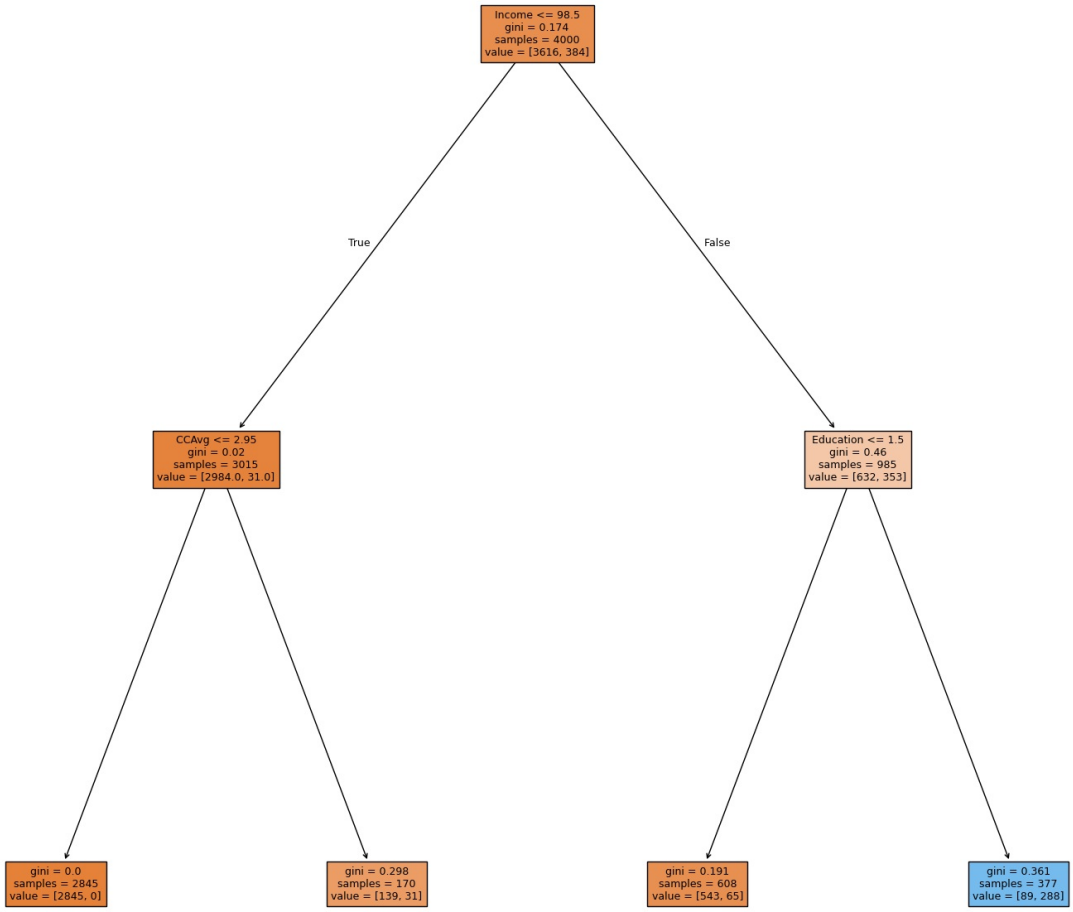
	Accuracy	Recall	Precision	F1
0	0.949	0.78125	0.714286	0.746269

Observation: The training and the test scores are very close to each other, indicating a better fit. Lets take the F1 scores since they are a mean.

Lets Visualize the tree after pre pruning

In [35]:

```
feature_names = list(X_train.columns)

# set the figure size for the plot
plt.figure(figsize=(20, 20))

# plotting the decision tree
out = tree.plot_tree(
    dtree2,                         # decision tree classifier model
    feature_names=feature_names,    # list of feature names (columns) in the dataset
    filled=True,                    # fill the nodes with colors based on class
    fontsize=9,                     # font size for the node text
    node_ids=False,                 # do not show the ID of each node
    class_names=None,               # whether or not to display class names
)

# add arrows to the decision tree splits if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")    # set arrow color to black
        arrow.set_linewidth(1)          # set arrow linewidth to 1

# displaying the plot
plt.show()
```



Observation: This is a far less complex tree than the previous one. We can observe the decision rules much more clearly in the plot.

In [36]:

```
# printing a text report showing the rules of a decision tree
print(
    tree.export_text(
        dtree2,    # specify the model
        feature_names=feature_names,    # specify the feature names
        show_weights=True    # specify whether or not to show the weights associated with the model
    )
)
```

```
|--- Income <= 98.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [2845.00, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- weights: [139.00, 31.00] class: 0
|--- Income >  98.50
|   |--- Education <= 1.50
|   |   |--- weights: [543.00, 65.00] class: 0
|   |--- Education >  1.50
|   |   |--- weights: [89.00, 288.00] class: 1
```

Decision Tree Post Pruning

In [37]:

```
# Create an instance of the decision tree model
clf = DecisionTreeClassifier(random_state=42)

# Compute the cost complexity pruning path for the model using the training data
path = clf.cost_complexity_pruning_path(X_train, y_train)

# Extract the array of effective alphas from the pruning path
ccp_alphas = abs(path.ccp_alphas)

# Extract the array of total impurities at each alpha along the pruning path
impurities = path.impurities
```

In [38]:

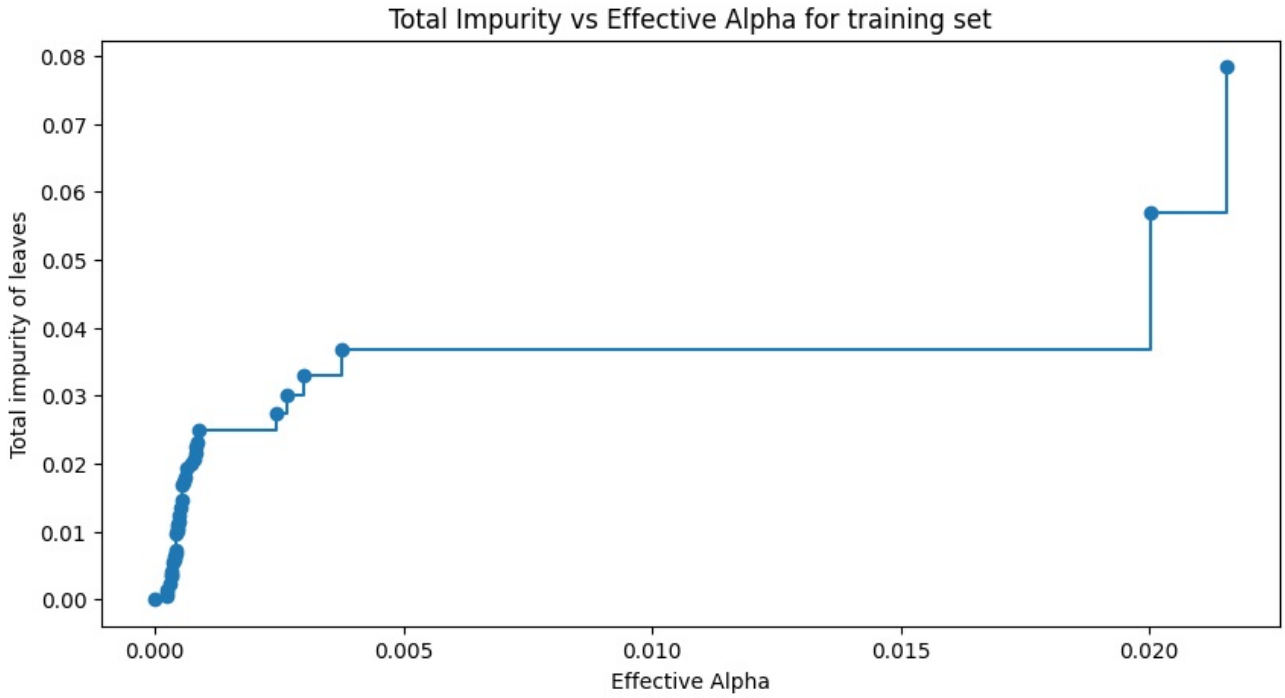
```
pd.DataFrame(path)
```

Out[38]:

	ccp_alphas	impurities
0	0.000000	0.000000
1	0.000235	0.000471
2	0.000244	0.000958
3	0.000246	0.001451
4	0.000306	0.002369
5	0.000331	0.003361
6	0.000333	0.003695
7	0.000333	0.004028
8	0.000373	0.005520
9	0.000375	0.005895
10	0.000400	0.006295
11	0.000417	0.006711
12	0.000418	0.007130
13	0.000419	0.009646
14	0.000438	0.010084
15	0.000438	0.010521
16	0.000455	0.010976
17	0.000467	0.011442
18	0.000493	0.012428
19	0.000510	0.013448
20	0.000542	0.014532
21	0.000550	0.016731
22	0.000584	0.017315
23	0.000613	0.017928
24	0.000646	0.019219
25	0.000714	0.019934
26	0.000779	0.020713
27	0.000823	0.021535
28	0.000831	0.022367
29	0.000837	0.023204
30	0.000870	0.024945
31	0.002424	0.027369
32	0.002667	0.030036
33	0.003000	0.033036
34	0.003753	0.036789
35	0.020023	0.056812
36	0.021549	0.078361
37	0.047604	0.173568

In [39]:

```
#Create a figure
fig, ax = plt.subplots(figsize=(10, 5))

# Plot the total impurities versus effective alphas, excluding the last value,
# using markers at each data point and connecting them with steps
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o", drawstyle="steps-post")

# Set the x-axis label
ax.set_xlabel("Effective Alpha")

# Set the y-axis label
ax.set_ylabel("Total impurity of leaves")

# Set the title of the plot
ax.set_title("Total Impurity vs Effective Alpha for training set");#
```



Next, we train a decision tree using the effective alphas.

The last value in ccp_alphas is the alpha value that prunes the whole tree, leaving the corresponding tree with one node.

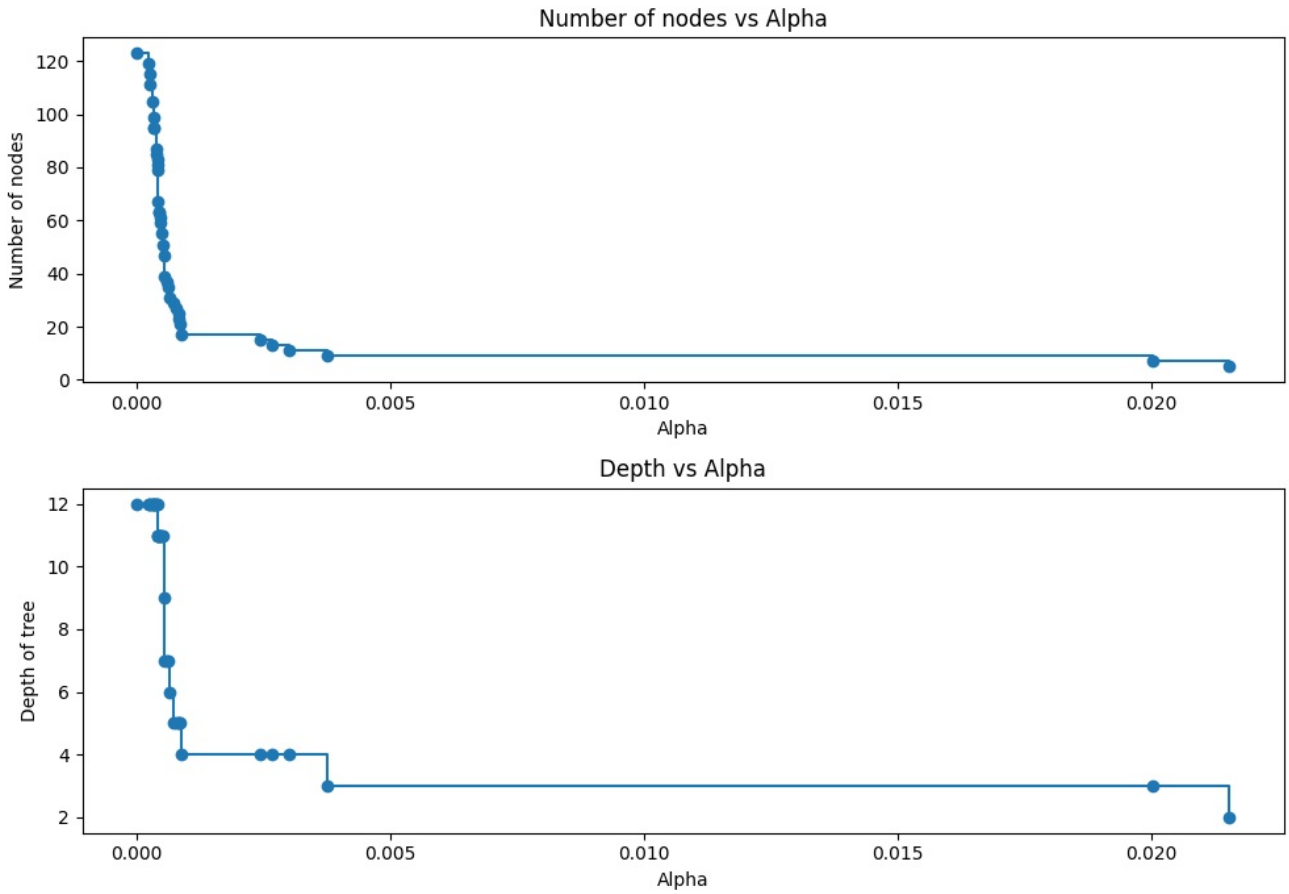
In [40]:

```
# Initialize an empty list to store the decision tree classifiers
clfs = []

# Iterate over each ccp_alpha value extracted from cost complexity pruning path
for ccp_alpha in ccp_alphas:
    # Create an instance of the DecisionTreeClassifier
    clf = DecisionTreeClassifier(ccp_alpha=ccp_alpha, random_state=42)

    # Fit the classifier to the training data
    clf.fit(X_train, y_train)

    # Append the trained classifier to the list
    clfs.append(clf)

# Print the number of nodes in the last tree along with its ccp_alpha value
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 0.04760359071815693
```

In [41]:

```
# Remove the last classifier and corresponding ccp_alpha value from the lists
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

# Extract the number of nodes in each tree classifier
node_counts = [clf.tree_.node_count for clf in clfs]

# Extract the maximum depth of each tree classifier
depth = [clf.tree_.max_depth for clf in clfs]

# Create a figure and a set of subplots
fig, ax = plt.subplots(2, 1, figsize=(10, 7))

# Plot the number of nodes versus ccp_alphas on the first subplot
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")

# Plot the depth of tree versus ccp_alphas on the second subplot
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")

# Adjust the layout of the subplots to avoid overlap
fig.tight_layout()
```



In [42]:

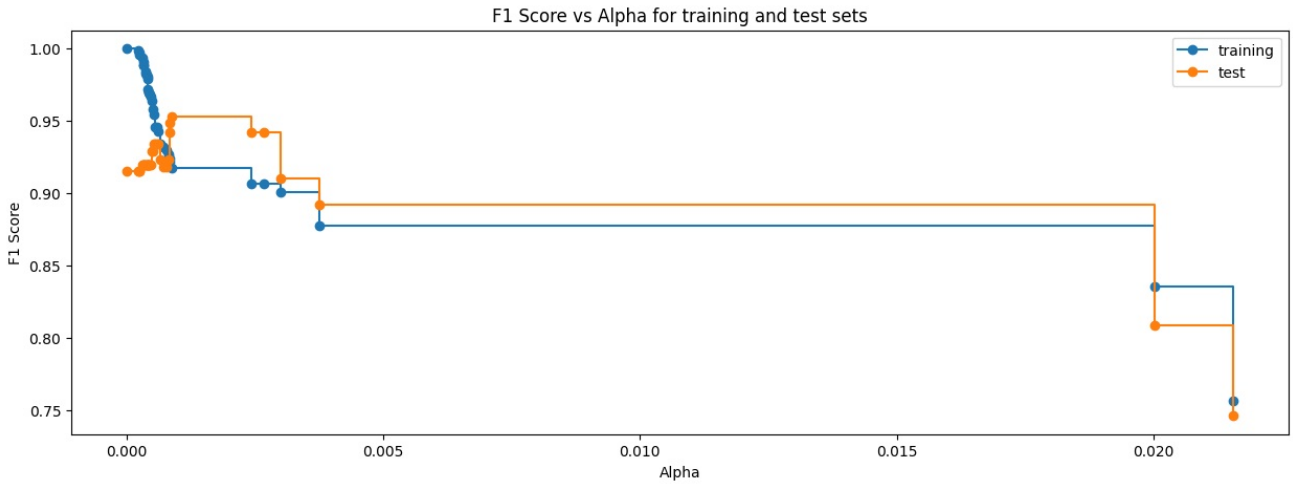
```
train_f1_scores = []  # Initialize an empty list to store F1 scores for training set for each decision tree class
ifier

# Iterate through each decision tree classifier in 'clfs'
for clf in clfs:
    # Predict labels for the training set using the current decision tree classifier
    pred_train = clf.predict(X_train)

    # Calculate the F1 score for the training set predictions compared to true labels
    f1_train = f1_score(y_train, pred_train)

    # Append the calculated F1 score to the train_f1_scores list
    train_f1_scores.append(f1_train)
```

In [43]:

```
test_f1_scores = []  # Initialize an empty list to store F1 scores for test set for each decision tree classifier

# Iterate through each decision tree classifier in 'clfs'
for clf in clfs:
    # Predict labels for the test set using the current decision tree classifier
    pred_test = clf.predict(X_test)

    # Calculate the F1 score for the test set predictions compared to true labels
    f1_test = f1_score(y_test, pred_test)

    # Append the calculated F1 score to the test_f1_scores list
    test_f1_scores.append(f1_test)
```

In [44]:

```
# Create a figure
fig, ax = plt.subplots(figsize=(15, 5))
ax.set_xlabel("Alpha")  # Set the label for the x-axis
ax.set_ylabel("F1 Score")  # Set the label for the y-axis
ax.set_title("F1 Score vs Alpha for training and test sets")  # Set the title of the plot

# Plot the training F1 scores against alpha, using circles as markers and steps-post style
ax.plot(ccp_alphas, train_f1_scores, marker="o", label="training", drawstyle="steps-post")

# Plot the testing F1 scores against alpha, using circles as markers and steps-post style
ax.plot(ccp_alphas, test_f1_scores, marker="o", label="test", drawstyle="steps-post")

ax.legend();  # Add a legend to the plot
```



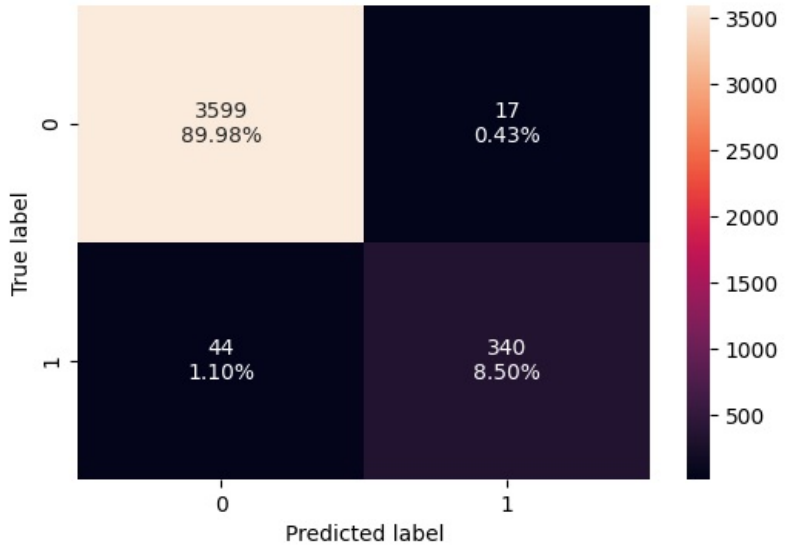
In [45]:

```
#creating the model where we get highest test F1 Score
index_best_model = np.argmax(test_f1_scores)

# selcting the decision tree model corresponding to the highest test score
dtree3 = clfs[index_best_model]
print(dtree3)
```

```
DecisionTreeClassifier(ccp_alpha=np.float64(0.0008702884311333967),
                       random_state=42)
```

In [46]:

```
plot_confusion_matrix(dtree3, X_train, y_train)
```



In [47]:

```
dtree3_train_perf = model_performance_classification(
    dtree3, X_train, y_train
)
dtree3_train_perf
```

Out[47]:

	Accuracy	Recall	Precision	F1
0	0.98475	0.885417	0.952381	0.917679

In [48]:

```
dtree3_test_perf = model_performance_classification(
    dtree3, X_test, y_test
)
dtree3_test_perf
```

Out[48]:

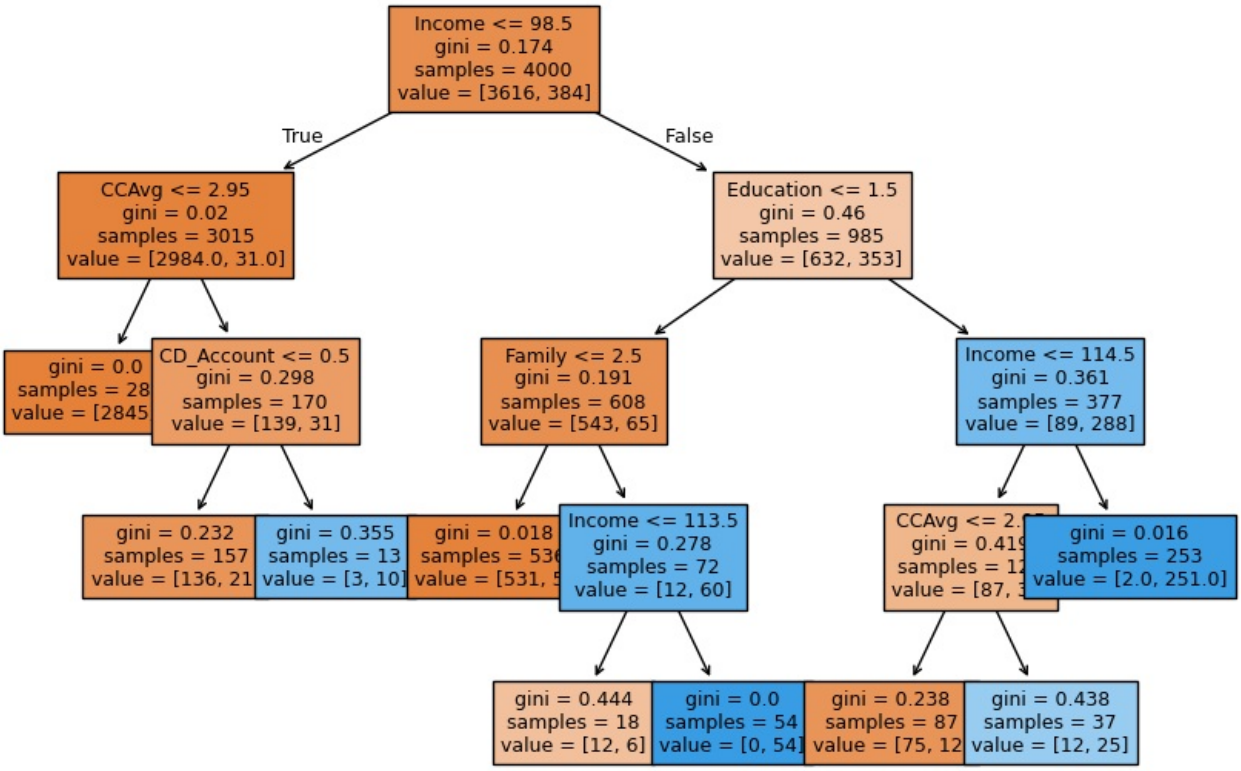
	Accuracy	Recall	Precision	F1
0	0.991	0.958333	0.948454	0.953368

Observations: The test score is greater than the training score, indicating a generalized performance. The F1 scores, and Precision are better in the post prune.

Lets visualize the tree now, post pruning

In [49]:

```
# list of feature names in X_train
feature_names = list(X_train.columns)

# set the figure size for the plot
plt.figure(figsize=(10, 7))

# plotting the decision tree
out = tree.plot_tree(
    dtree3,                         # decision tree classifier model
    feature_names=feature_names,    # list of feature names (columns) in the dataset
    filled=True,                    # fill the nodes with colors based on class
    fontsize=9,                     # font size for the node text
    node_ids=False,                 # do not show the ID of each node
    class_names=None,               # whether or not to display class names
)

# add arrows to the decision tree splits if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")    # set arrow color to black
        arrow.set_linewidth(1)          # set arrow linewidth to 1

# displaying the plot
plt.show()
```



In [50]:

```
# printing a text report showing the rules of a decision tree
print(
    tree.export_text(
        dtree3,    # specify the model
        feature_names=feature_names,    # specify the feature names
        show_weights=True    # specify whether or not to show the weights associated with the model
    )
)
```

```
|--- Income <= 98.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [2845.00, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- CD_Account <= 0.50
|   |   |   |--- weights: [136.00, 21.00] class: 0
|   |   |--- CD_Account >  0.50
|   |   |   |--- weights: [3.00, 10.00] class: 1
|--- Income >  98.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [531.00, 5.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- Income <= 113.50
|   |   |   |   |--- weights: [12.00, 6.00] class: 0
|   |   |   |--- Income >  113.50
|   |   |   |   |--- weights: [0.00, 54.00] class: 1
|   |--- Education >  1.50
|   |   |--- Income <= 114.50
|   |   |   |--- CCAvg <= 2.95
|   |   |   |   |--- weights: [75.00, 12.00] class: 0
|   |   |   |--- CCAvg >  2.95
|   |   |   |   |--- weights: [12.00, 25.00] class: 1
|   |   |--- Income >  114.50
|   |   |   |--- weights: [2.00, 251.00] class: 1
```

Final Model Selection

In [51]:

```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        dtree1_train_perf.T,
        dtree2_train_perf.T,
        dtree3_train_perf.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree (sklearn default)",
    "Decision Tree (Pre-Pruning)",
    "Decision Tree (Post-Pruning)",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[51]:

	Decision Tree (sklearn default)	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	1.0	0.953750	0.984750
Recall	1.0	0.750000	0.885417
Precision	1.0	0.763926	0.952381
F1	1.0	0.756899	0.917679

In [52]:

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [
        dtree1_test_perf.T,
        dtree2_test_perf.T,
        dtree3_test_perf.T,
    ],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree (sklearn default)",
    "Decision Tree (Pre-Pruning)",
    "Decision Tree (Post-Pruning)",
]
print("Test set performance comparison:")
models_test_comp_df
```

```
Test set performance comparison:
```

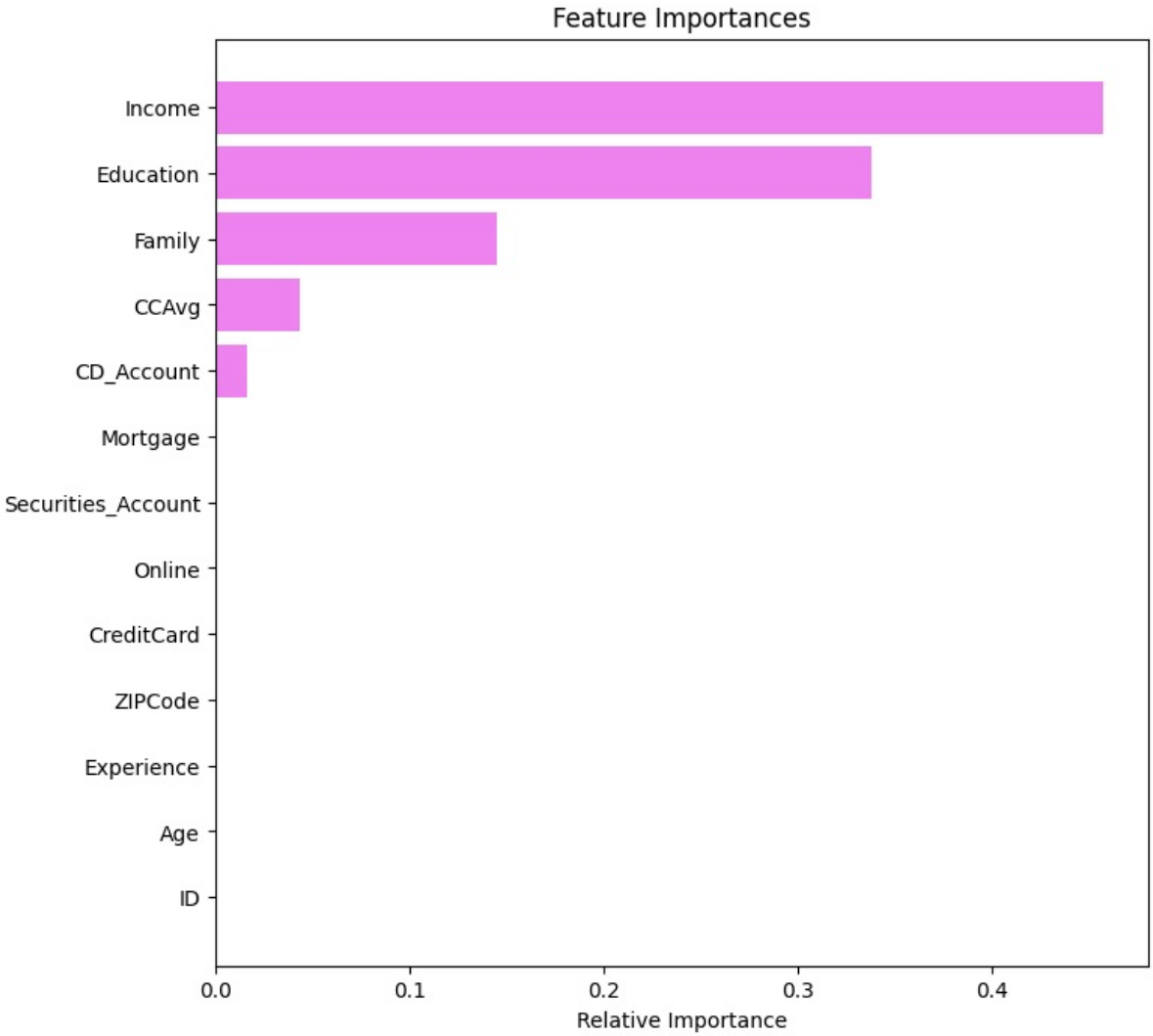
Out[52]:

	Decision Tree (sklearn default)	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.983000	0.949000	0.991000
Recall	0.958333	0.781250	0.958333
Precision	0.876190	0.714286	0.948454
F1	0.915423	0.746269	0.953368

Features used in the ***post pruned tree*** = Income, CCAvg, Education, Family and CD_Account
Features used in the ***Pre pruned tree*** = Icome, CCAvg and Education.
The post pruned tree also generalizes well with better results on the test data than the training data, so we *** WILL CHOOSE THE POST PRUNED TREE***

In [53]:

```
#Lets look at the predictability and the feature set

# importance of features in the tree building
importances = dtree3.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(8, 8))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



In [55]:

```
%%time
# choosing a data point
applicant_details = X_test.iloc[:1, :]

# making a prediction
approval_prediction = dtree3.predict(applicant_details)

print(approval_prediction)
```

```
[0]
CPU times: user 2.64 ms, sys: 37 µs, total: 2.68 ms
Wall time: 2.68 ms
```

The model predicts under 1 second and this is a wonderful performance.

ACTIONABLE INSIGHTS AND BUSINESS RECOMMENDATIONS

- **Education matters:** The prior approvals have at least a graduate degree. The higher the education the better are the chances of approval. So target customers with at least a graduate degree.
- **Family:** The marketing should target customers who are married with children. There is only a small section of approval for customers who are single hence marketing campaign should try and attract married and/or married with children.
- **CD Accounts:** Consider targetting customers who have CD accounts. The prior approvals had a better acceptance rate on people who had CD Accounts.
- **Income:** People with higher income and higher mortage have a better chance of getting approvals. The marketing strategy needs to target them.
- **CC_Avg:** Target marketing toward people with higher CC Average.
- **Constantly run the models and ensure that the strategies are adjusted to fit the latest trends.**