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 [201…

```
# 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==2.2.0 -q --user
```

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

In [202…

```
 # Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)

# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)

# Library to split data
from sklearn.model_selection import train_test_split

# To build model for prediction
```

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

# To tune different models
from sklearn.model_selection import GridSearchCV

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

# Ignore waringings as they do not affect the code
import warnings
warnings.filterwarnings("ignore")

 # Set defaults for seaborn Library
 sns.set()
```

Loading the dataset

In [203…

```
# Define the path to the dataset
file_path = '/home/critter/Learning/data/Loan_Modelling.csv'

# Open and read the data
df = pd.read_csv(file_path)

# Create a copy of the data to avoid changes to the original data
data = df.copy()
```

Data overview

Overview of the dataset

In [204…
```
# Viewing the first five rows of the data
data.head()
```

Out[204…

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

In [205…
```
# Viewing the last five rows of data
data.tail()
```

Out[205…

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Account	CD_Account	O
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	

Understand the shape of the dataset

In [206…
```
# Checking the shape of the data (number of rows and columns)
data.shape
rows, columns = data.shape
```

```
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
```

```
Number of rows: 5000
Number of columns: 14
```

Checking the data types of the columns of the dataset

In [207…

```
# Checkng the data types of the columns in the dataset
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
```

Checking for missing values

In [208…

```
# Checking for missing values
data.isnull().sum()
```

Out[208…
```
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
```

Observations

- There is no missing values in the dataset

Checking for duplicate values in the data set

In [209…
```
# print duplicate values
print(data[data.duplicated(keep=False)])
```

```
Empty DataFrame
Columns: [ID, Age, Experience, Income, ZIPCode, Family, CCAvg, Education, Mortgage, Personal_Loan, Securities_Account, CD_Accou
nt, Online, CreditCard]
Index: []
```

In [210…
```
# Check for duplicate values in ID column
data['ID'].duplicated().sum()
```

Out[210…
```
0
```

Observations

- There are no duplicate values in ID (Customer ID). Therefore, there are no duplicate values in this column.
- Duplicate values are expected in the other columns

Dropping columns

We will drop the ID column as it does not add any value to the analysis

In [211…
```
data.drop(columns=['ID'],inplace=True)
```

Check the statistical summary of the data

In [212…
```
# Checking the statistical summary of the data
data.describe().T
```

Out[212…

	count	mean	std	min	25%	50%	75%	max
Age	5000.0	45.338400	11.463166	23.0	35.0	45.0	55.0	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.0	20.0	30.0	43.0
Income	5000.0	73.774200	46.033729	8.0	39.0	64.0	98.0	224.0
ZIPCode	5000.0	93169.257000	1759.455086	90005.0	91911.0	93437.0	94608.0	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.0	2.0	3.0	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.7	1.5	2.5	10.0
Education	5000.0	1.881000	0.839869	1.0	1.0	2.0	3.0	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.0	0.0	101.0	635.0
Personal_Loan	5000.0	0.096000	0.294621	0.0	0.0	0.0	0.0	1.0
Securities_Account	5000.0	0.104400	0.305809	0.0	0.0	0.0	0.0	1.0
CD_Account	5000.0	0.060400	0.238250	0.0	0.0	0.0	0.0	1.0
Online	5000.0	0.596800	0.490589	0.0	0.0	1.0	1.0	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.0	0.0	1.0	1.0

Observations

- *Age* is evenly distributed, ranging from 23 to 67.
- *Experience* is evenly distributed, ranging from -3 to 43.
- *Income* is right-skewed with large values pulling and raising the mean.
- *Zipcode* will be treated as a categorical variable as it will not have meaningful numerical relationships.
- *Family* is normally distributed, ranging from 1 to 4.
- *CCAvg* is slightly right-skewed, ranging from 0 to 10.
- *Education* is slightly left-skewed, ranging from 1 to 3.
- *Mortgage* is heavily right-skewed, ranging from 0 to 635.
- *Personal_Loan is a binary value, with 9.6% of customers having a personal loan.
- *Securities_Account* is a binary value, with 10% of customers having a securities account.
- *CD_Account* is a binary value, with 6% of customers having a CD Account.
- *Online* is a binary value, with 59% of customers banking online.
- *CreditCard* is a binary value, with 29% of users having a credit card.

Data Processing part 1

Checking for anomalous data

In [213…
```
# Check what unique values are in Experience
data['Experience'].unique()
```

Out[213…
```
array([ 1, 19, 15,  9,  8, 13, 27, 24, 10, 39,  5, 23, 32, 41, 30, 14, 18,
       21, 28, 31, 11, 16, 20, 35,  6, 25,  7, 12, 26, 37, 17,  2, 36, 29,
        3, 22, -1, 34,  0, 38, 40, 33,  4, -2, 42, -3, 43])
```

- Experience has some negative values that are not correct (-1, -2 and -3). For this analysis, I am going to convert those to positive numbers of the same value.

In [214…
```
# Correcting the experience values
data["Experience"].replace(-1, 1, inplace=True)
```

```
data["Experience"].replace(-2, 2, inplace=True)
data["Experience"].replace(-3, 3, inplace=True)
```

In [215…
```
# verify the data is now correct
data['Experience'].unique()
```

Out[215…
```
array([ 1, 19, 15,  9,  8, 13, 27, 24, 10, 39,  5, 23, 32, 41, 30, 14, 18,
       21, 28, 31, 11, 16, 20, 35,  6, 25,  7, 12, 26, 37, 17,  2, 36, 29,
        3, 22, 34,  0, 38, 40, 33,  4, 42, 43])
```

Zip Code analysis and fearure engineering

- My plan is to evaluate how many unique zip codes there are and reduce the digits of the zipcode untel we get a number that we can use realistically and meaningfully in this analysis.
- After looking at this, using the fist 2 digits looks like the right choice.

In [216…
```
# Count unique ZIP codes
unique_zip_codes = data["ZIPCode"].nunique()
print(f"Number of unique ZIP codes: {unique_zip_codes}")
```

```
Number of unique ZIP codes: 467
```

In [217…
```
# Process ZIPCode: truncate to first 2 digits and convert to categorical
data["ZIPCode"] = data["ZIPCode"].astype(str).str[:2].astype("category")
# Show the resutls
print(f"Number of unique ZIP code prefixes: {data['ZIPCode'].nunique()}")
```

```
Number of unique ZIP code prefixes: 7
```

Convert binary variables to category type

In [218…
```
# Convert binary variables to 'category' type
cat_cols = [
    "Personal_Loan",
    "Securities_Account",
    "CD_Account",
    "Online",
    "CreditCard",
```

```
    "ZIPCode"
]
data[cat_cols] = data[cat_cols].astype("category")
```

In [219…

```
# Map the Education values into 1= Undergraduate, 2= Graduate, 3= Advanced/Professional
data['Education'].replace(1, 'Undergraduate', inplace=True)
data['Education'].replace(2, 'Graduate', inplace=True)
data['Education'].replace(3, 'Advanced/Professional', inplace=True)
```

Exploratory Data Analysis.

In [220…

```
# Function to make a histogram and boxplot
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined
    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    # Creating the 2 subplots
    # Number of rows of the subplot grid = 2
    # x-axis will be shared among all subplots
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,
        sharex=True,
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )

    # Boxplot will be created and a star will indicate the mean value of the column
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )

    # For histogram
    if bins:
        sns.histplot(
```

```
            data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
        )
    else:
        sns.histplot(
            data=data, x=feature, kde=kde, ax=ax_hist2
        )

    # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )

    # Add median to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )
```

In [221…

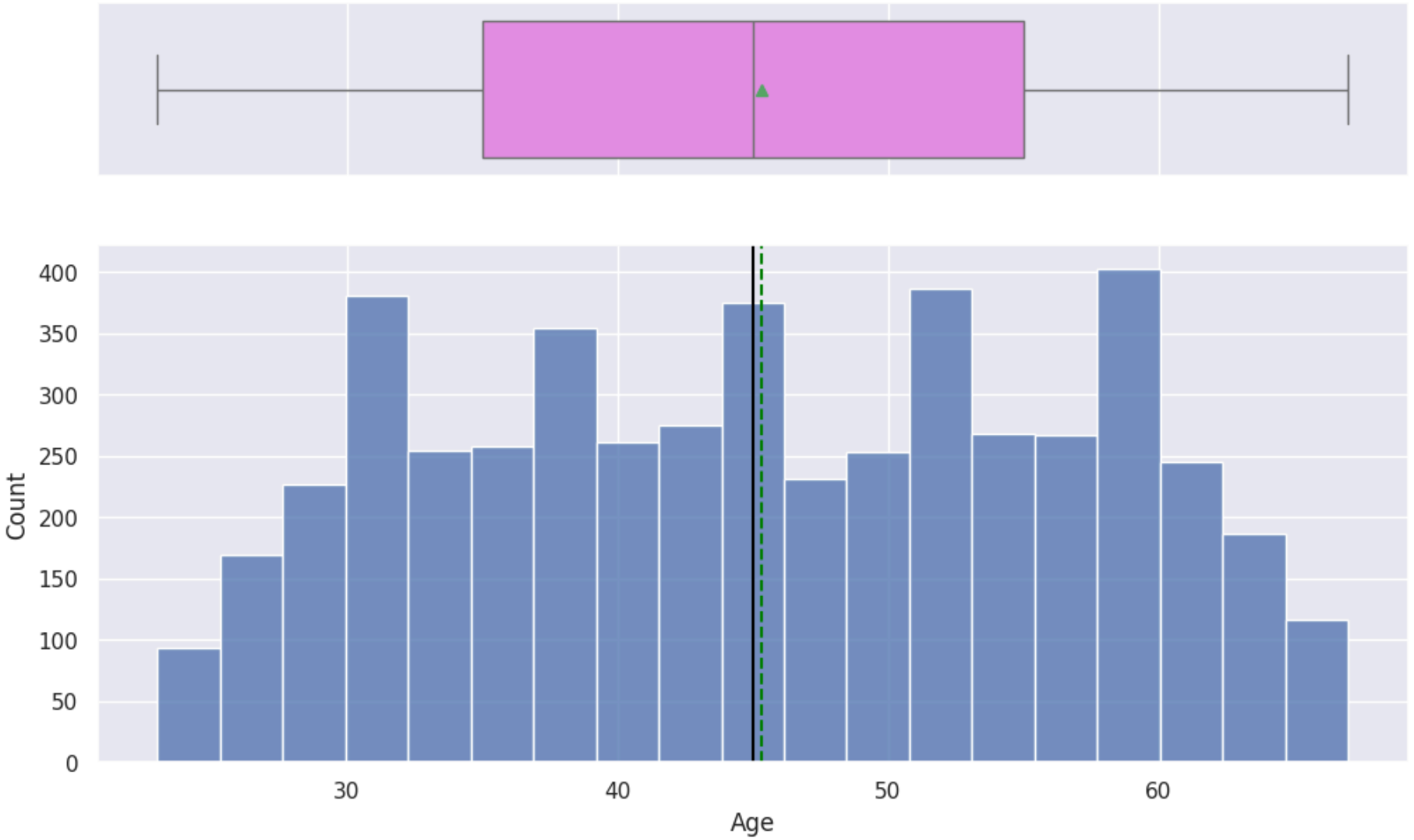
```
# Function to create labeled barplots
def labeled_barplot(data, feature, perc=False, n=None):
    """
    Creates a bar plot with labels on top of each bar.

    Parameters:
    data (DataFrame): The input dataframe
    feature (str): The column name to be plotted
    perc (bool): If True, display percentages instead of counts (default False)
    n (int): Number of top categories to display (default None, displays all)
    """

    total = len(data[feature])  # Get total number of rows
    count = data[feature].nunique()  # Count unique values in the feature

    # Set figure size based on the number of categories
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)  # Rotate x-axis labels for readability
```

```
    # Create the count plot
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    # Add labels on top of each bar
    for p in ax.patches:
        if perc:
            label = "{:.1f}%".format(100 * p.get_height() / total)  # Calculate percentage
        else:
            label = p.get_height()  # Use count

        # Calculate the position for the label
        x = p.get_x() + p.get_width() / 2
        y = p.get_height()

        # Add label to the plot
        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )

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

Graphs and observations on Age

In [222…

```
histogram_boxplot(data, "Age")
```

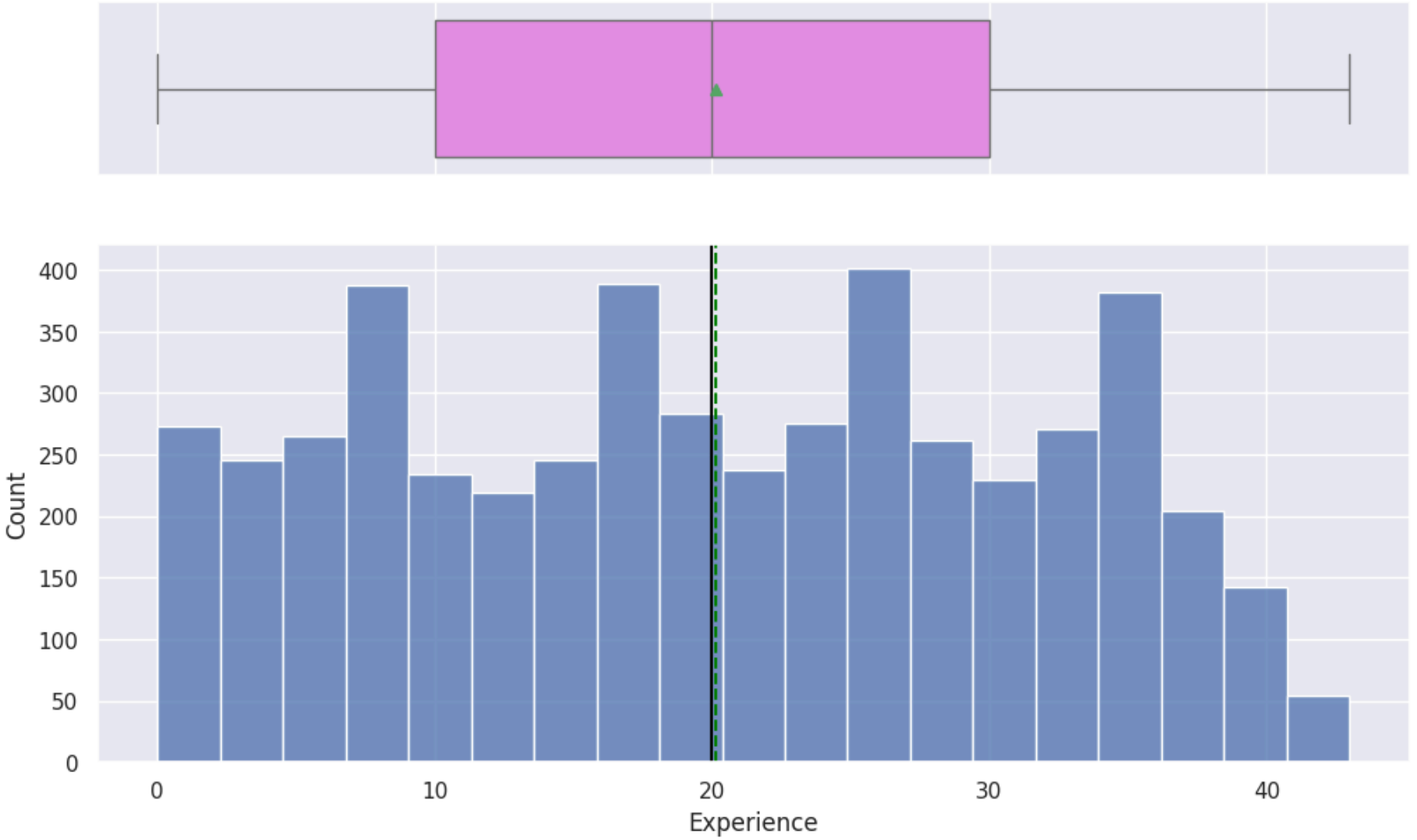


Observations

- The data is normally distributed with no outliers.

Graphs and observations on Experience

In [223…
```
histogram_boxplot(data, "Experience")
```



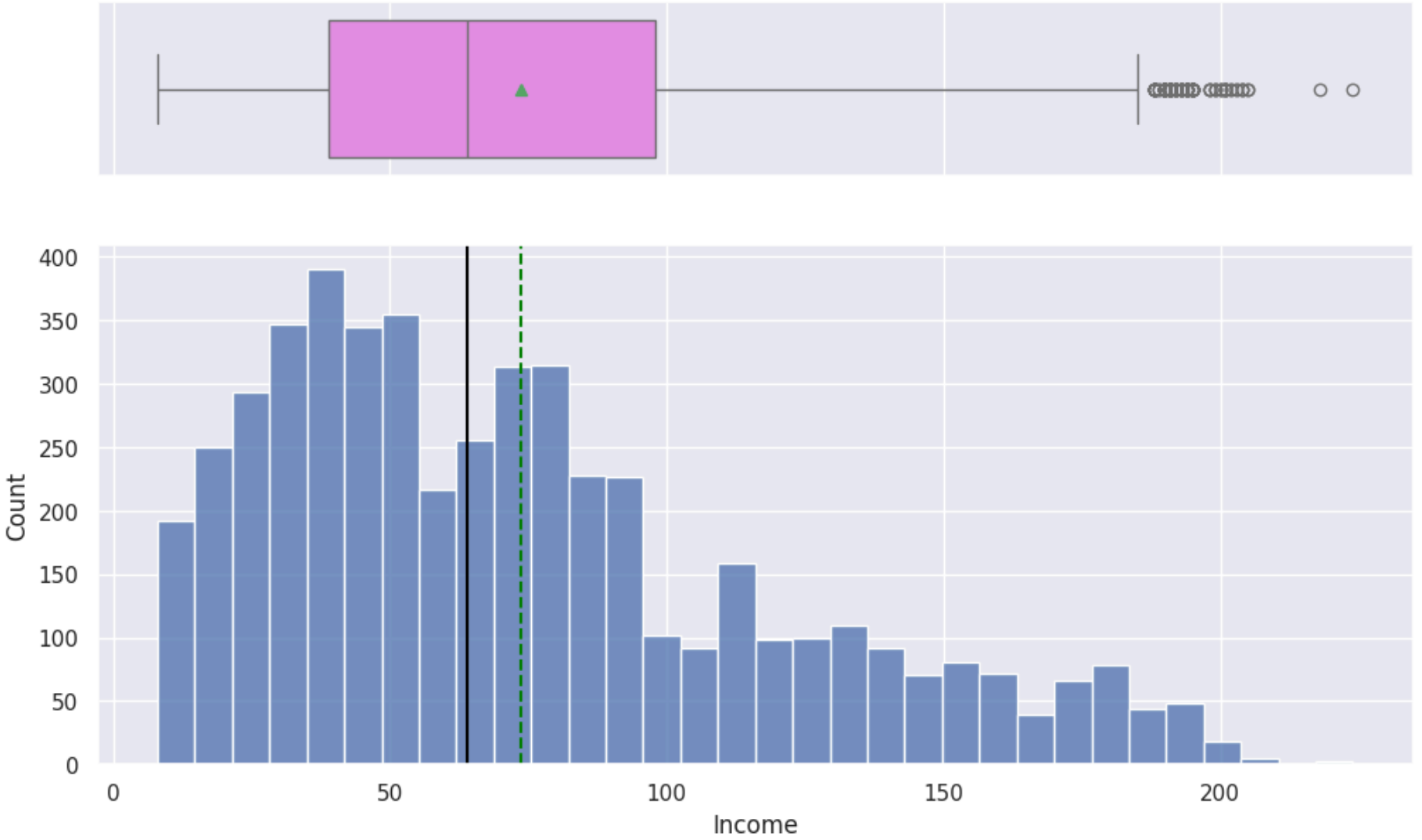
Observations

- This variable is normally distributed with no outliers

Graphs and observations on Income

In [224…

```
histogram_boxplot(data, "Income")
```



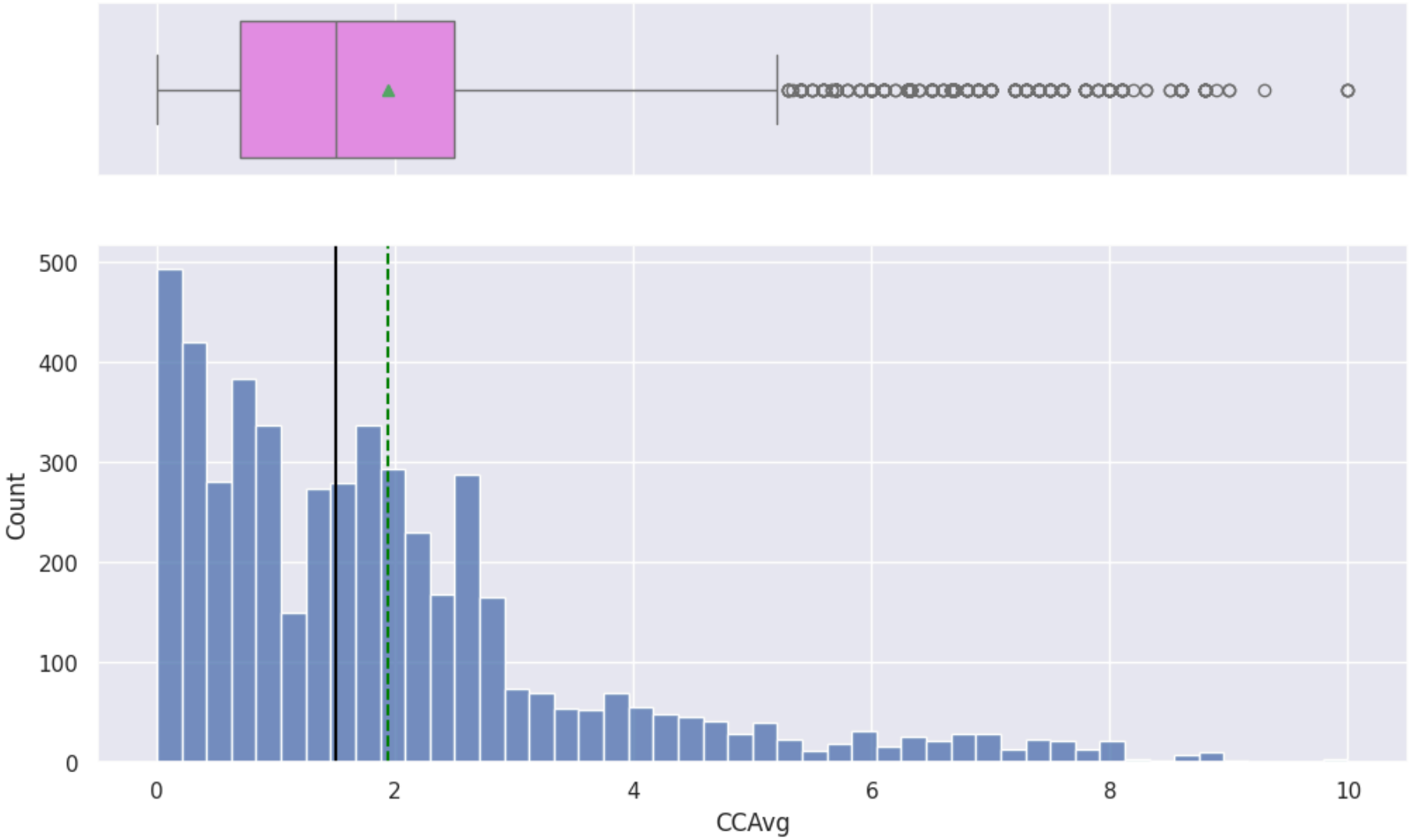
Observations

- This variable is right-skewed
- There are many outliers above the 3rd quartile in this variable.

Graphs and observations on CCAvg

In [225…

```
histogram_boxplot(data, "CCAvg")
```

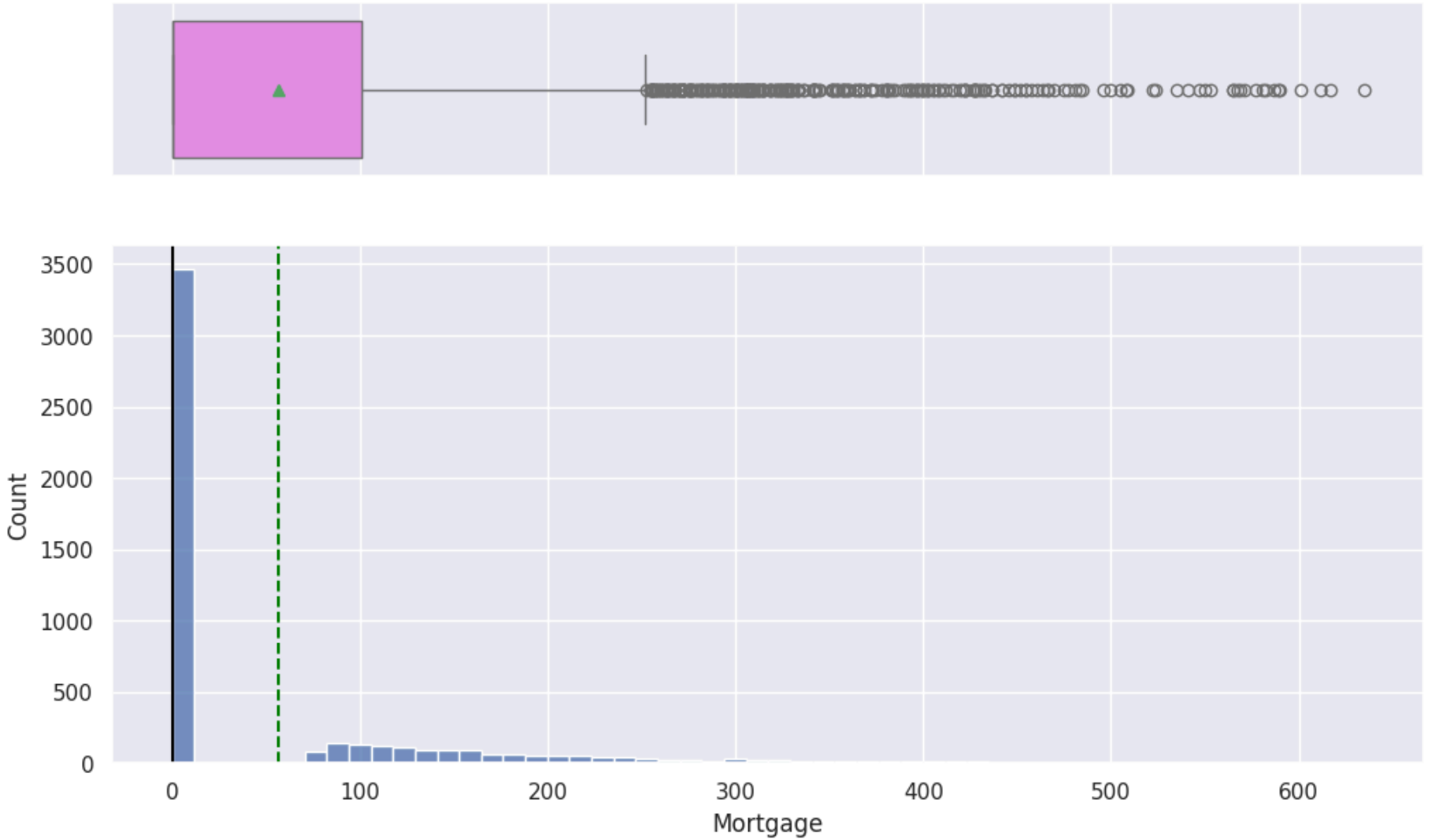


Observations

- This variable is right-skewed.
- There are many outliers above the 3rd quartile in this variable.

Graphs and observations on Mortgage

In [226…
```
histogram_boxplot(data, "Mortgage")
```



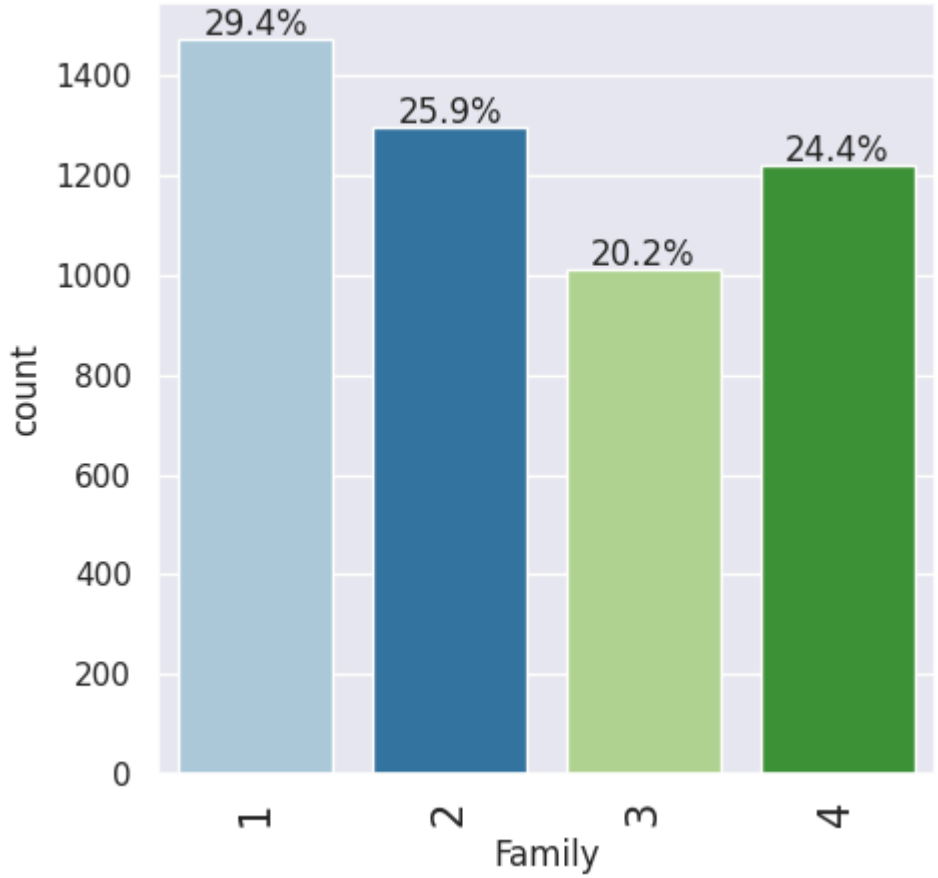
Observations

- This variable is heavily right-skewed.
- There are many outliers above the 3rd quartile in this variable.

Graphs and observations on Family

In [227…

```
labeled_barplot(data, "Family", perc=True)
```



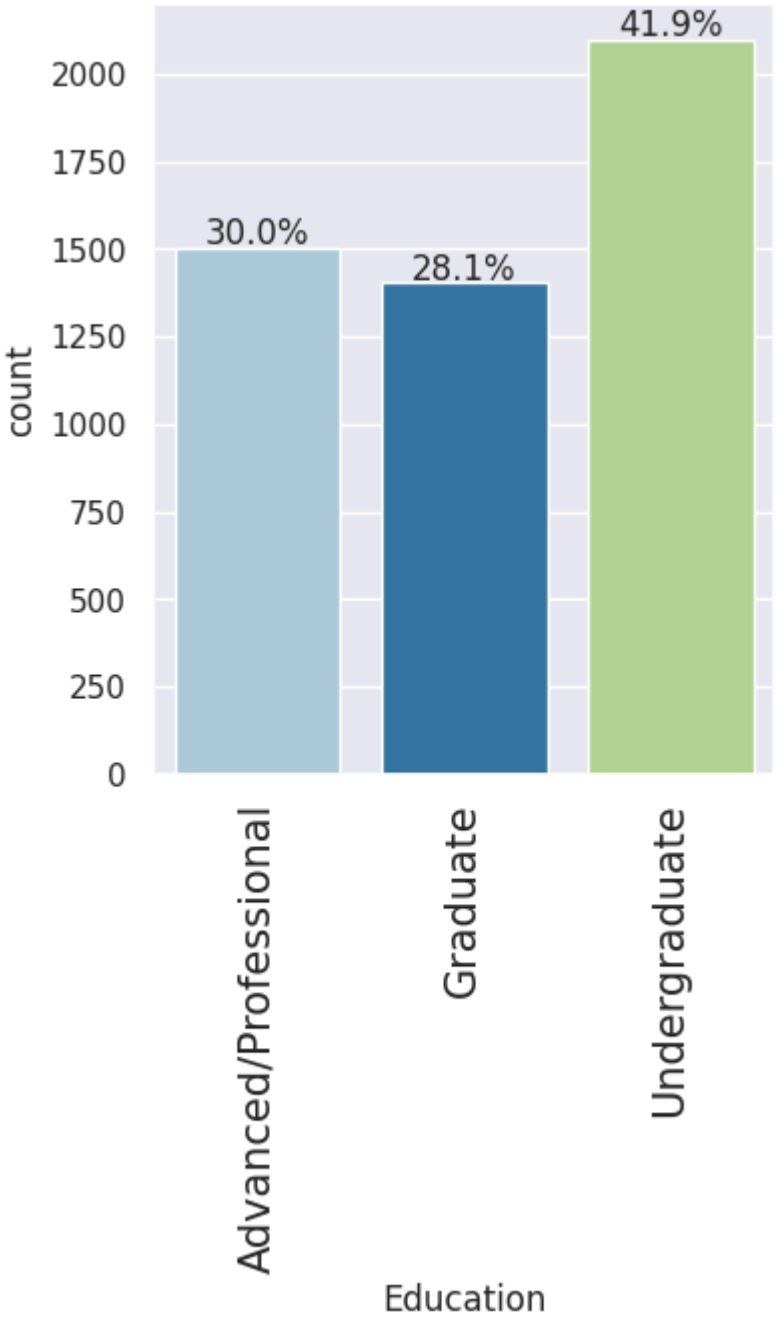
Observations

- 29.4% of customers have one child.
- only 20.2% of customers have three children.

Graphs and observations on Education

In [228…

```
labeled_barplot(data, "Education", perc=True)
```


41.9%
30.0%
28.1%
2000
1750
1500
1250
1000
750
500
250
0
count
Advanced/Professional
Graduate
Undergraduate
Education

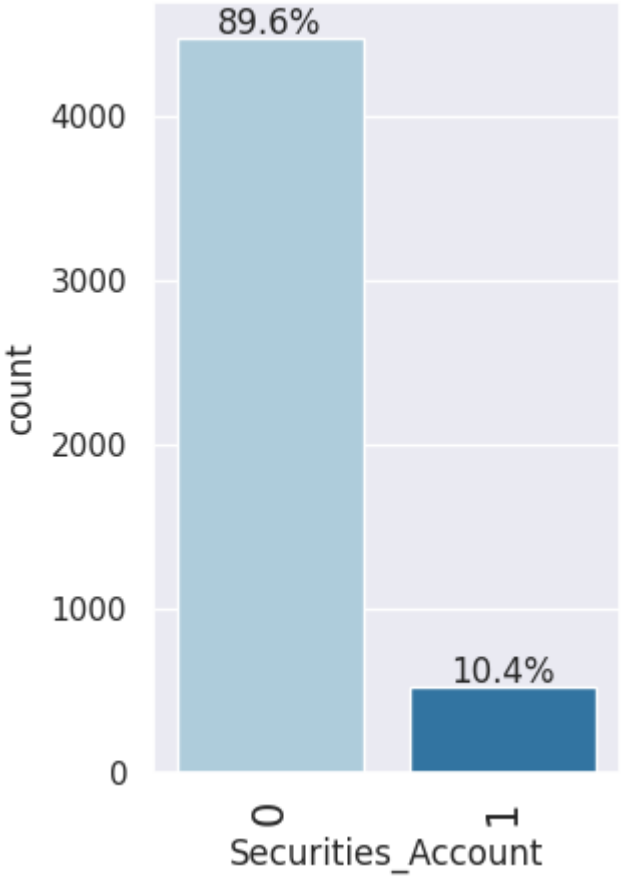
Observations

- The majority (42%) of customers have an undergraduate level of education.
- 28% of customers have a graduate level of education.
- 30% of customers have advanced/professional education

Graphs and observations on Securities Account

In [229…

```
labeled_barplot(data, "Securities_Account", perc=True)
```



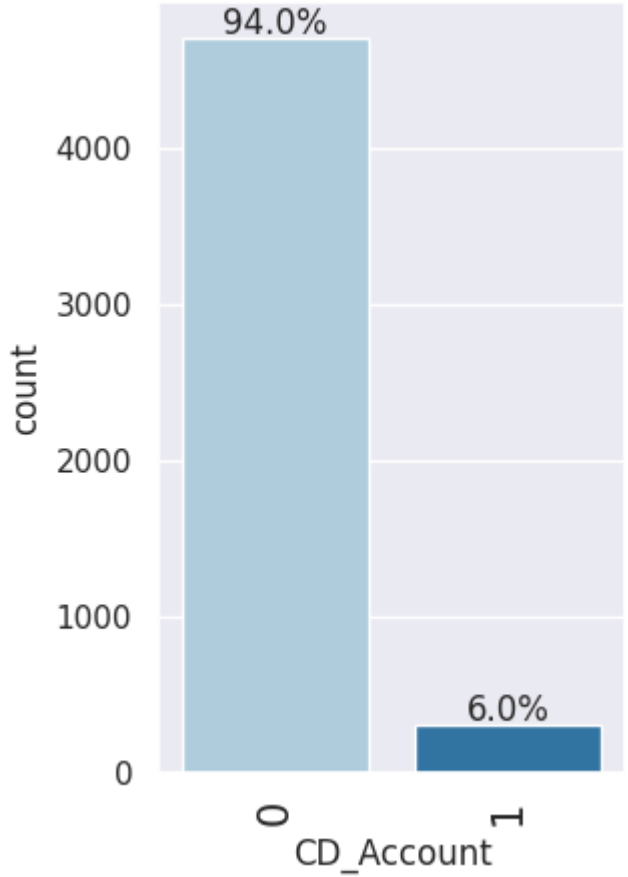
Observations

- 89.6% of customers do not have a securities account

Graphs and observations on CD_Account

In [230…

```
labeled_barplot(data, "CD_Account", perc=True)
```



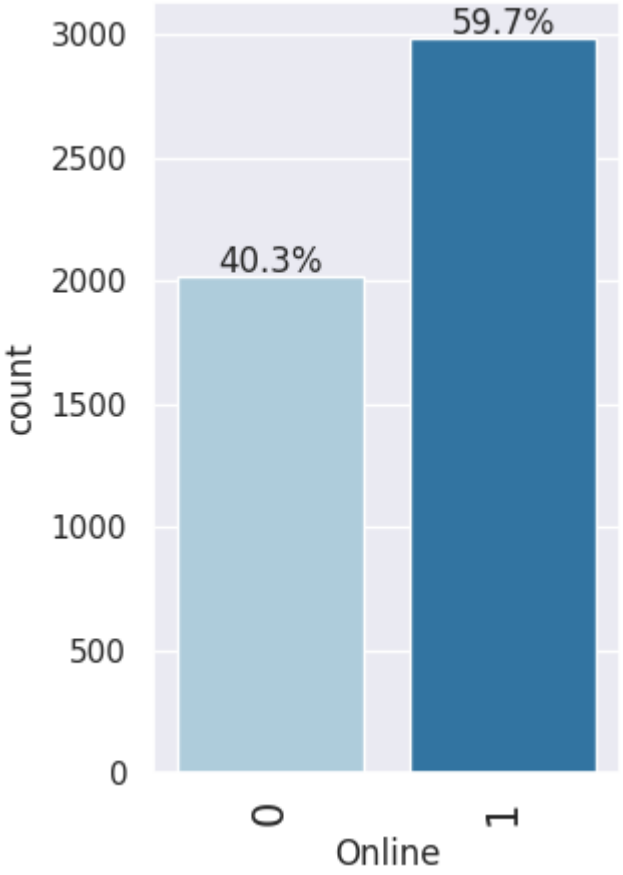
Observations

- 94% of customers do not have a CD account

Graphs and observations on Online

In [231…

```
labeled_barplot(data, "Online", perc=True)
```

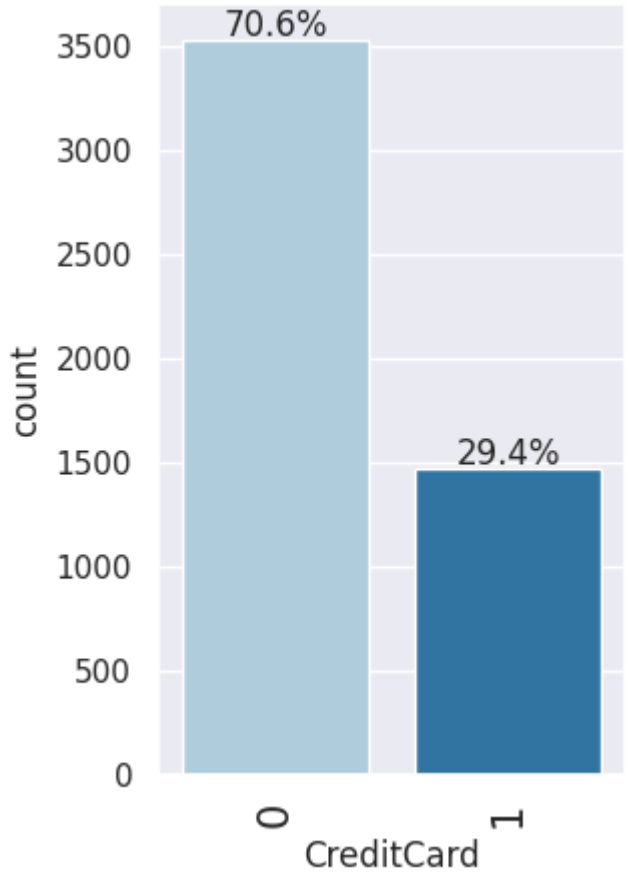


Observations

- 59% of users do their banking online

Graphs and observations on CreditCard

In [232…
```
labeled_barplot(data, "CreditCard", perc=True)
```

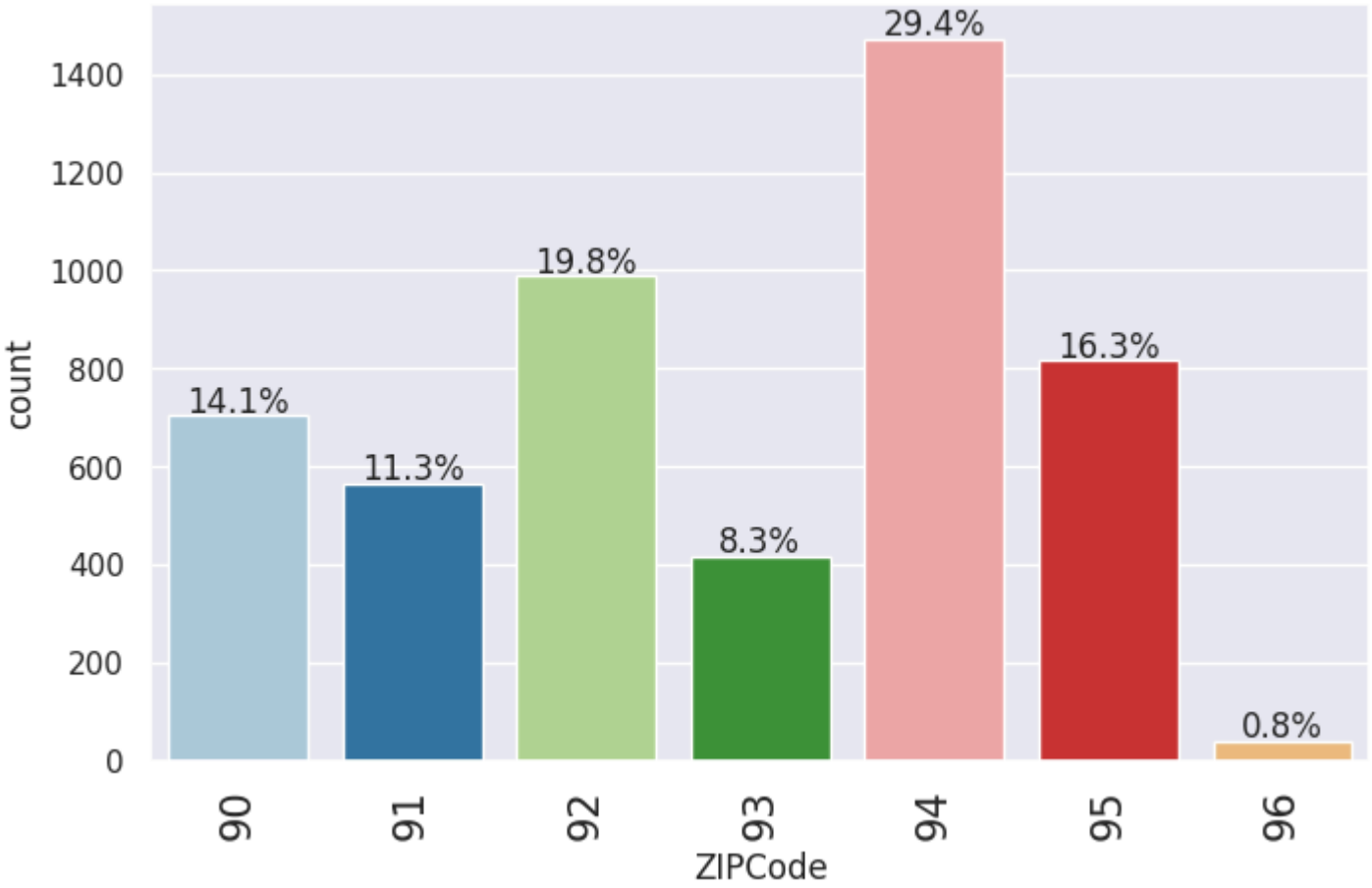


Observations

- 70% of customers do not have a credit card

Graphs and observations on ZIPCode

In [233…
```
labeled_barplot(data, "ZIPCode", perc=True)
```



Observations

- The majority of customers (29.4%) live in area 94.
- Very few customers (0.8%) live in area 96

Bivariate Analysis

In [234…

```
# function to create stacked barplots
def stacked_barplot(data, predictor, target):
    """
```

```
    Print category counts and plot a stacked bar chart.

    Parameters:
    data (DataFrame): Input dataframe
    predictor (str): Independent variable column name
    target (str): Target variable column name
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]

    # Create and display cross-tabulation
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(by=sorter, ascending=False)
    print(tab1)
    print("-" * 120)

    # Create normalized cross-tabulation for plotting
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values(by=sorter, ascending=False)

 # Plot stacked bar chart
    ax = tab.plot(kind="bar", stacked=True, figsize=(count + 5, 5))

    # Adjust legend
    plt.legend(title=target, bbox_to_anchor=(1.05, 1), loc='upper left')

    # Rotate x-axis labels by 90 degrees
    plt.xticks(rotation=90)

    # Adjust layout to prevent cutting off labels
    plt.tight_layout()

    # Show the plot
    plt.show()

    return ax
```

In [235…

```
# function to plot distributions with regards to target
def distribution_plot_wrt_target(data, predictor, target):
    # Create a 2x2 subplot
    fig, axs = plt.subplots(2, 2, figsize=(12, 10))
```

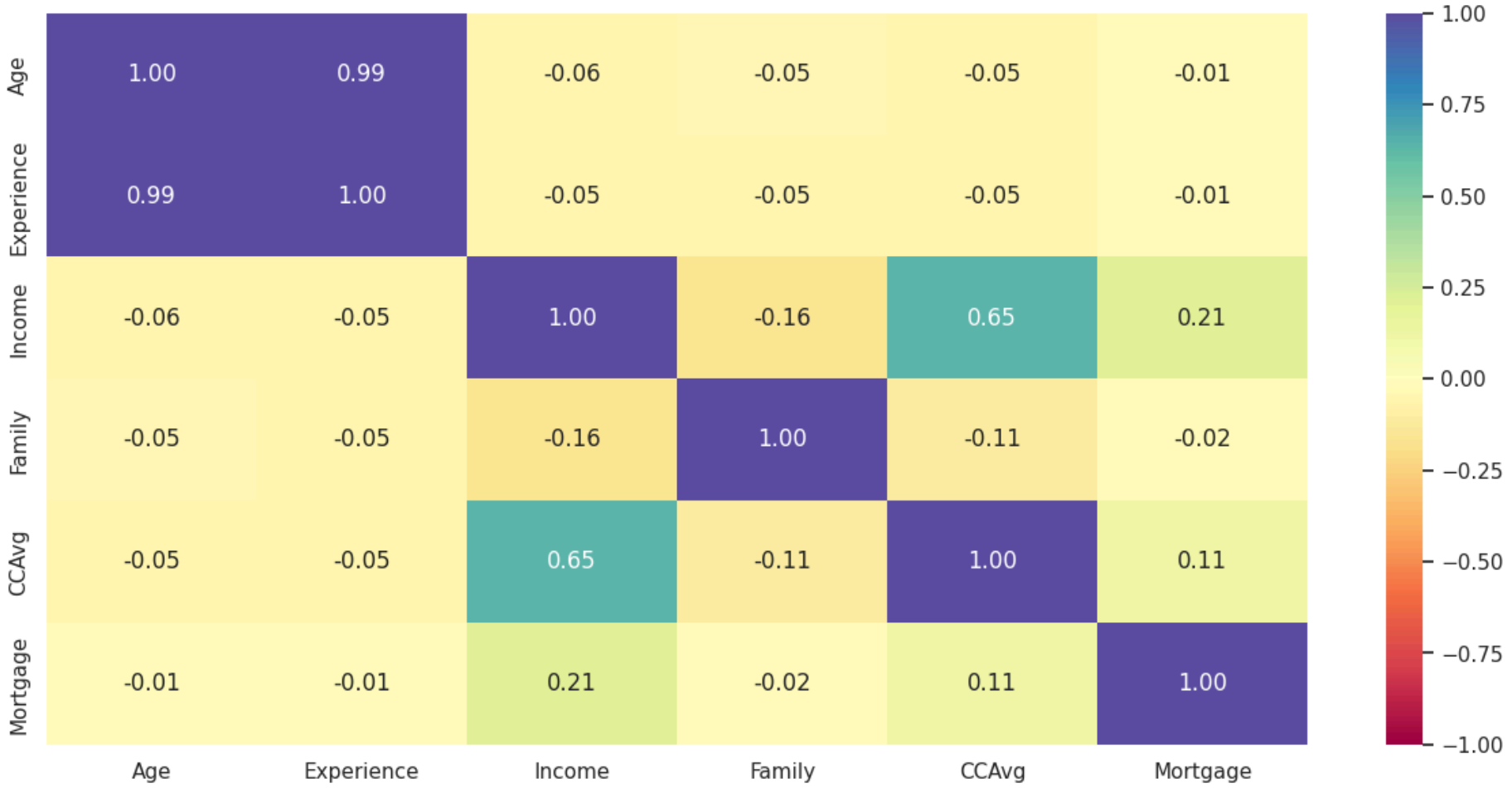
```
    # Get unique values of the target variable
    target_uniq = data[target].unique()

    # Plot distribution for first target value
    axs[0, 0].set_title("Distribution of target for target=" + str(target_uniq[0]))
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
        stat="density",
    )

    # Plot distribution for second target value
    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[1]))
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
        stat="density",
    )

    # Plot boxplot with respect to target
    axs[1, 0].set_title("Boxplot w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist_rainbow")

    # Plot boxplot without outliers
    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    # Adjust layout and display the plot
```

```
    plt.tight_layout()
    plt.show()
```

Data Correleation check

In [236…

```
# Create a heatmap of the correlation matrix of the data
plt.figure(figsize=(15, 7))
sns.heatmap(data.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral");
```



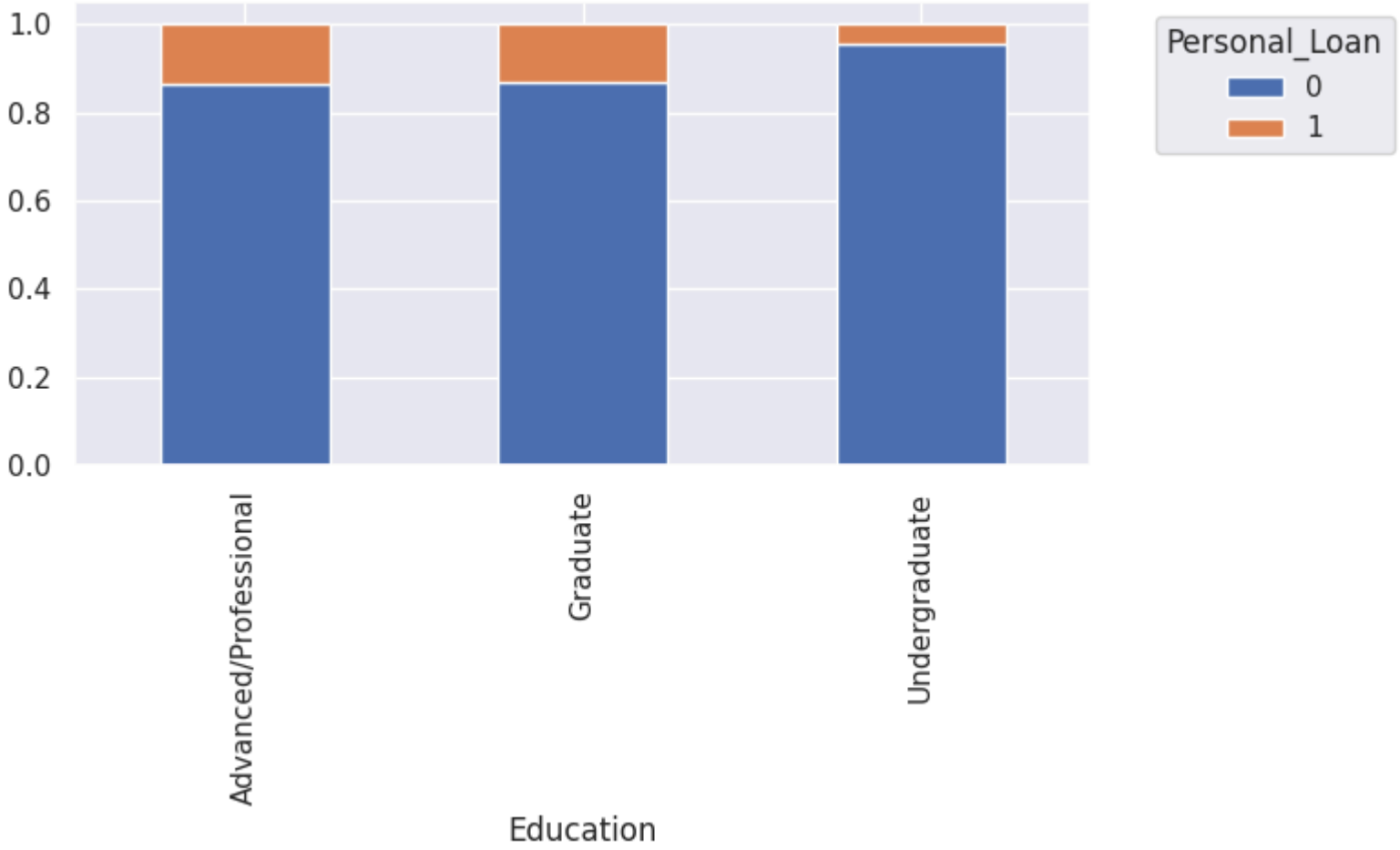
Observations

- There is an extremely strong correlation (0.99) between age and experience.
- There is a strong correlation (.65) between income and credit card average.

Education vs Personal_loan

In [237…
```
stacked_barplot(data, "Education", "Personal_Loan");
```

```
Personal_Loan              0    1   All
Education
All                     4520  480  5000
Advanced/Professional   1296  205  1501
Graduate                1221  182  1403
Undergraduate           2003   93  2096
------------------------------------------------------------------------------------------------------------------------
```



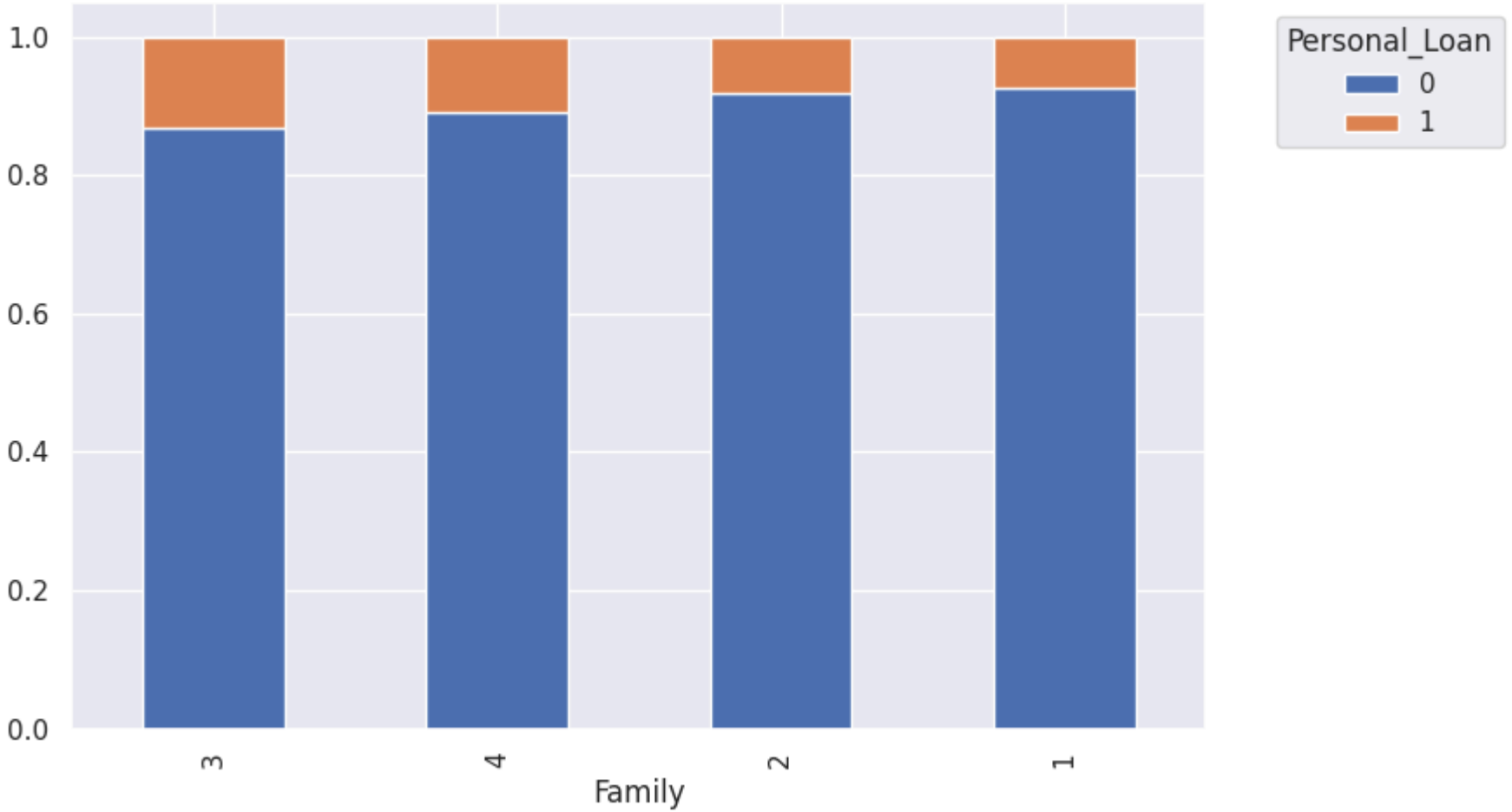
Observations

- 90% of customers do not have a personal loan
- Customers with advanced/professional degrees are about three times more likely to have a personal loan than those with undergraduate degrees.
- Education of advanced/professional and graduates have similar rates of personal loans, both significantly higher than undergraduate.

Personal_Loan vs Family

In [238…
```
stacked_barplot(data, "Family", "Personal_Loan");
```

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



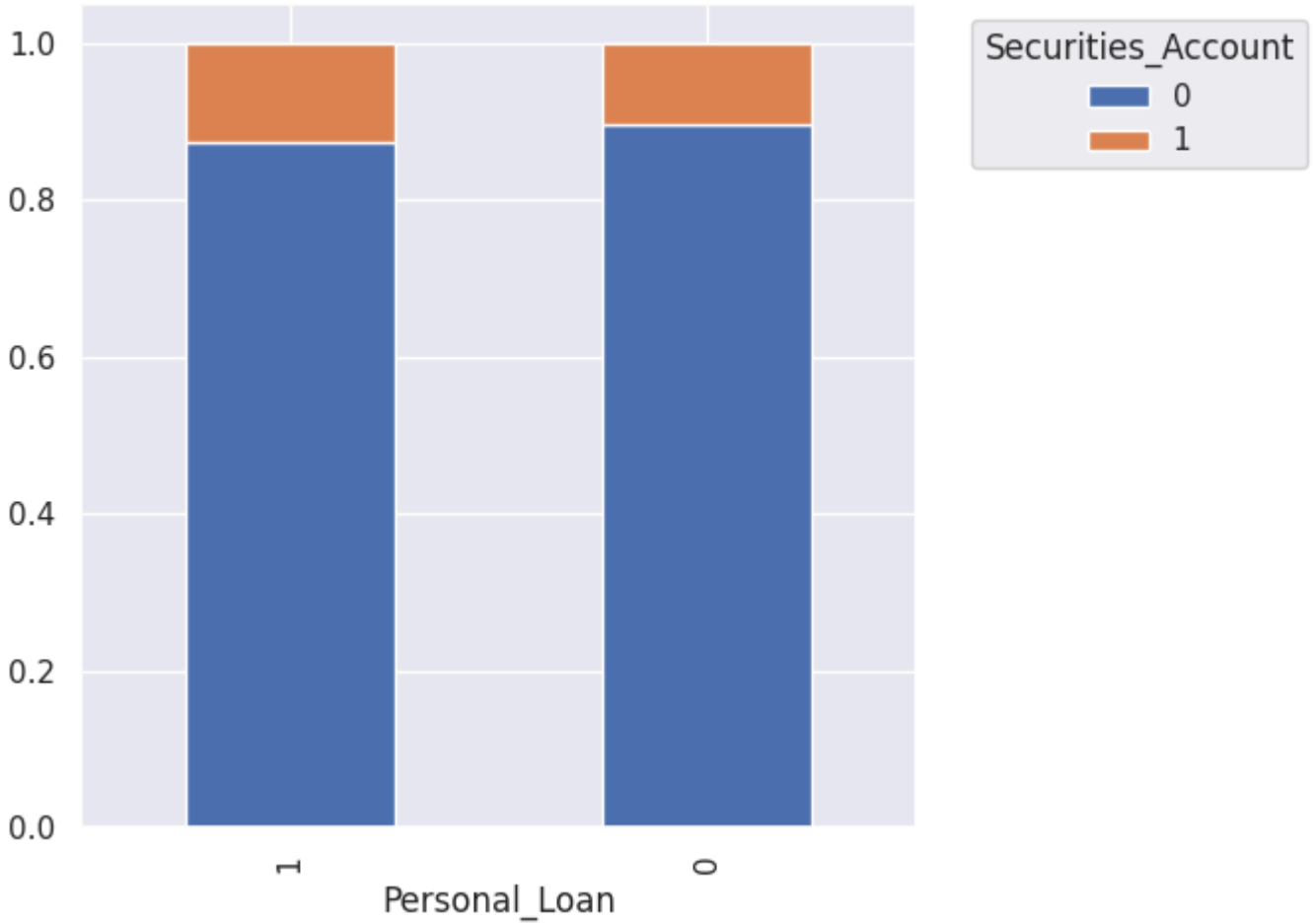
Observations

- 90.4% of families do not have a personal loan
- Larger families (3 -4 persons) seem to have a higher percentage of personal loans compared to smaller families (1- 2 persons)

Personal_Loan vs Securities_Account

In [239…
```
stacked_barplot(data, 'Personal_Loan', 'Securities_Account');
```

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



Observations

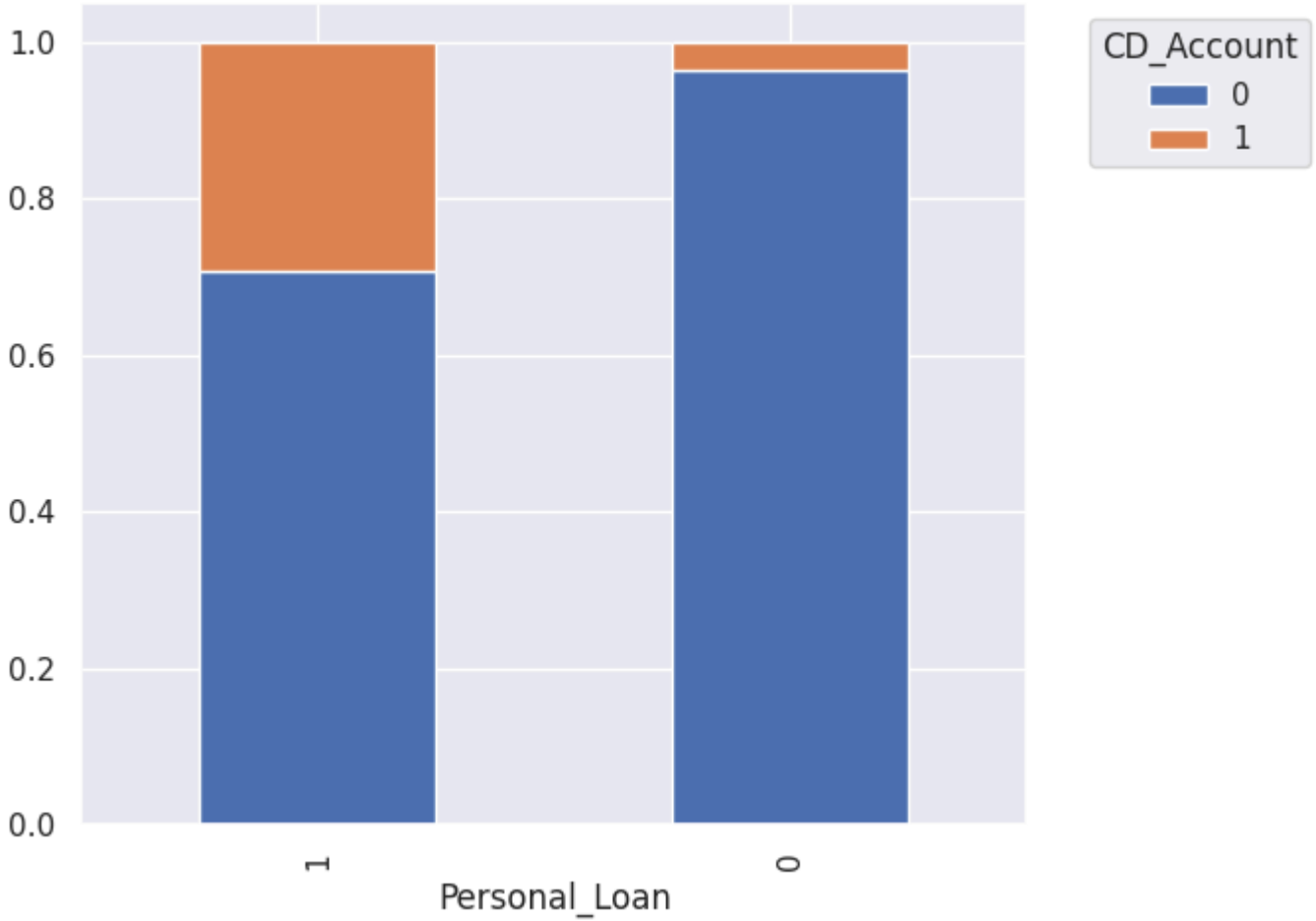
- 89.6% of customers do not have a securities account
- There's a slightly higher percentage of securities account holders among those with personal loans (12.5%) than those without (10.22%).

Personal_Loan vs CD_Account

In [240…

```
stacked_barplot(data, 'Personal_Loan', 'CD_Account');
```

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



Observations

- 93% of customers do not have a CD account.

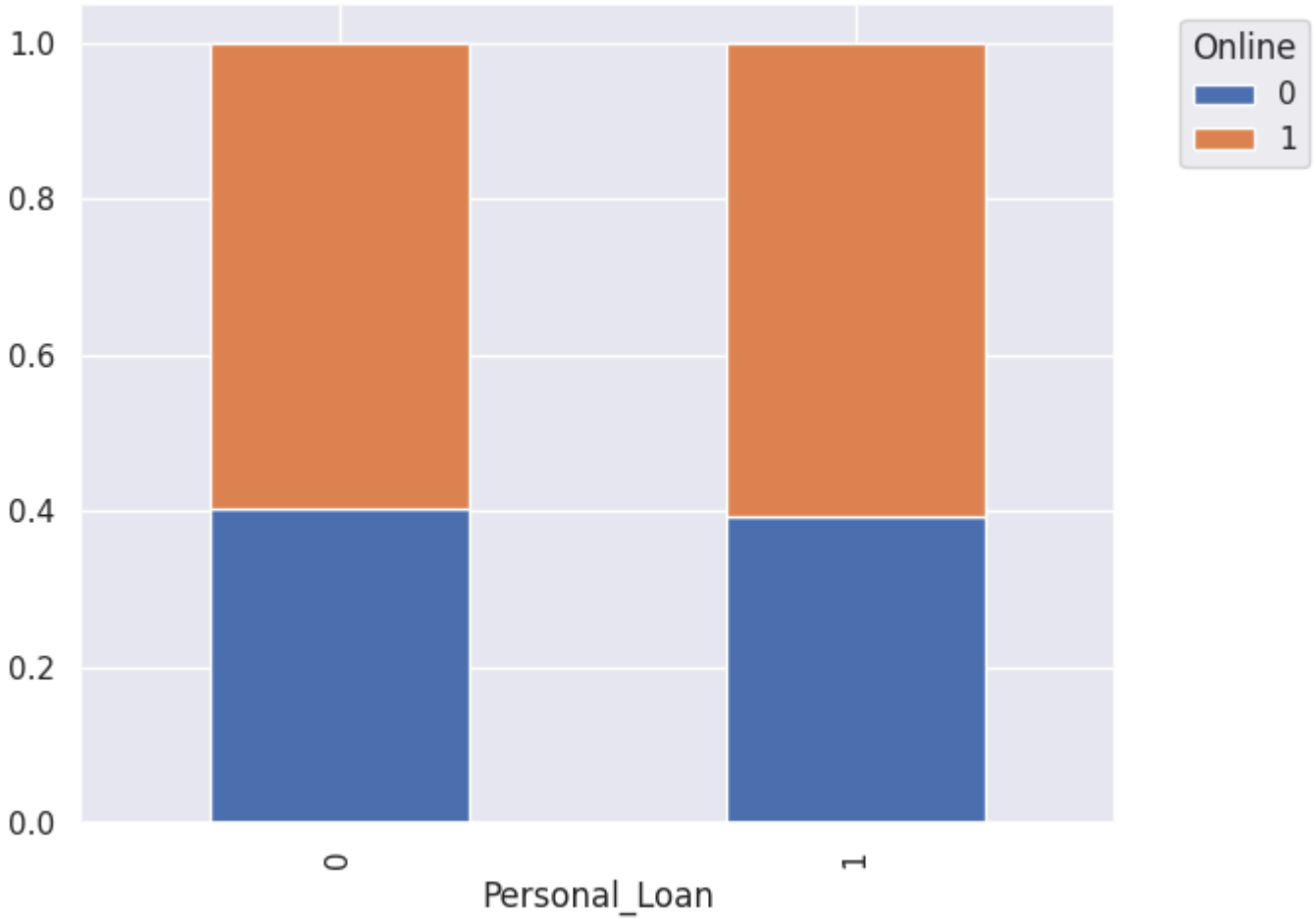
- There seems to be a higher proportion of individuals with a personal loan among those with a CD account than those who do not have a CD account.

Personal_Loan vs Online

In [241…

```
stacked_barplot(data, 'Personal_Loan', 'Online');
```

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



Observations

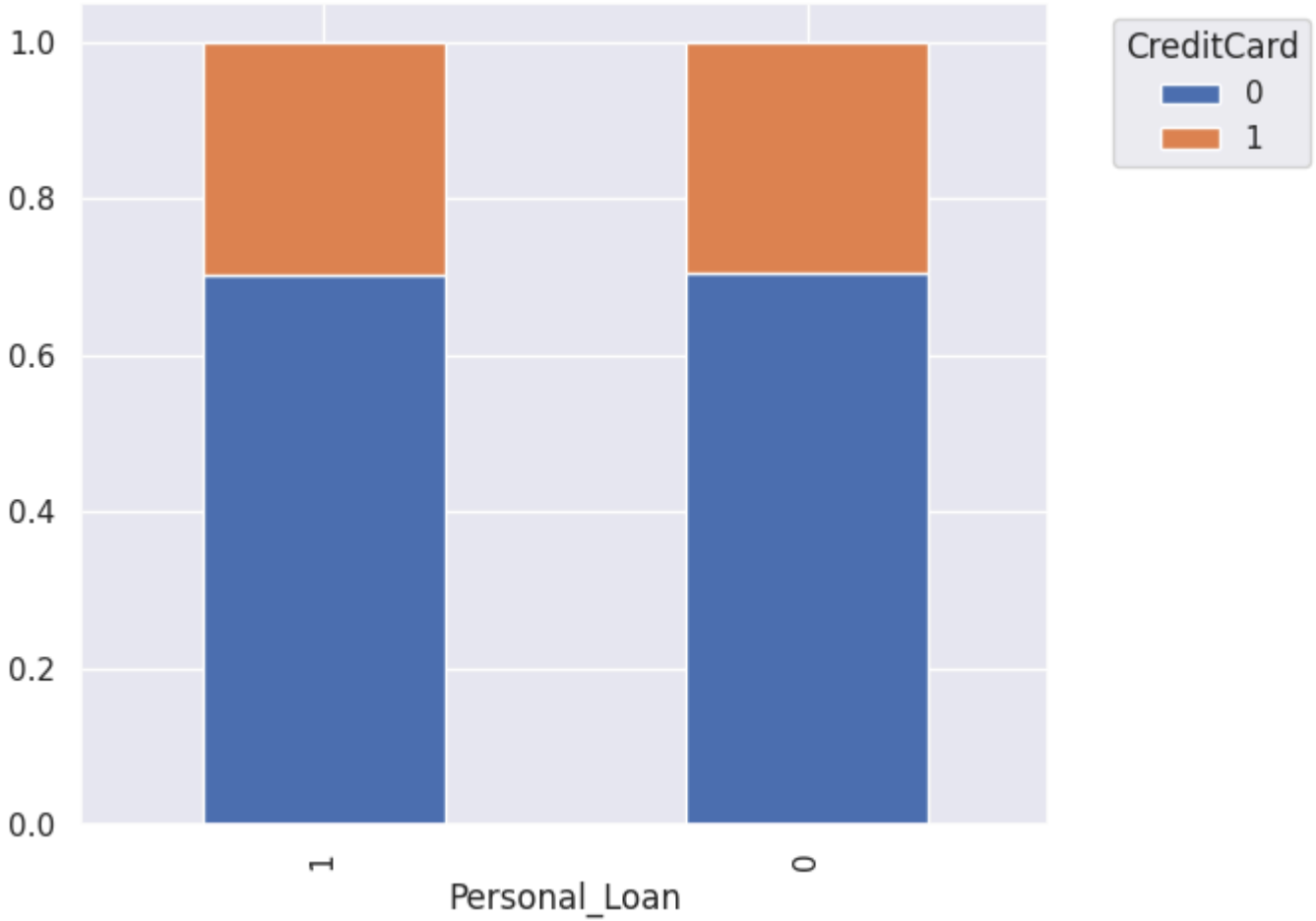
- 59.68% of customers use online banking
- Online banking is more popular than traditional banking in this dataset, regardless of loan status

Personal_Loan vs CreditCard

In [242…

```
stacked_barplot(data, 'Personal_Loan', 'CreditCard');
```

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



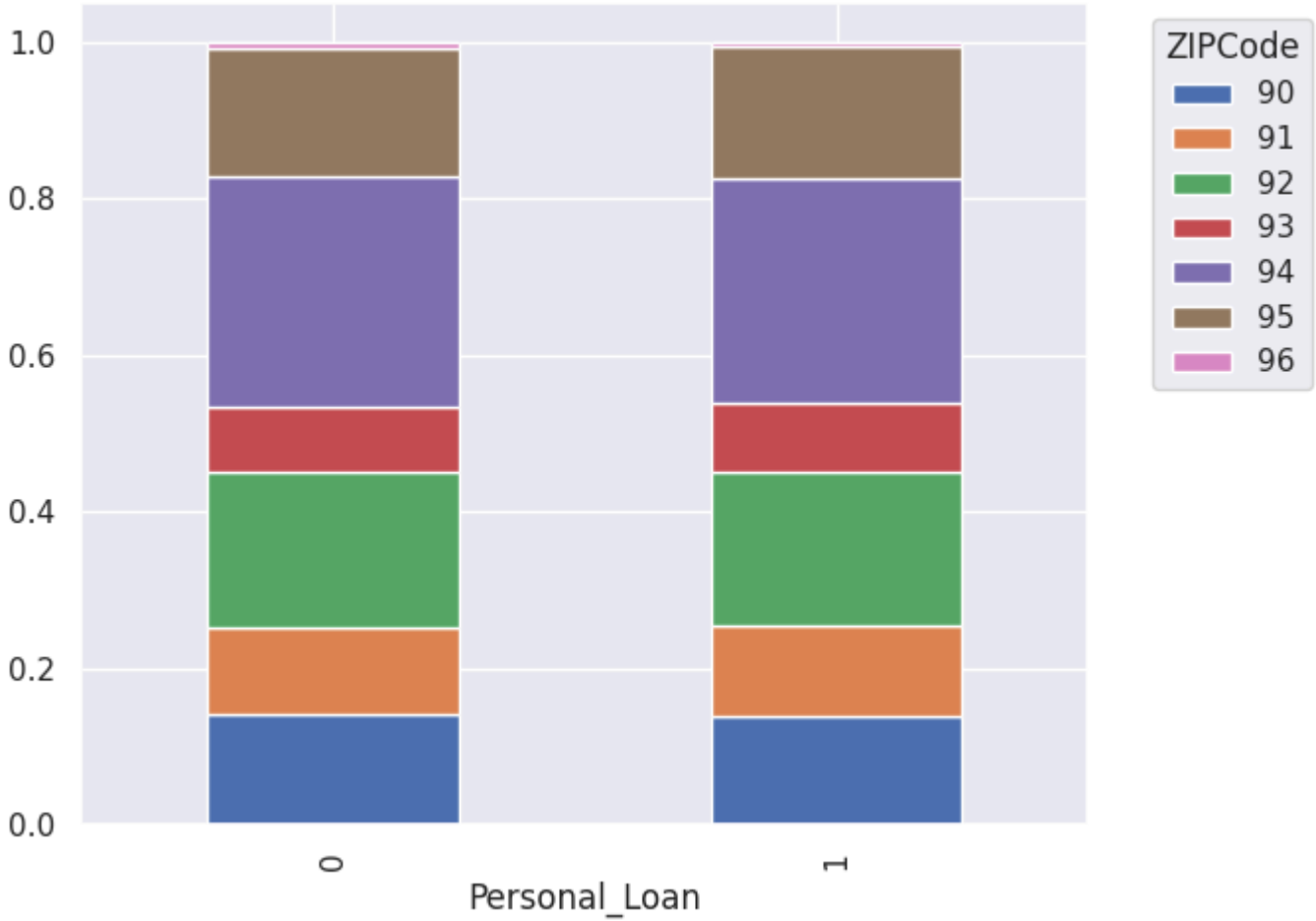
Observations

- 70.6% of customers do not have a credit card.
- There is little difference in credit card ownership rates between those customers with personal loans and those without.

Personal_Loan vs ZIPCode

In [243…
```
stacked_barplot(data, 'Personal_Loan', 'ZIPCode');
```

```
ZIPCode          90   91   92   93    94   95  96   All
Personal_Loan
All             703  565  988  417  1472  815  40  5000
0               636  510  894  374  1334  735  37  4520
1                67   55   94   43   138   80   3   480
```



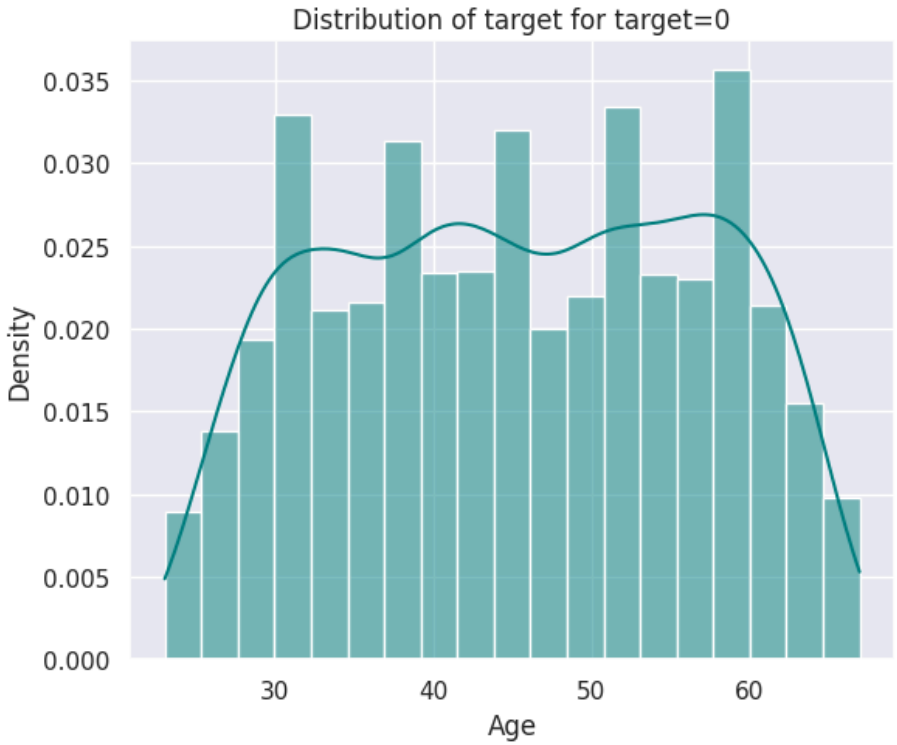
Observations

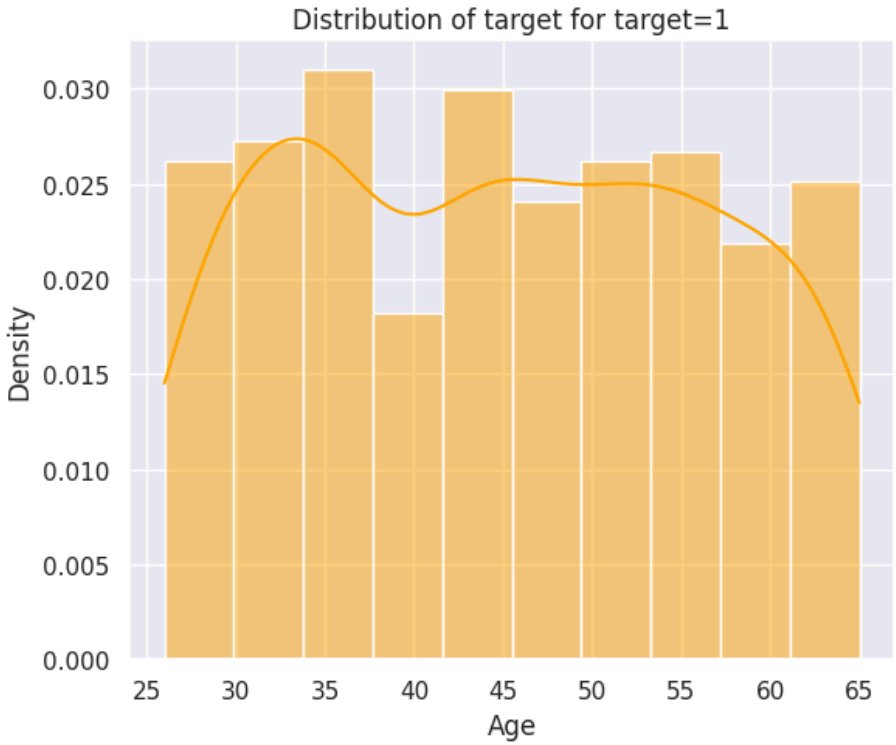
- ZIP code 94 has the highest population (1472, 29.44% of the sample).
- ZIP code 96 has the lowest population (40, 0.8% of the sample).
- Personal loan rates are relatively consistent across most ZIP codes, ranging from 9.37% to 10.31% for ZIP codes 90-95.

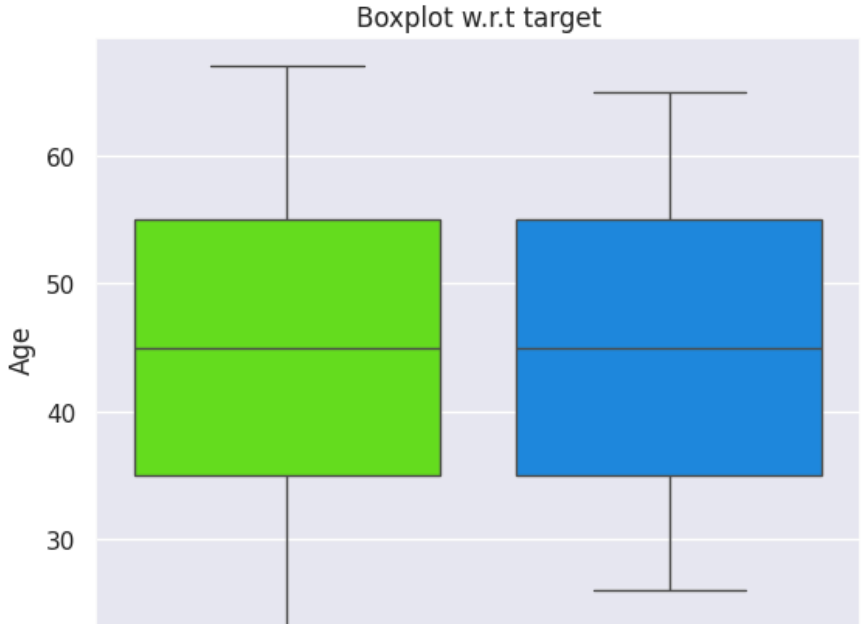
Personal_Loan vs Age

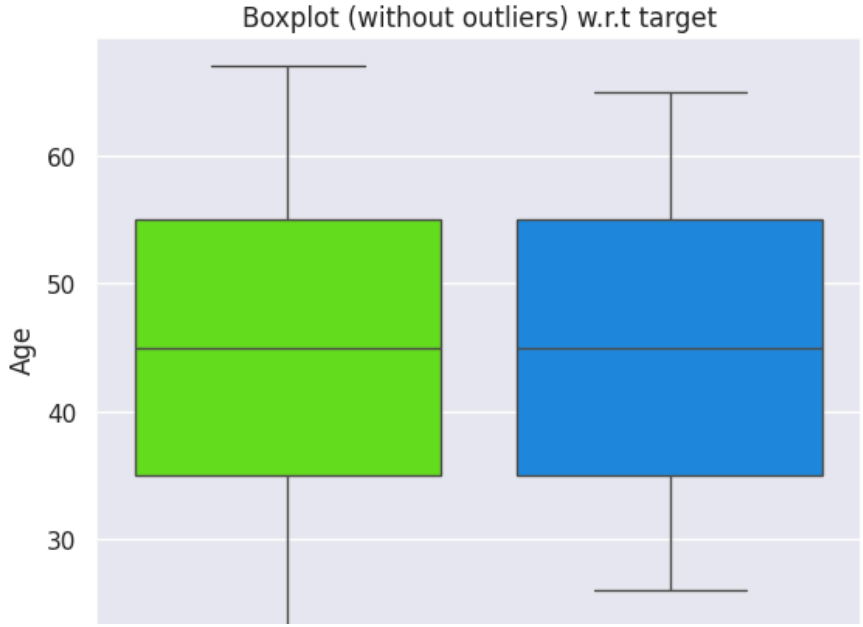
In [244…

```
distribution_plot_wrt_target(data, "Age", "Personal_Loan")
```


Distribution of target for target=0
Density
Age
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
30
40
50
60


Distribution of target for target=1
Density
Age
0.030
0.025
0.020
0.015
0.010
0.005
0.000
25
30
35
40
45
50
55
60
65


Boxplot w.r.t target
Age
60
50
40
30


Boxplot (without outliers) w.r.t target
Age
60
50
40
30

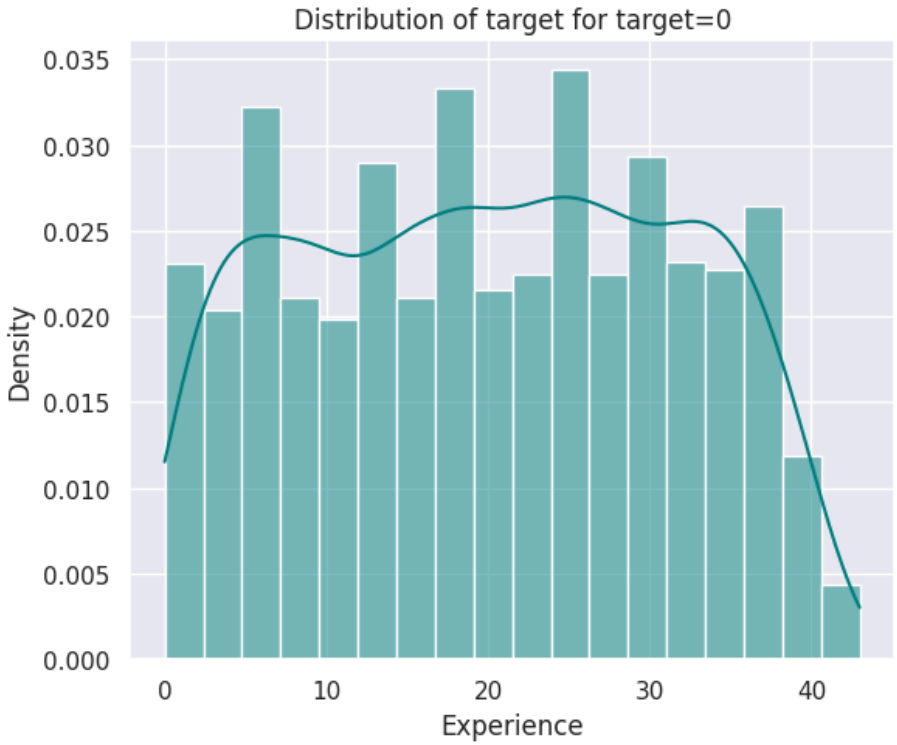
Observations

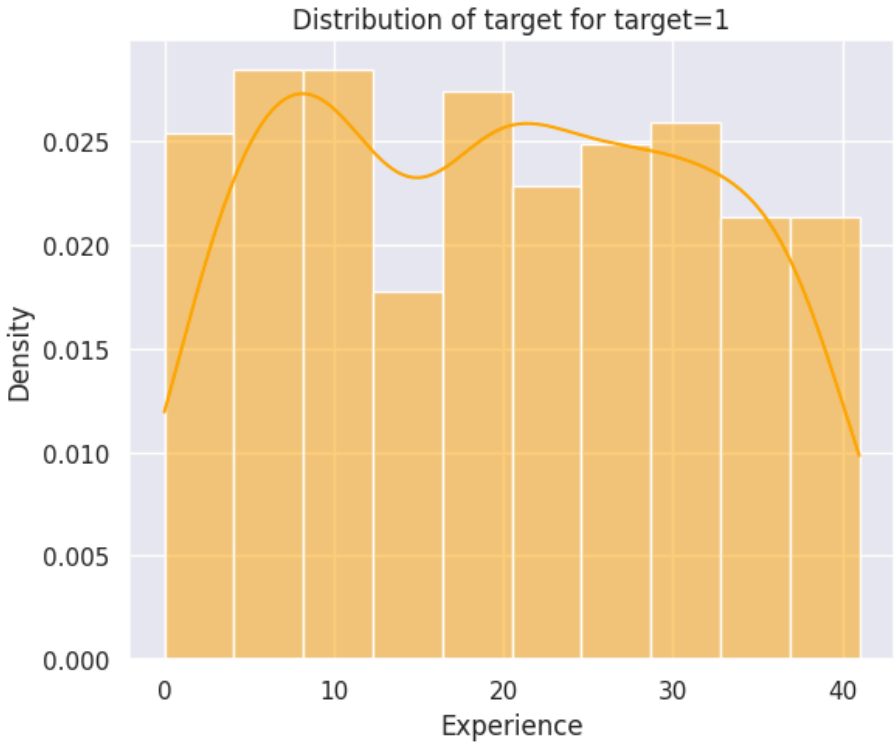
- Age vs personal loan data is normally distributed with no outliers
- Income, CCAvg, and Mortgage have a significant number of values above the 3rd quartile that are considered outliers. However, these values are legitimate and do not require outlier treatment.

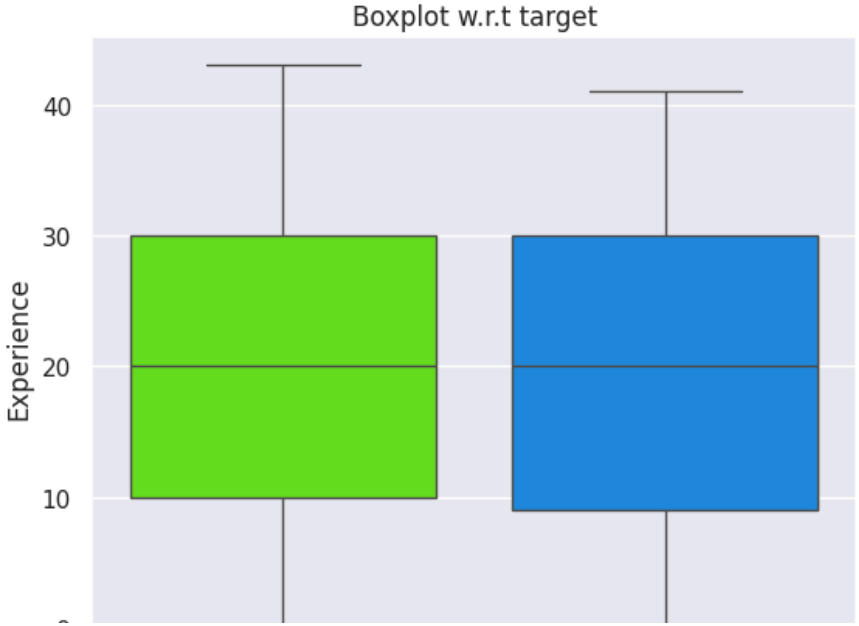
Personal Loan vs Experience

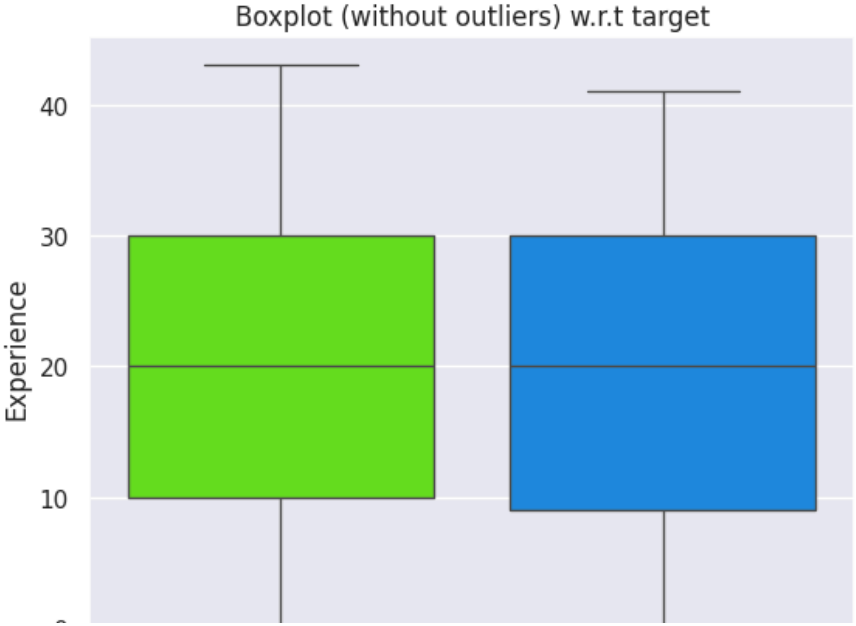
In [245…

```
distribution_plot_wrt_target(data, 'Experience', 'Personal_Loan');
```


Distribution of target for target=0
Density
Experience
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
10
20
30
40


Distribution of target for target=1
Density
Experience
0.025
0.020
0.015
0.010
0.005
0.000
0
10
20
30
40


Boxplot w.r.t target
Experience
40
30
20
10


Boxplot (without outliers) w.r.t target
Experience
40
30
20
10

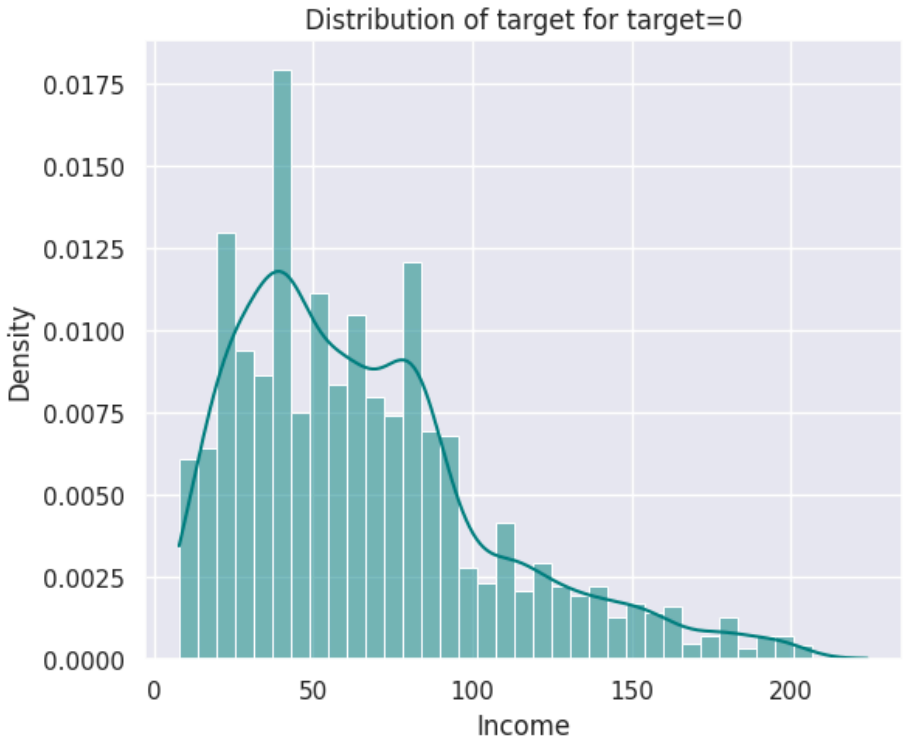
Observations

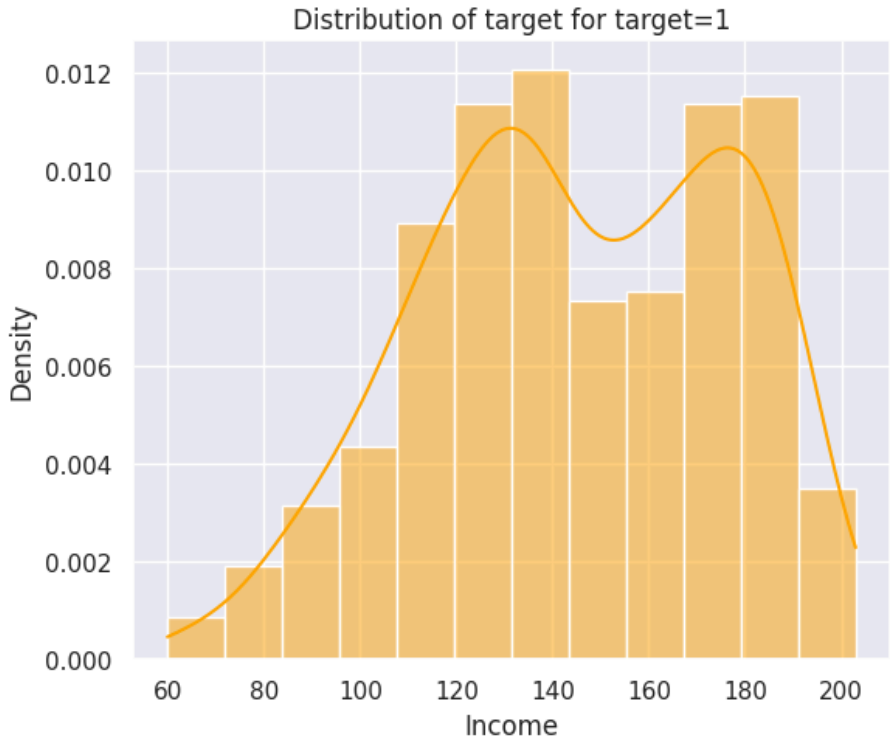
- Experience vs personal loan data is normally distributed with no outliers.
- There is a drop of in personal loans as the customers have more experience.

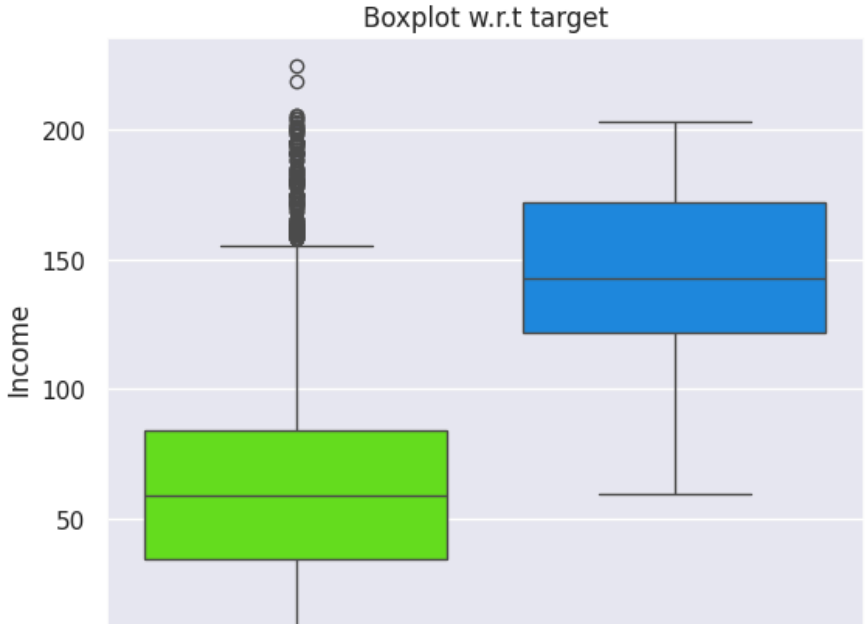
Personal Loan vs Income

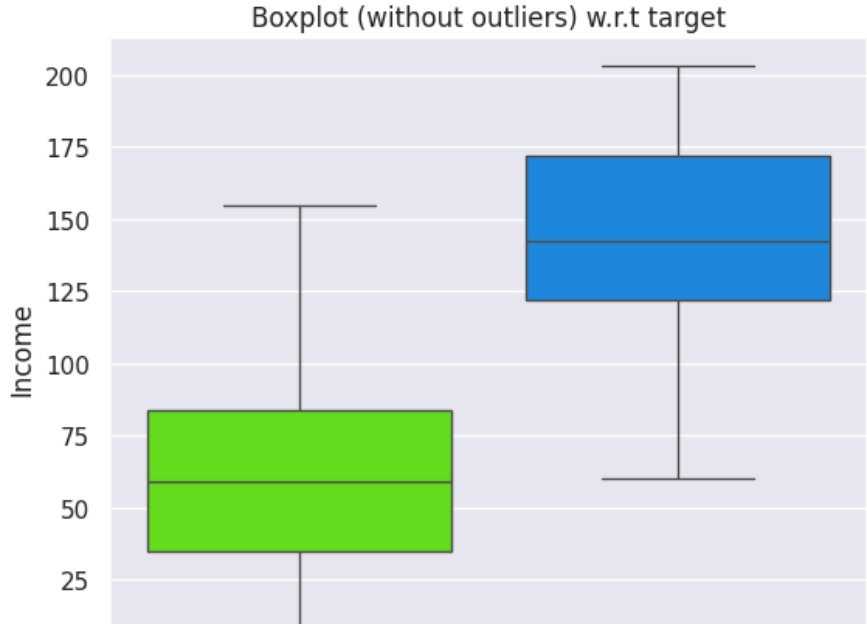
In [246…

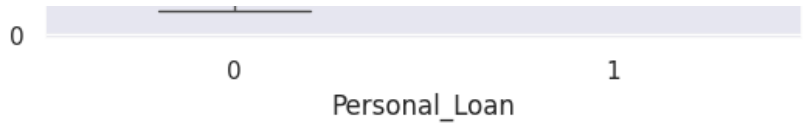
```
distribution_plot_wrt_target(data, 'Income', 'Personal_Loan');
```


Distribution of target for target=0
Density
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
50
100
150
200
Income


Distribution of target for target=1
Density
0.012
0.010
0.008
0.006
0.004
0.002
0.000
60
80
100
120
140
160
180
200
Income


Boxplot w.r.t target
Income
200
150
100
50


Boxplot (without outliers) w.r.t target
Income
200
175
150
125
100
75
50
25



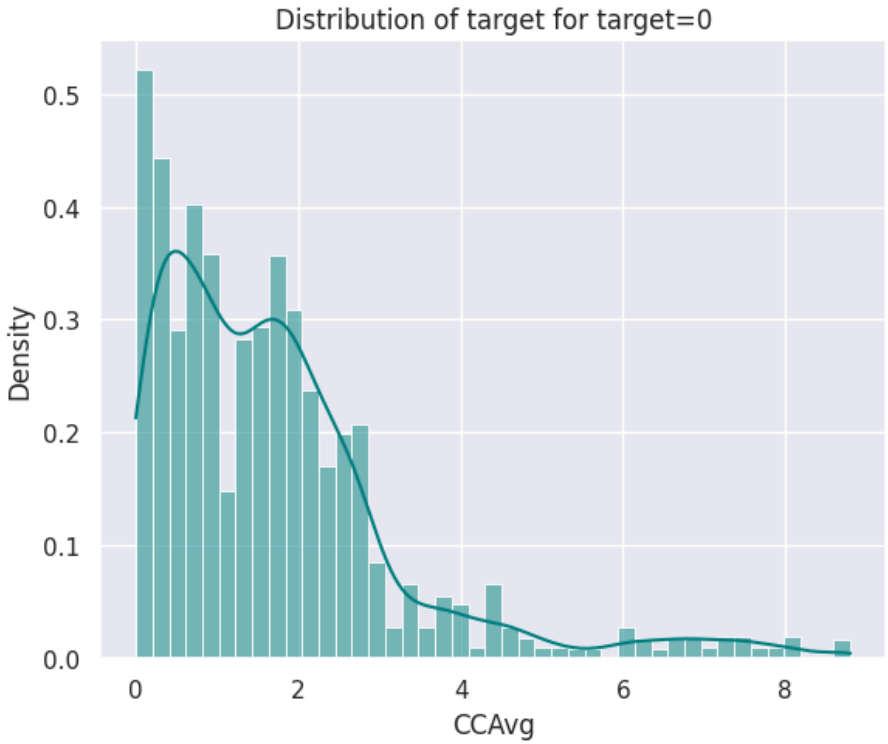
Observations

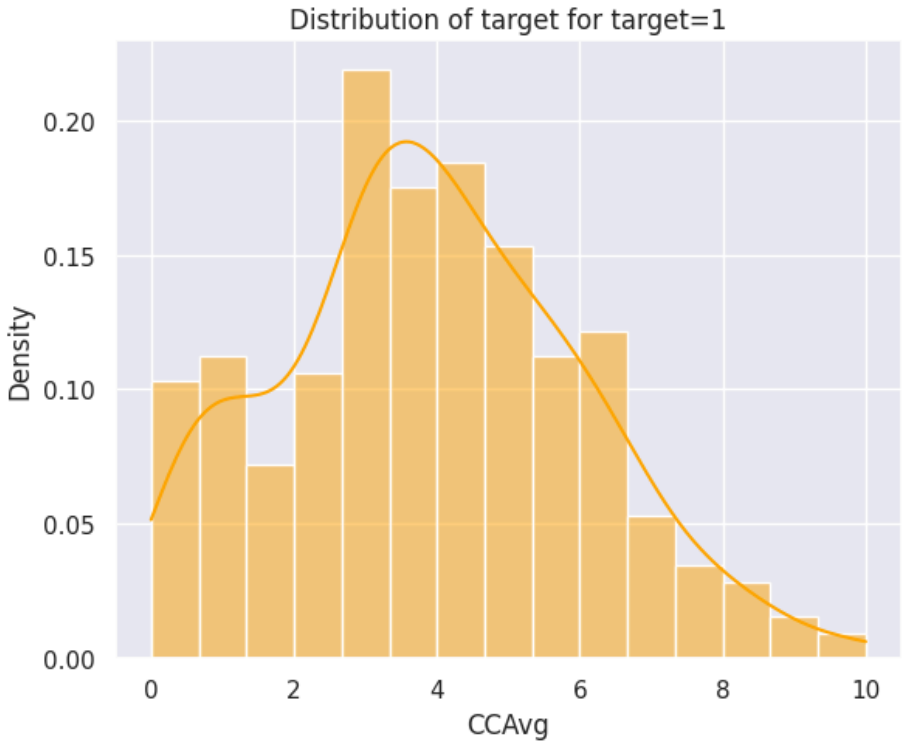
- The income vs. personal loan data is right-skewed for customers without personal loans.
- The income vs. personal loan data is slightly left-skewed for customers with personal loans.
- There are many outliers above the 3rd quartile.

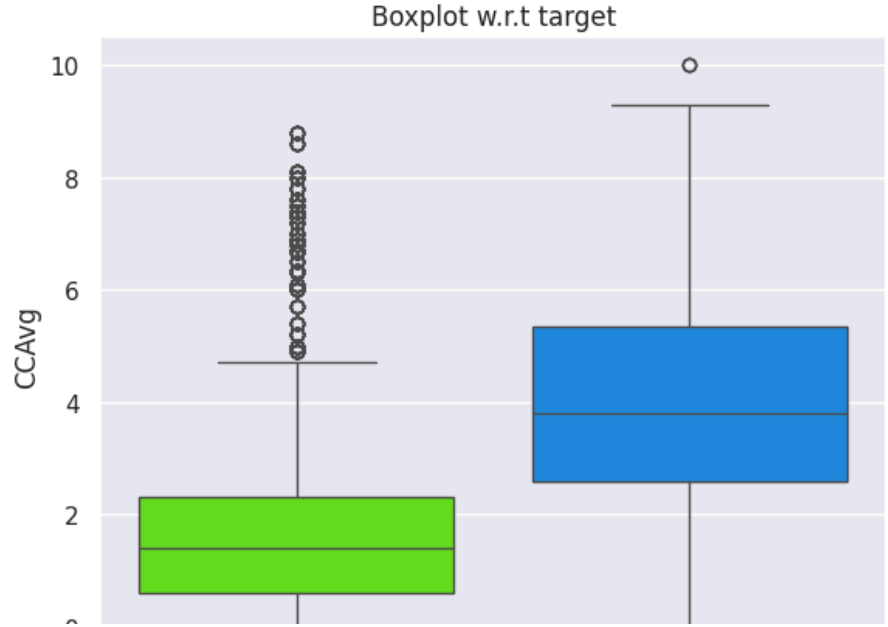
Personal Loan vs CCAvg

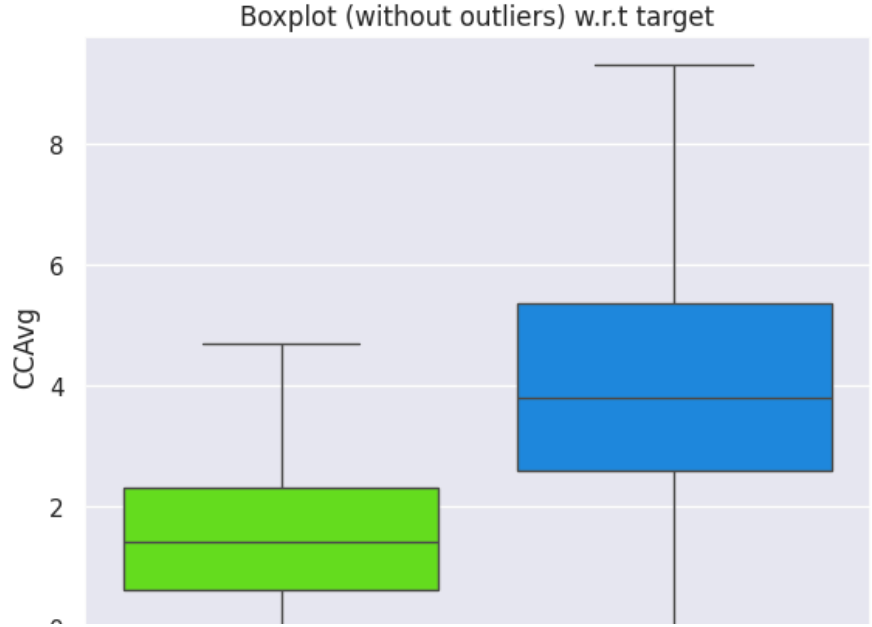
In [247…

```
distribution_plot_wrt_target(data, 'CCAvg', 'Personal_Loan');
```


Distribution of target for target=0
Density
CCAvg
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8


Distribution of target for target=1
Density
CCAvg
0.00
0.05
0.10
0.15
0.20
0
2
4
6
8
10


Boxplot w.r.t target
CCAvg
2
4
6
8
10


Boxplot (without outliers) w.r.t target
CCAvg
2
4
6
8

Observations

- The CCCavg vs personal loan data is right-skewed.
- There are many outliers above the 3rd quartile.

Questions:

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

- Heavily right-skewed with a large number of outliers above the 3rd quartile

2. How many customers have credit cards?

- 3,500

3. What are the attributes that have a strong correlation with the target attribute (personal loan)?

- customers with education level 3 are about three times more likely to have a personal loan than those with level 1.

4. How does a customer's interest in purchasing a loan vary with age?

- Most are between 35 and 55

5. How does a customer's interest in purchasing a loan vary with their education?

- Customers with education level 3 are about three times more likely to have a personal loan than those with level 1.

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)

Model Building

Model Evaluation Criterion

Outlier Detection

In [248…
```
# Find the 25th percentile and 75th percentile.
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)

# Inter Quantile Range (75th percentile - 25th percentile)
IQR = Q3 - Q1

# Finding lower and upper bounds for all values. All values outside these bounds are outliers
lower = (
 Q1 - 1.5 * IQR
)
upper = Q3 + 1.5 * IQR
```

In [249…
```
# determine which values are below the lower bound
# and above the upper bound and convert this into a percentage
(
    (data.select_dtypes(include=["float64", "int64"]) < lower)
    | (data.select_dtypes(include=["float64", "int64"]) > upper)
).sum() / len(data) * 100
```

Out[249…
```
Age           0.00
Experience    0.00
Income        1.92
Family        0.00
CCAvg         6.48
Mortgage      5.82
dtype: float64
```

- Age and experience are highly correlated, so we can drop experience.
- Income, CCAvg, and Mortgage have a significant number of values above the 3rd quartile considered outliers. However, these values are legitimate and do not require outlier treatment.

Data preperation for Modeling

The heatmap shows that Age and Experience are very closely related (0.99), so Experience can be dropped

In [250…
```
# Dropping Experience as it is closely correlated with Age
# Dropping Personal_Loan as it is the target variable we are predicting
X = data.drop(["Personal_Loan", "Experience"], axis=1)
Y = data["Personal_Loan"]

# One hot encoding categorical variables
X = pd.get_dummies(X, columns=["ZIPCode", "Education"], drop_first=True)
```

In [251…
```
# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.30, random_state=1
)
```

In [252…
```
# Display data ab out the test and training sets
print("Shape of Training set : ", X_train.shape)
print("Shape of test set : ", X_test.shape)
print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
print("Percentage of classes in test set:")
print(y_test.value_counts(normalize=True))
```

```
Shape of Training set :  (3500, 17)
Shape of test set :  (1500, 17)
Percentage of classes in training set:
0    0.905429
1    0.094571
Name: Personal_Loan, dtype: float64
Percentage of classes in test set:
0    0.900667
1    0.099333
Name: Personal_Loan, dtype: float64
```

- The class distribution is very similar between the training and test sets, which is good. It indicates that the split was likely done with stratification, preserving the overall class distribution in both sets.
- The dataset is imbalanced, with the majority class (0) representing about 90% of the data and the minority class (1) only about 10%. This imbalance can affect the performance of the decision tree, potentially biasing it towards the majority class.

Model Building

In [253…

```
# Define a function to compute different metrics for evaluating the performance of a classification model built using sklearn
def model_performance_classification_sklearn(model, predictors, target):
    """
    Compute various metrics to evaluate classification model performance.

    Parameters:
    model (sklearn classifier): The trained classification model
    predictors (array-like): Independent variables used for prediction
    target (array-like): True labels for the target variable

    Returns:
    pandas.DataFrame: A single-row dataframe containing Accuracy, Recall, Precision, and F1 score
    """
    # Make predictions using the independent variables
    pred = model.predict(predictors)

    # Compute Accuracy: the proportion of correct predictions (both true positives and true negatives) among the total number
    acc = accuracy_score(target, pred)

    # Compute Recall: the proportion of actual positive cases that were correctly identified
```

```
    recall = recall_score(target, pred)

    # Compute Precision: the proportion of predicted positive cases that are actually positive
    precision = precision_score(target, pred)

    # Compute F1-score: the harmonic mean of precision and recall, providing a single score that balances both metrics
    f1 = f1_score(target, pred)

    # Create a dataframe to store and return all computed metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    # Return the dataframe of performance metrics
    return df_perf
```

In [254…

```
# Define a function to create and display a confusion matrix
def confusion_matrix_sklearn(model, predictors, target):
    """
    Plot the confusion matrix with percentages for a classification model.

    Parameters:
    model: classifier object
    predictors: independent variables (features)
    target: dependent variable (true labels)
    """
    # Generate predictions using the model
    y_pred = model.predict(predictors)

    # Compute the confusion matrix
    cm = confusion_matrix(target, y_pred)

    # Create labels for the confusion matrix
    # Each cell will display 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)

    # Set up the matplotlib figure
    plt.figure(figsize=(6, 4))

    # Create a heatmap of the confusion matrix
    # annot=labels adds the custom labels to each cell
    sns.heatmap(cm, annot=labels, fmt="")

    # Set labels for the axes
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Build Decision Tree Model

In [255…

```
# Building a decision tree model
model = DecisionTreeClassifier(criterion="gini", random_state=1)
model.fit(X_train, y_train)
```

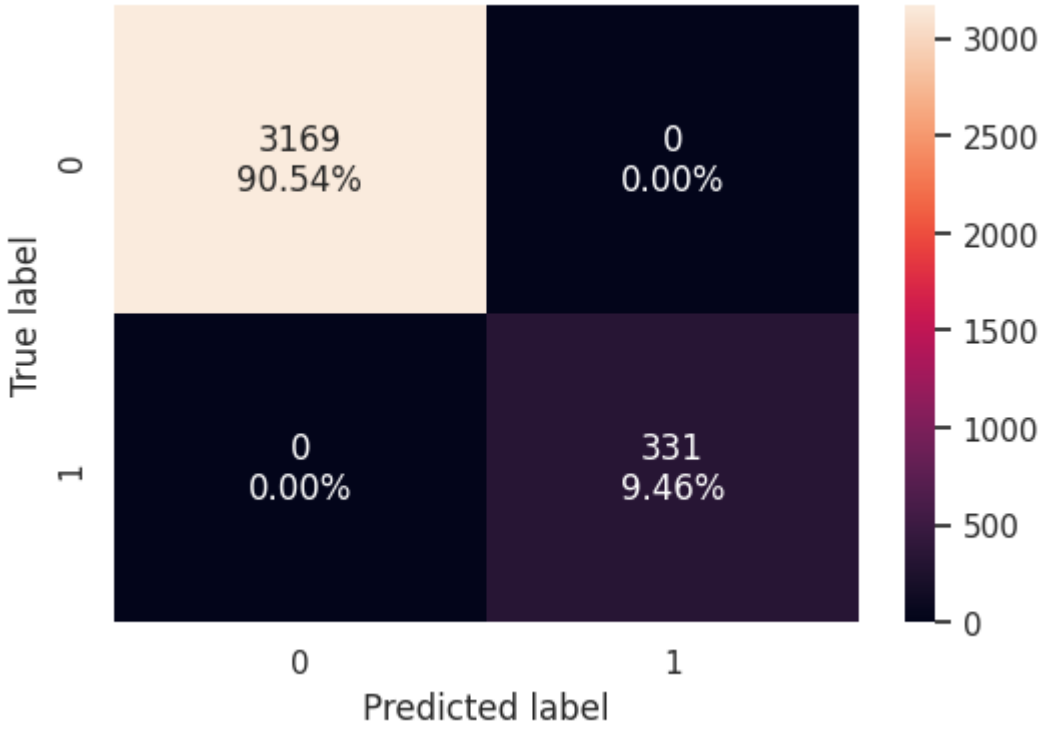
Out[255…

```
DecisionTreeClassifier(random_state=1)
```

Checking model performance on training data

In [256…

```
# Compute and display the confusion matrix for the training set
confusion_matrix_sklearn(model, X_train, y_train)
```



In [257…

```
# Compute and display the performance metrics for the training set
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, y_train
)
decision_tree_perf_train
```

Out[257…

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

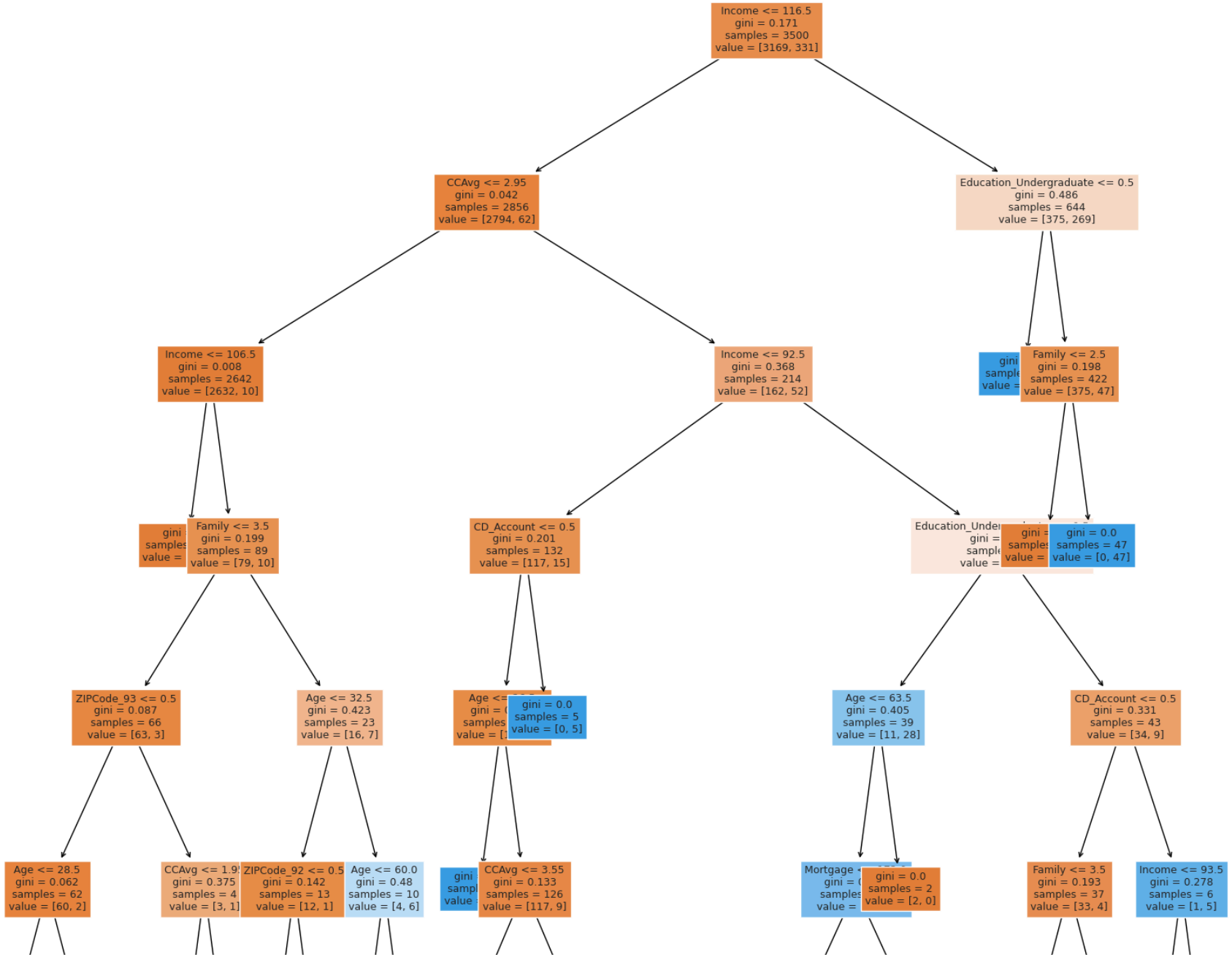
- The classification model has achieved perfect performance metrics across the board, with an Accuracy, Recall, Precision, and F1-score all equal to 1.0. This indicates that the model correctly classified all instances, with no false positives or false negatives.
- While this is an ideal result, it is important to consider whether this performance is realistic or if it might indicate overfitting, especially if the model was evaluated on the training data or a non-representative test set

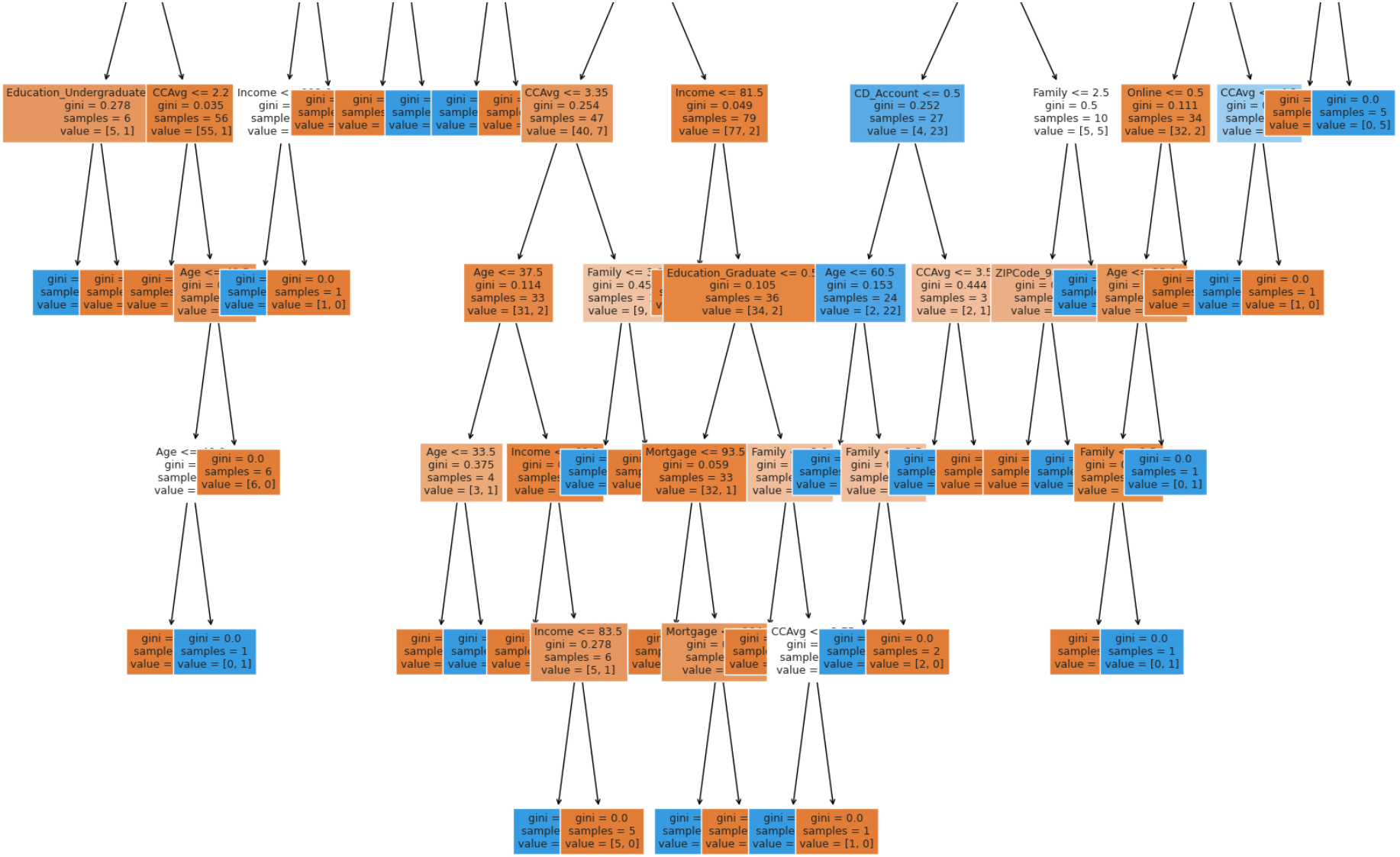
Visualizing the Decision Tree for the training data

In [258…
```
# Extract and print feature names from the training dataset
feature_names = list(X_train.columns)
print(feature_names)
```

```
['Age', 'Income', 'Family', 'CCAvg', 'Mortgage', 'Securities_Account', 'CD_Account', 'Online', 'CreditCard', 'ZIPCode_91', 'ZIP
Code_92', 'ZIPCode_93', 'ZIPCode_94', 'ZIPCode_95', 'ZIPCode_96', 'Education_Graduate', 'Education_Undergraduate']
```

In [259…
```
# Display the Decision Tree for this training data
plt.figure(figsize=(20, 30))
out = tree.plot_tree(
    model,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# Add arrows to the decision tree split if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```


Income <= 116.5
gini = 0.171
samples = 3500
value = [3169, 331]
CCAvg <= 2.95
gini = 0.042
samples = 2856
value = [2794, 62]
Education_Undergraduate <= 0.5
gini = 0.486
samples = 644
value = [375, 269]
Income <= 106.5
gini = 0.008
samples = 2642
value = [2632, 10]
Income <= 92.5
gini = 0.368
samples = 214
value = [162, 52]
Family <= 2.5
gini = 0.198
samples = 422
value = [375, 47]
Family <= 3.5
gini = 0.199
samples = 89
value = [79, 10]
CD_Account <= 0.5
gini = 0.201
samples = 132
value = [117, 15]
gini = 0.0
samples = 47
value = [0, 47]
ZIPCode_93 <= 0.5
gini = 0.087
samples = 66
value = [63, 3]
Age <= 32.5
gini = 0.423
samples = 23
value = [16, 7]
gini = 0.0
samples = 5
value = [0, 5]
Age <= 63.5
gini = 0.405
samples = 39
value = [11, 28]
CD_Account <= 0.5
gini = 0.331
samples = 43
value = [34, 9]
Age <= 28.5
gini = 0.062
samples = 62
value = [60, 2]
ZIPCode_92 <= 0.5
gini = 0.142
samples = 13
value = [12, 1]
Age <= 60.0
gini = 0.48
samples = 10
value = [4, 6]
CCAvg <= 3.55
gini = 0.133
samples = 126
value = [117, 9]
gini = 0.0
samples = 2
value = [2, 0]
Family <= 3.5
gini = 0.193
samples = 37
value = [33, 4]
Income <= 93.5
gini = 0.278
samples = 6
value = [1, 5]



In [260…

```
# Text report showing the rules of a decision tree -
print(tree.export_text(model, feature_names=feature_names, 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
|   |   |   |   |--- ZIPCode_93 <= 0.50
|   |   |   |   |   |--- Age <= 28.50
|   |   |   |   |   |   |--- Education_Undergraduate <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- Education_Undergraduate >  0.50
|   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |--- Age >  28.50
|   |   |   |   |   |   |--- CCAvg <= 2.20
|   |   |   |   |   |   |   |--- weights: [48.00, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  2.20
|   |   |   |   |   |   |   |--- Age <= 48.50
|   |   |   |   |   |   |   |   |--- Age <= 40.00
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Age >  40.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- Age >  48.50
|   |   |   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |   |   |--- ZIPCode_93 >  0.50
|   |   |   |   |   |   |--- CCAvg <= 1.95
|   |   |   |   |   |   |   |--- Income <= 112.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- Income >  112.00
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  1.95
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |--- Family >  3.50
|   |   |   |   |   |--- Age <= 32.50
|   |   |   |   |   |   |--- ZIPCode_92 <= 0.50
|   |   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |   |   |--- ZIPCode_92 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- Age >  32.50
|   |   |   |   |   |   |--- Age <= 60.00
|   |   |   |   |   |   |   |--- 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
|   |   |   |   |   |   |   |--- Age <= 37.50
|   |   |   |   |   |   |   |   |--- Age <= 33.50
|   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Age >  33.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- Age >  37.50
|   |   |   |   |   |   |   |   |--- 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
|   |   |   |   |   |   |   |   |--- Family <= 2.00
```

```
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Family >  2.00
|   |   |   |   |   |   |   |   |   |--- CCAvg <= 3.75
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- CCAvg >  3.75
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- CD_Account >  0.50
|   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |--- Income >  92.50
|   |   |   |   |--- Education_Undergraduate <= 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
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- CCAvg >  3.50
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- Mortgage >  172.00
|   |   |   |   |   |   |   |--- Family <= 2.50
|   |   |   |   |   |   |   |   |--- ZIPCode_94 <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- ZIPCode_94 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- Family >  2.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |--- Age >  63.50
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |--- Education_Undergraduate >  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
|   |   |   |   |   |   |--- CCAvg <= 4.20
|   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- CCAvg >  4.20
|   |   |   |   |   |   |   |--- 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_Undergraduate <= 0.50
|   |   |--- weights: [0.00, 222.00] class: 1
|   |--- Education_Undergraduate >  0.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [375.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [0.00, 47.00] class: 1
```

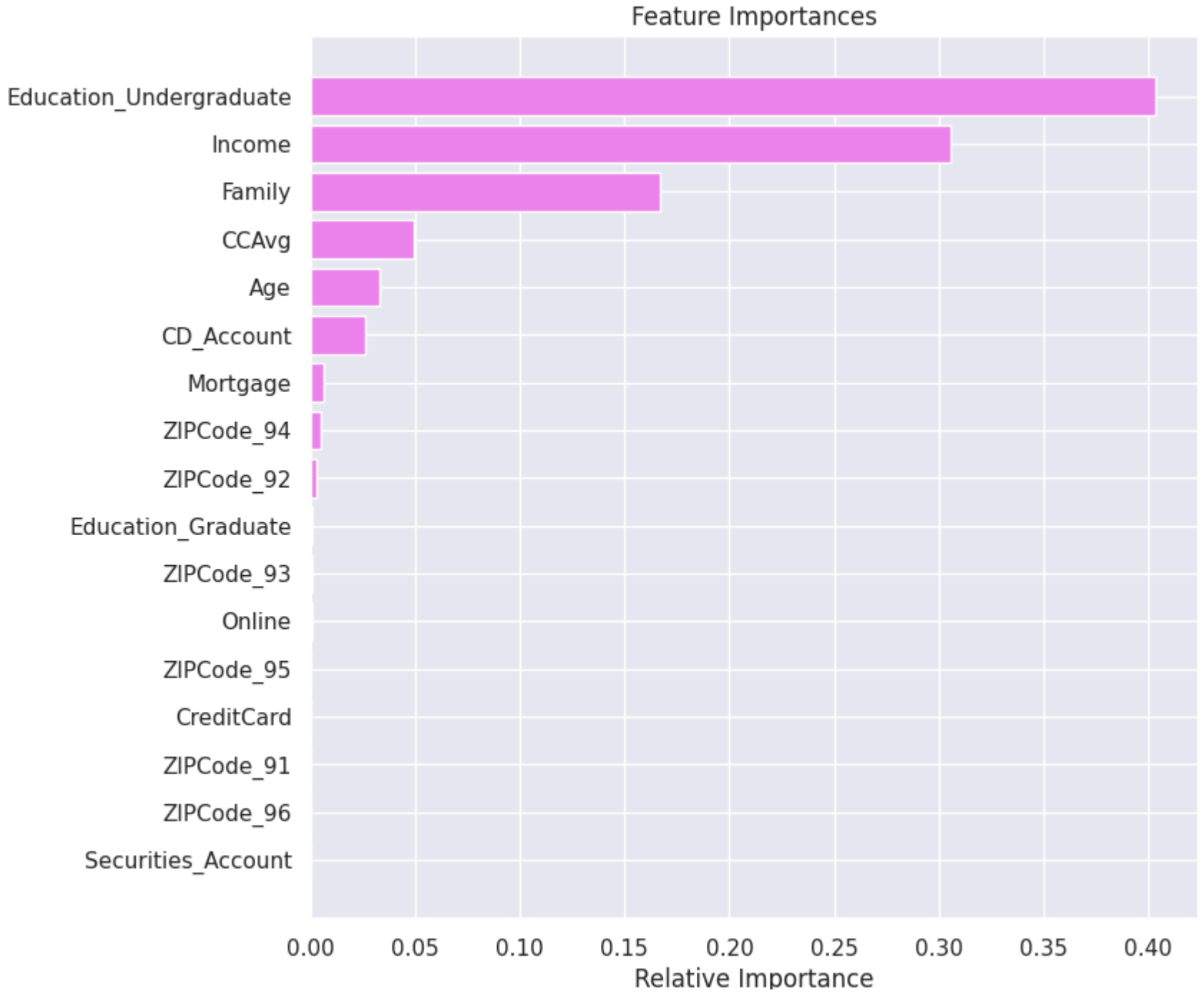
In [261…

```python
# Create and print a DataFrame of feature importances
# 1. Extract feature importances from the model
# 2. Create a DataFrame with importances and feature names computed as the  Gini importance
# 3. Display the result
print(
    pd.DataFrame(
        model.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                            Imp
Education_Undergraduate  0.403732
Income                   0.305688
Family                   0.166733
CCAvg                    0.049492
Age                      0.032549
CD_Account               0.025711
Mortgage                 0.006250
ZIPCode_94               0.004767
ZIPCode_92               0.003080
Education_Graduate       0.000843
ZIPCode_93               0.000594
Online                   0.000561
Securities_Account       0.000000
CreditCard               0.000000
ZIPCode_91               0.000000
ZIPCode_96               0.000000
ZIPCode_95               0.000000
```

In [262…

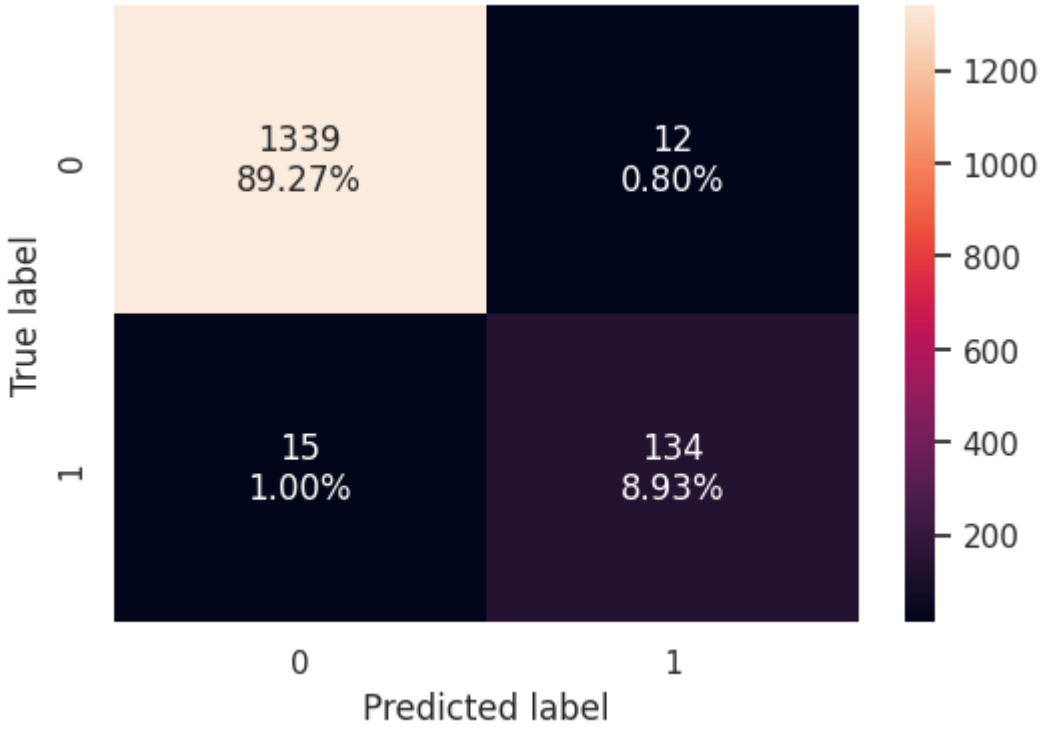
```python
# Visualize feature importances of the model
importances = model.feature_importances_
indices = np.argsort(importances)

# Create a bar plot of feature importances
plt.figure(figsize=(8, 8))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")

# Customize y-axis with feature names
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])

# Display the plot
plt.xlabel("Relative Importance")
plt.show()
```


Feature Importances
Education_Undergraduate
Income
Family
CCAvg
Age
CD_Account
Mortgage
ZIPCode_94
ZIPCode_92
Education_Graduate
ZIPCode_93
Online
ZIPCode_95
CreditCard
ZIPCode_91
ZIPCode_96
Securities_Account
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Relative Importance

- The most important feature is Income with an importance of 0.31)
- The other important features in order of importance, are Family, Education2, Education 3, CCAvg, age and CCAvg.

Checking model performance on test data

In [263…
```
# Compute and display the confusion matrix for the test set
confusion_matrix_sklearn(model, X_test, y_test)
```



In [264…
```
# Compute the performance metrics for the test set
decision_tree_perf_test = model_performance_classification_sklearn(
    model, X_test, y_test
)
```

```
# Display the performance metrics
decision_tree_perf_test
```

Out[264…

	Accuracy	Recall	Precision	F1
0	0.982	0.899329	0.917808	0.908475

Model Performance Improvement

Pre-pruning

In [265…

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

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(6, 15),
    "min_samples_leaf": [1, 2, 5, 7, 10],
    "max_leaf_nodes": [2, 3, 5, 10],
}

# Type of scoring used to compare parameter combinations
acc_scorer = make_scorer(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[265…

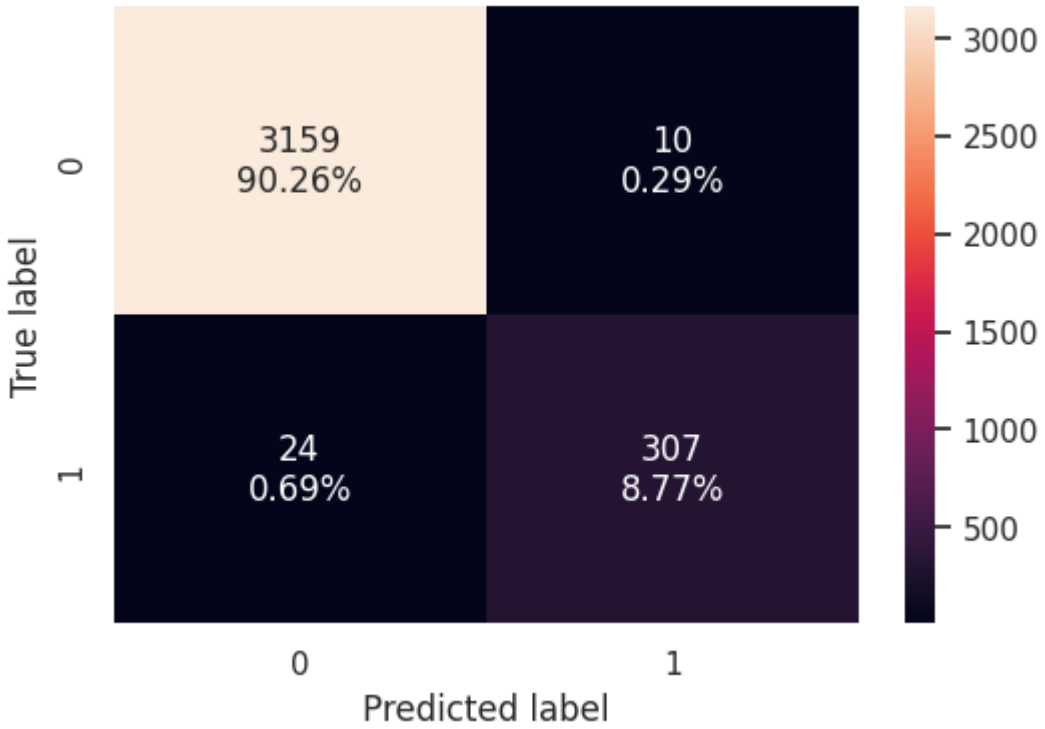
DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=6, max_leaf_nodes=10, random_state=1)
```

Checking performance on the training data

In [266…

```
# Compute and display the confusion matrix for the training set
confusion_matrix_sklearn(estimator, X_train, y_train)
```



In [267…

```
# Evaluate the performance of the tuned decision tree on the training data
decision_tree_tune_perf_train = model_performance_classification_sklearn(estimator, X_train, y_train)

# Display the performance metrics
decision_tree_tune_perf_train
```

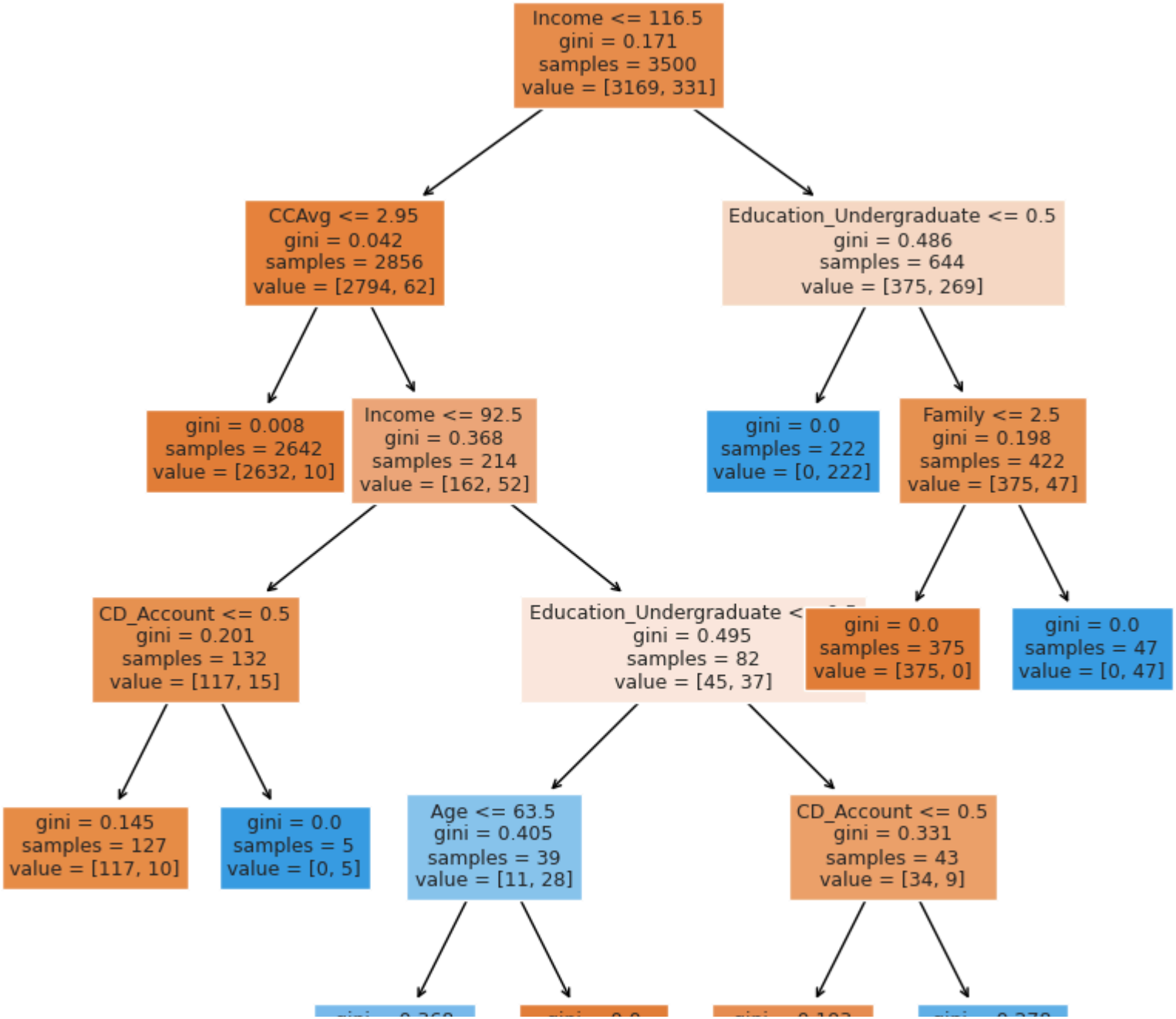
Out[267…

	Accuracy	Recall	Precision	F1
0	0.990286	0.927492	0.968454	0.947531

Visualizing the Decision Tree

In [268…

```
# Display the Decision Tree for this training data
plt.figure(figsize=(10, 10))
out = tree.plot_tree(
    estimator,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# Add arrows to the decision tree split if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```


Income <= 116.5
gini = 0.171
samples = 3500
value = [3169, 331]
CCAvg <= 2.95
gini = 0.042
samples = 2856
value = [2794, 62]
Education_Undergraduate <= 0.5
gini = 0.486
samples = 644
value = [375, 269]
gini = 0.008
samples = 2642
value = [2632, 10]
Income <= 92.5
gini = 0.368
samples = 214
value = [162, 52]
gini = 0.0
samples = 222
value = [0, 222]
Family <= 2.5
gini = 0.198
samples = 422
value = [375, 47]
CD_Account <= 0.5
gini = 0.201
samples = 132
value = [117, 15]
Education_Undergraduate <
gini = 0.495
samples = 82
value = [45, 37]
gini = 0.0
samples = 375
value = [375, 0]
gini = 0.0
samples = 47
value = [0, 47]
gini = 0.145
samples = 127
value = [117, 10]
gini = 0.0
samples = 5
value = [0, 5]
Age <= 63.5
gini = 0.405
samples = 39
value = [11, 28]
CD_Account <= 0.5
gini = 0.331
samples = 43
value = [34, 9]

In [269…

```
# Text report showing the rules of a decision tree -
print(tree.export_text(estimator, feature_names=feature_names, show_weights=True))
```

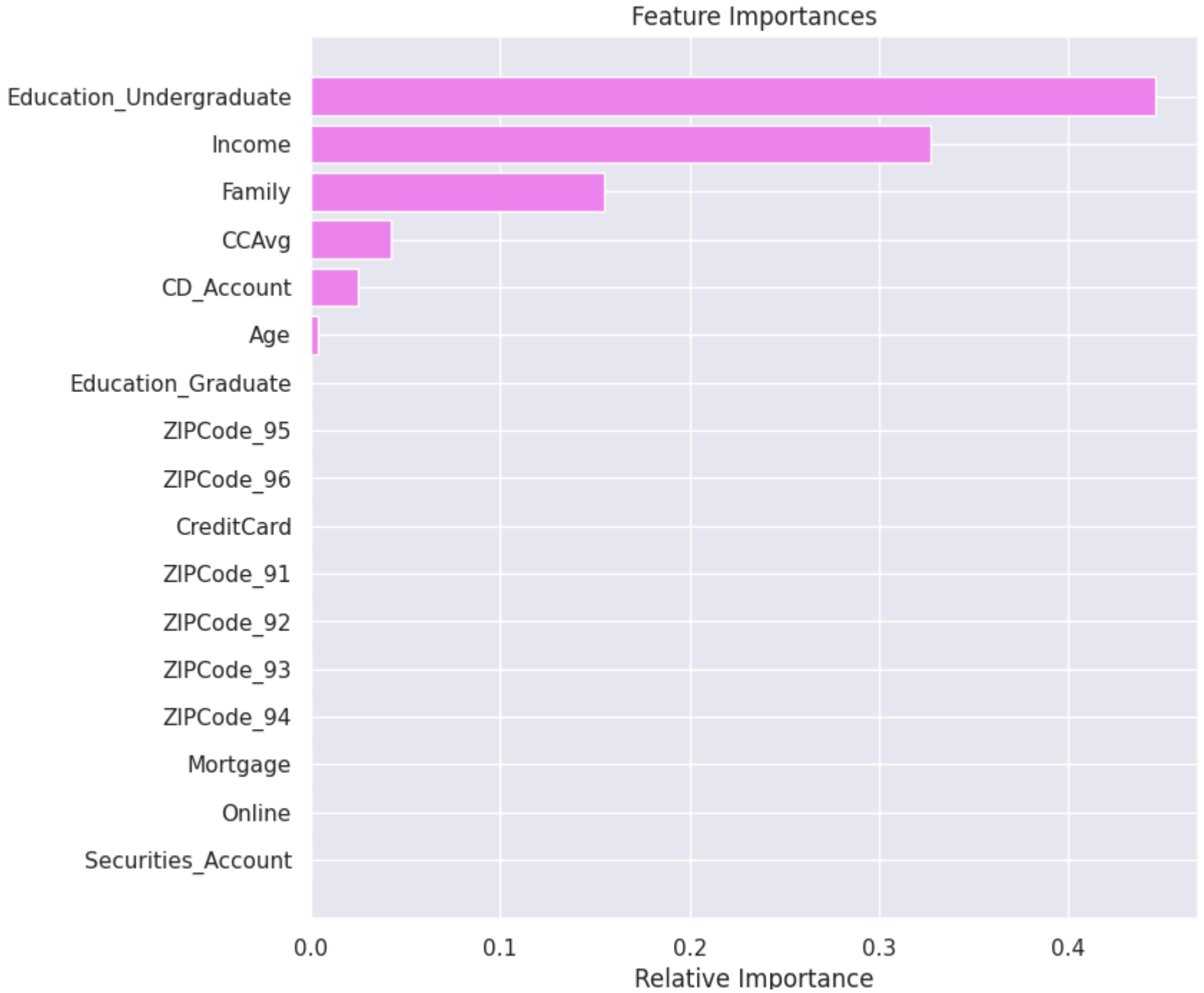
```
|--- Income <= 116.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [2632.00, 10.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- Income <= 92.50
|   |   |   |--- CD_Account <= 0.50
|   |   |   |   |--- weights: [117.00, 10.00] class: 0
|   |   |   |--- CD_Account >  0.50
|   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |--- Income >  92.50
|   |   |   |--- Education_Undergraduate <= 0.50
|   |   |   |   |--- Age <= 63.50
|   |   |   |   |   |--- weights: [9.00, 28.00] class: 1
|   |   |   |   |--- Age >  63.50
|   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |--- Education_Undergraduate >  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_Undergraduate <= 0.50
|   |   |--- weights: [0.00, 222.00] class: 1
|   |--- Education_Undergraduate >  0.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [375.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [0.00, 47.00] class: 1
```

In [270…
```
# Create and print a DataFrame of feature importances
# 1. Extract feature importances from the model
# 2. Create a DataFrame with importances and feature names computed as the  Gini importance
# 3. Display the result
print(
    pd.DataFrame(
        estimator.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                           Imp
Education_Undergraduate  0.446191
Income                   0.327387
Family                   0.155083
CCAvg                    0.042061
CD_Account               0.025243
Age                      0.004035
Mortgage                 0.000000
Online                   0.000000
Securities_Account       0.000000
CreditCard               0.000000
ZIPCode_91               0.000000
ZIPCode_93               0.000000
ZIPCode_92               0.000000
ZIPCode_94               0.000000
ZIPCode_95               0.000000
ZIPCode_96               0.000000
Education_Graduate       0.000000
```

In [271…
```
# Visualize feature importances of the model
importances = estimator.feature_importances_
indices = np.argsort(importances)

# Create a bar plot of feature importances
plt.figure(figsize=(8, 8))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")

# Customize y-axis with feature names
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
```

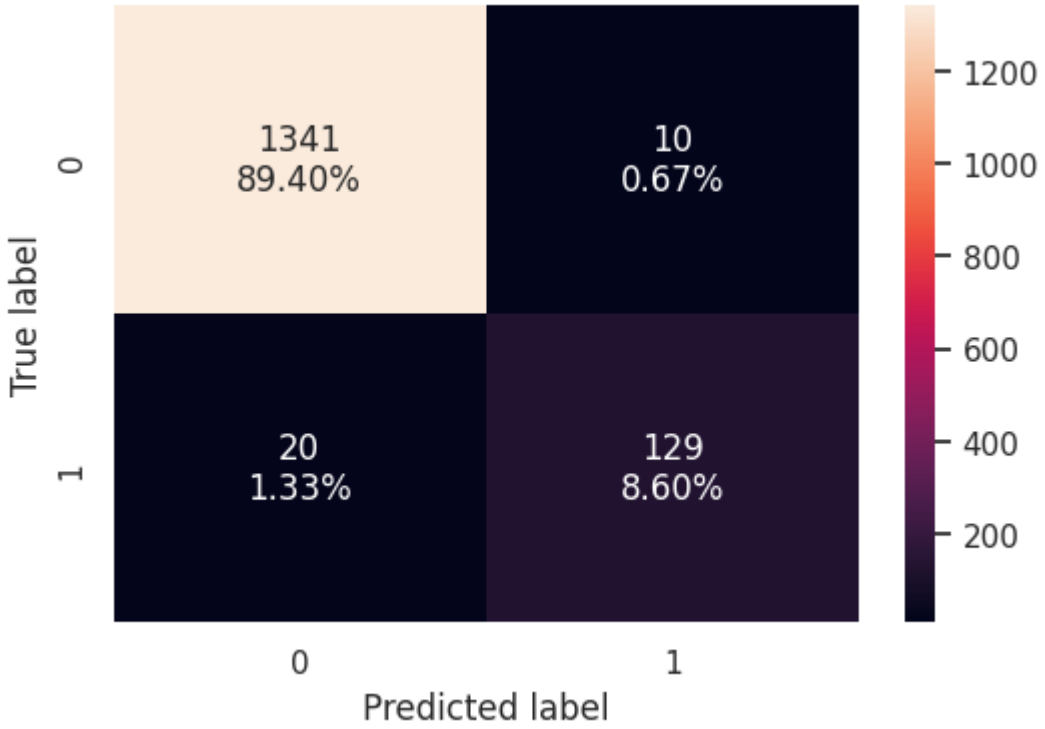
```
# Display the plot
plt.xlabel("Relative Importance")
plt.show()
```


Feature Importances
Education_Undergraduate
Income
Family
CCAvg
CD_Account
Age
Education_Graduate
ZIPCode_95
ZIPCode_96
CreditCard
ZIPCode_91
ZIPCode_92
ZIPCode_93
ZIPCode_94
Mortgage
Online
Securities_Account
0.0
0.1
0.2
0.3
0.4
Relative Importance

- The most important feature is Income with an importance of 0.34)
- The other important features in order of importance are Family, Education2, Education 3, CCAvg and age.

Checking the performance on the test data

In [272…
```
# Compute and display the confusion matrix for the test set
confusion_matrix_sklearn(estimator, X_test, y_test)
```



In [273…
```
# Evaluate the performance of the tuned decision tree on the test data
decision_tree_tune_perf_test = model_performance_classification_sklearn(estimator, X_test, y_test)

# Display the performance metrics
decision_tree_tune_perf_test
```

Out[273...

	Accuracy	Recall	Precision	F1
0	0.98	0.865772	0.928058	0.895833

Cost-complexity Pruning

- Cost complexity pruning helps to find the optimal tree size by balancing the trade-off between tree complexity and its accuracy on the training data.
- The ccp_alpha parameter determines the strength of the pruning.

In [274...

```
# Perform cost complexity pruning on the decision 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 [275...

```
# Display the cost complexity pruning path as a DataFrame
pd.DataFrame(path)
```

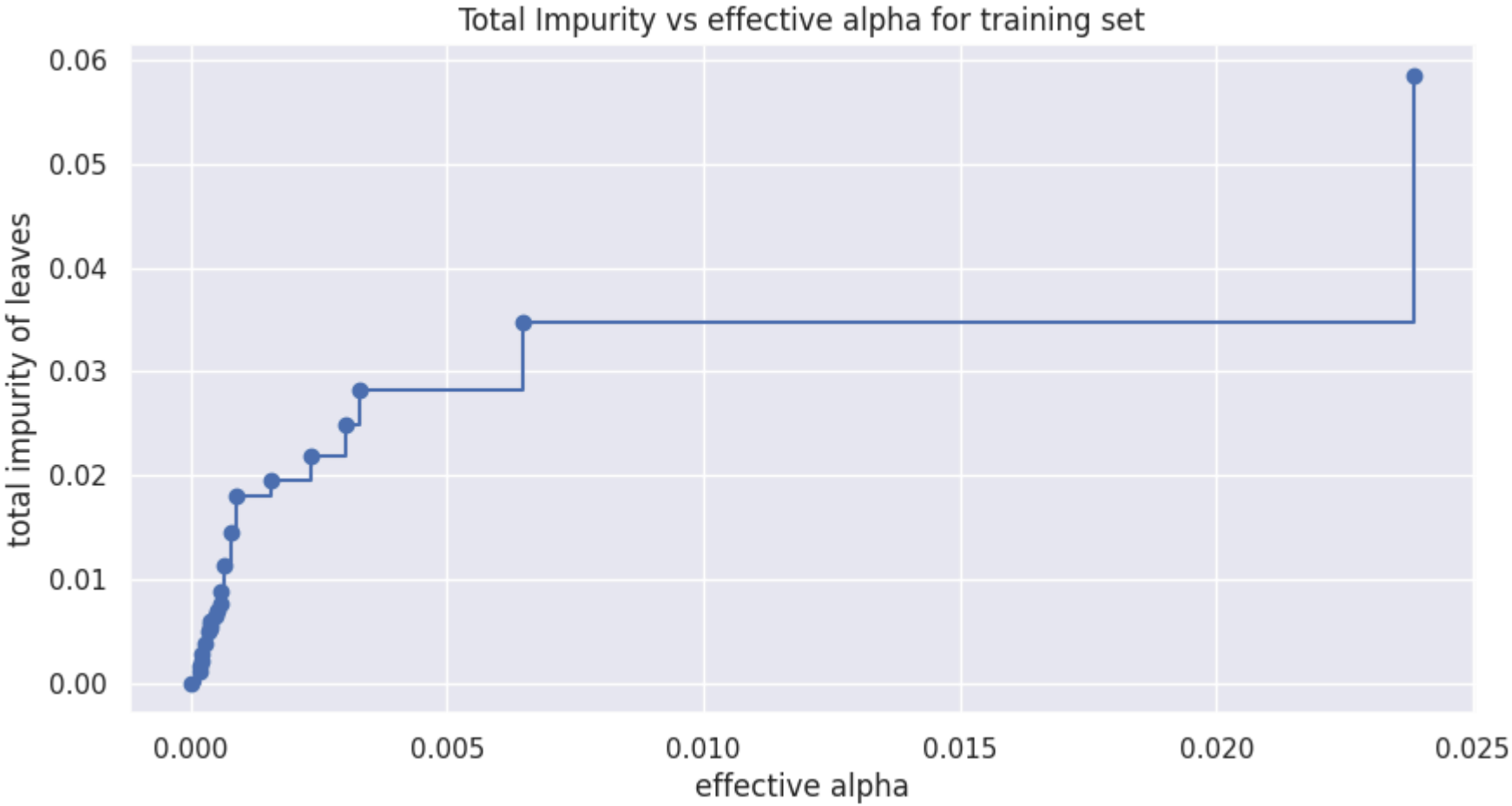
Out[275...

	ccp_alphas	impurities
0	0.000000	0.000000
1	0.000186	0.001114
2	0.000187	0.001675
3	0.000214	0.002104
4	0.000216	0.002750
5	0.000268	0.003824
6	0.000359	0.004900
7	0.000381	0.005280
8	0.000381	0.005661
9	0.000381	0.006042
10	0.000476	0.006519
11	0.000527	0.007046
12	0.000582	0.007628
13	0.000593	0.008813
14	0.000641	0.011379
15	0.000769	0.014456
16	0.000882	0.017985
17	0.001552	0.019536
18	0.002333	0.021869
19	0.003024	0.024893
20	0.003294	0.028187
21	0.006473	0.034659

	ccp_alphas	impurities
22	0.023866	0.058525
23	0.056365	0.171255

In [276…

```
# Create a figure and axis object with specified size
fig, ax = plt.subplots(figsize=(10, 5))

# Plot the relationship between ccp_alphas and impurities
# Use markers for data points and step-style for the line
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")

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



In [277…

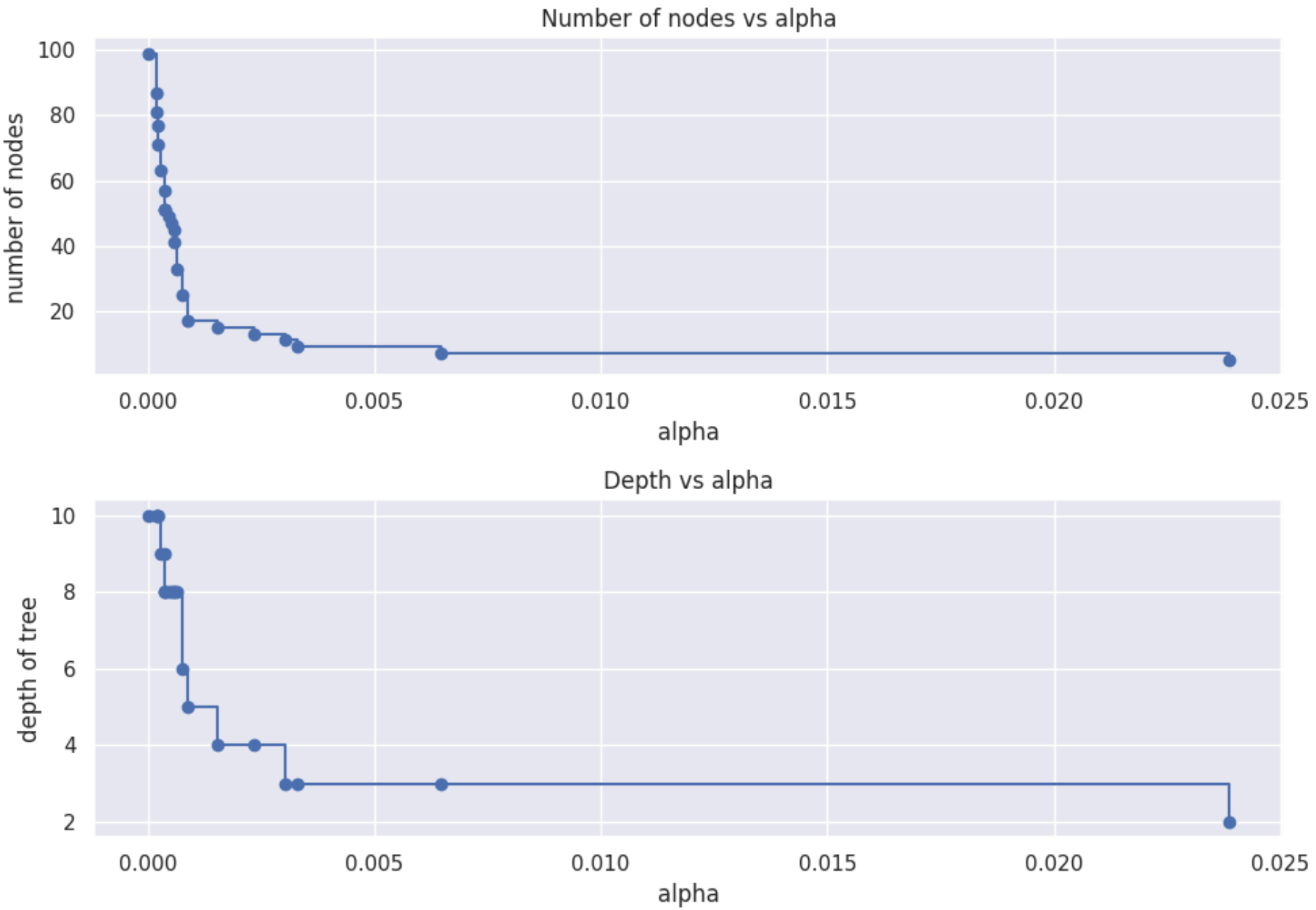
```
# Initialize an empty list to store the decision tree classifiers
clfs = []

# Loop through each value of ccp_alpha
for ccp_alpha in ccp_alphas:
    # Create a DecisionTreeClassifier with the current ccp_alpha and a fixed random_state for reproducibility
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha)

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

```
    # Append the fitted classifier to the list of classifiers
    clfs.append(clf)

# Print the number of nodes in the last decision tree and its corresponding 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: 0.056364969335601575
```

In [278…

```
# Remove the last element from the list of classifiers
clfs = clfs[:-1]

# Remove the last element from the list of ccp_alphas
ccp_alphas = ccp_alphas[:-1]

# Create a list of the number of nodes for each classifier in clfs
node_counts = [clf.tree_.node_count for clf in clfs]

# Create a list of the maximum depths for each classifier in clfs
depth = [clf.tree_.max_depth for clf in clfs]

# Create a figure with two subplots (one for number of nodes vs alpha and one for depth vs alpha)
fig, ax = plt.subplots(2, 1, figsize=(10, 7))

# Plot the number of nodes vs alpha in 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 the tree vs alpha in 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 figure to prevent overlap
fig.tight_layout()
```



Recall vs alpha for training and testing sets

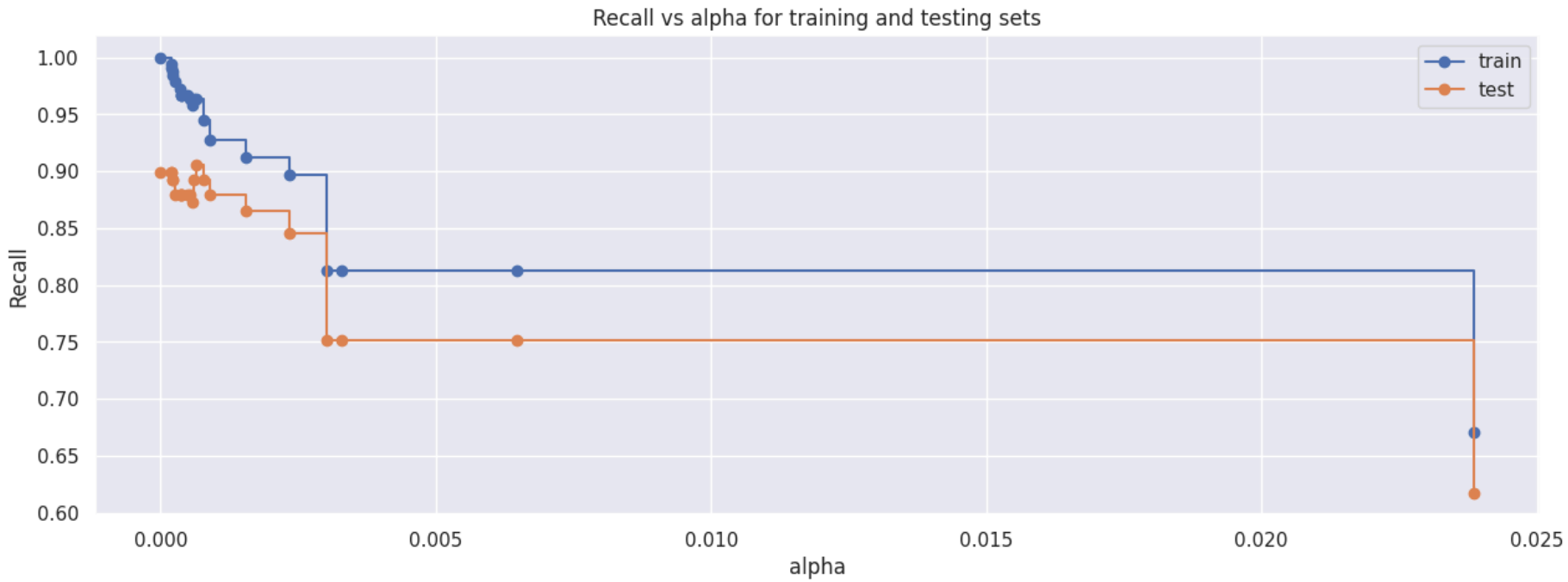
In [279…

```
# Initialize an empty list to store recall scores for the training data
recall_train = []

# Loop through each classifier in the list of classifiers
for clf in clfs:
    # Predict the training data using the current classifier
    pred_train = clf.predict(X_train)

    # Calculate the recall score for the training data predictions
    values_train = recall_score(y_train, pred_train)

    # Append the recall score to the recall_train list
    recall_train.append(values_train)

# Initialize an empty list to store recall scores for the test data
recall_test = []

# Loop through each classifier in the list of classifiers
for clf in clfs:
    # Predict the test data using the current classifier
    pred_test = clf.predict(X_test)

    # Calculate the recall score for the test data predictions
    values_test = recall_score(y_test, pred_test)

    # Append the recall score to the recall_test list
    recall_test.append(values_test)
```

In [280…

```
# Create a figure and axis object with specified size (15 inches wide, 5 inches tall)
fig, ax = plt.subplots(figsize=(15, 5))

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

# Set the y-axis label
ax.set_ylabel("Recall")

# Set the title of the plot
ax.set_title("Recall vs alpha for training and testing sets")
```

```
# Plot the recall scores for the training set
# Use blue circles as markers and a step-style line
ax.plot(ccp_alphas, recall_train, marker="o", label="train", drawstyle="steps-post")

# Plot the recall scores for the test set
# Use orange circles as markers and a step-style line
ax.plot(ccp_alphas, recall_test, marker="o", label="test", drawstyle="steps-post")

# Add a legend to distinguish between train and test lines
ax.legend()

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



In [281…

```
# Find the index of the classifier with the highest recall score on the test data
index_best_model = np.argmax(recall_test)

# Select the classifier with the highest recall score on the test data
```

```
best_model = clfs[index_best_model]

# Print the details of the best classifier
print("Best model index:", index_best_model)
print("Best model type:", type(best_model))
print("Best model parameters:")
for param, value in best_model.get_params().items():
    print(f"  {param}: {value}")

# Print feature importances
if hasattr(best_model, 'feature_importances_'):
    print("\nFeature Importances:")
    for feature, importance in zip(X_train.columns, best_model.feature_importances_):
        print(f"  {feature}: {importance}")
```

```
Best model index: 14
Best model type: <class 'sklearn.tree._classes.DecisionTreeClassifier'>
Best model parameters:
  ccp_alpha: 0.0006414326414326415
  class_weight: None
  criterion: gini
  max_depth: None
  max_features: None
  max_leaf_nodes: None
  min_impurity_decrease: 0.0
  min_samples_leaf: 1
  min_samples_split: 2
  min_weight_fraction_leaf: 0.0
  random_state: 1
  splitter: best

Feature Importances:
  Age: 0.01716306090970831
  Income: 0.3190125472761435
  Family: 0.16485198876572466
  CCAvg: 0.045185370331316074
  Mortgage: 0.0
  Securities_Account: 0.0
  CD_Account: 0.02429818267912022
  Online: 0.0
  CreditCard: 0.0
  ZIPCode_91: 0.0
  ZIPCode_92: 0.0
  ZIPCode_93: 0.0
  ZIPCode_94: 0.0
  ZIPCode_95: 0.0
  ZIPCode_96: 0.0
  Education_Graduate: 0.0
  Education_Undergraduate: 0.4294888500379874
```

Post Pruning

In [282…

```
# Create a new DecisionTreeClassifier with the ccp_alpha value from the best model identified earlier,
# specified class weights, and a fixed random state for reproducibility
```

```
estimator_2 = DecisionTreeClassifier(
    ccp_alpha=best_model.ccp_alpha,  # Use the ccp_alpha value from the best model
    class_weight={0: 0.15, 1: 0.85},  # Assign class weights to handle class imbalance
    random_state=1  # Set random state for reproducibility
)

# Fit the new classifier to the training data (X_train and y_train)
estimator_2.fit(X_train, y_train)
```

Out[282…

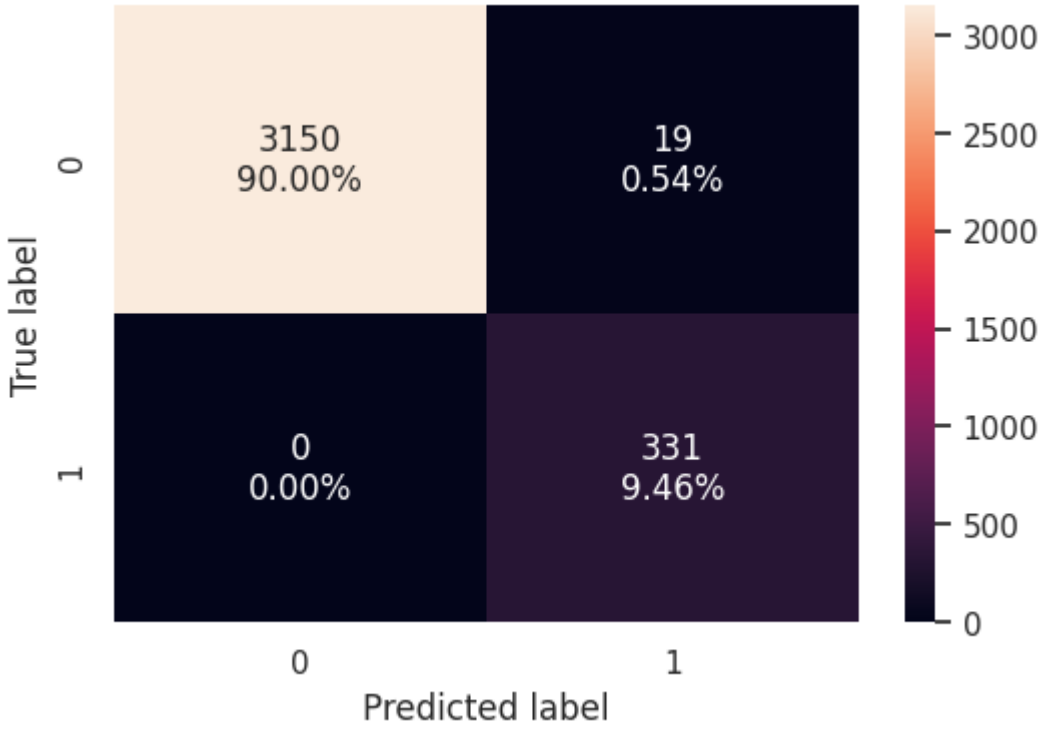
DecisionTreeClassifier

```
DecisionTreeClassifier(ccp_alpha=0.0006414326414326415,
                       class_weight={0: 0.15, 1: 0.85}, random_state=1)
```

Check the performance on training data

In [283…

```
# Compute and display the confusion matrix for the training set
confusion_matrix_sklearn(estimator_2, X_train, y_train)
```



In [284…

```
# Compute and display the performance metrics for the training set
decision_tree_perf_train = model_performance_classification_sklearn(
    estimator_2, X_train, y_train
)
# Display the performance metrics
decision_tree_perf_train
```

Out[284…

	Accuracy	Recall	Precision	F1
0	0.994571	1.0	0.945714	0.9721

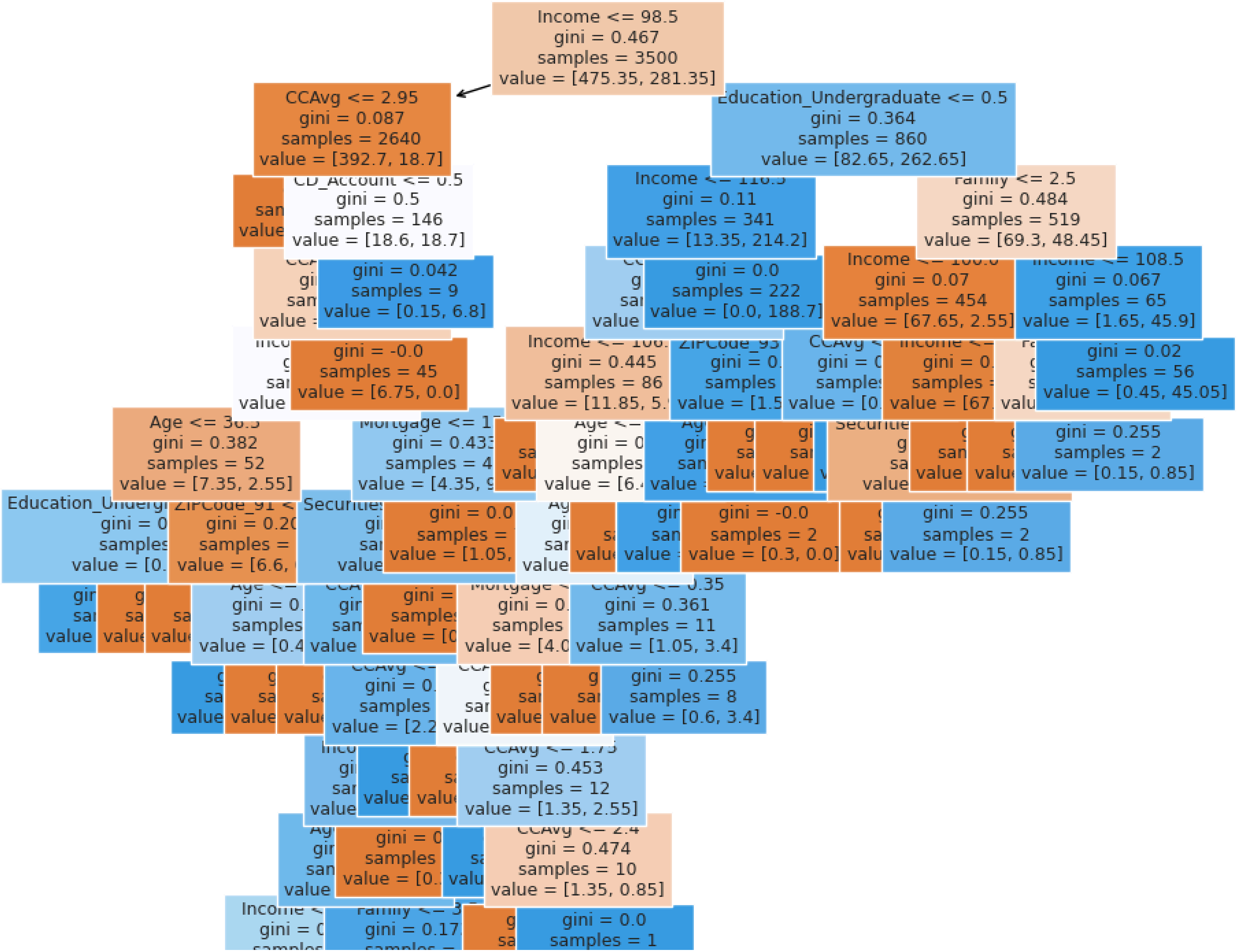
- The classification model has achieved perfect performance metrics across the board, with an Accuracy, Recall, Precision, and F1-score all equal to 1.0. This indicates that the model correctly classified all instances with no false positives or false negatives.

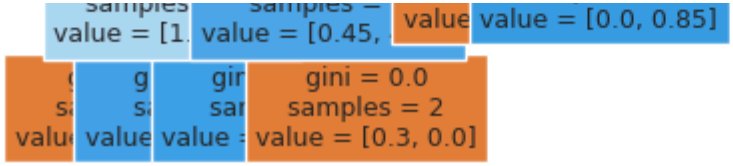
- While this is an ideal result, it is important to consider whether this performance is realistic or if it might indicate overfitting, especially if the model was evaluated on the training data or a non-representative test set

Visualizing the Decision Tree

In [285…

```
# Set up the figure size for the plot
plt.figure(figsize=(10, 10))

# Plot the decision tree using the specified estimator and configuration
out = tree.plot_tree(
    estimator_2,                # The decision tree classifier to plot
    feature_names=feature_names,  # Names of the features used in the decision tree
    filled=True,                # Color the nodes to reflect the majority class
    fontsize=9,                 # Set the font size for the text in the plot
    node_ids=False,             # Do not display node IDs
    class_names=None            # Do not display class names
)

# Loop through the plotted tree elements to 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 the edge color of the arrows to black
        arrow.set_linewidth(1)        # Set the line width of the arrows to 1

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


Income <= 98.5
gini = 0.467
samples = 3500
value = [475.35, 281.35]
CCAvg <= 2.95
gini = 0.087
samples = 2640
value = [392.7, 18.7]
Education_Undergraduate <= 0.5
gini = 0.364
samples = 860
value = [82.65, 262.65]
gini = 0.5
samples = 146
value = [18.6, 18.7]
gini = 0.11
samples = 341
value = [13.35, 214.2]
gini = 0.484
samples = 519
value = [69.3, 48.45]
gini = 0.042
samples = 9
value = [0.15, 6.8]
gini = 0.0
samples = 222
value = [0.0, 188.7]
gini = 0.07
samples = 454
value = [67.65, 2.55]
gini = 0.067
samples = 65
value = [1.65, 45.9]
gini = -0.0
samples = 45
value = [6.75, 0.0]
gini = 0.445
samples = 86
gini = 0.02
samples = 56
value = [0.45, 45.05]
gini = 0.382
samples = 52
value = [7.35, 2.55]
gini = 0.255
samples = 2
value = [0.15, 0.85]
gini = -0.0
samples = 2
value = [0.3, 0.0]
gini = 0.255
samples = 2
value = [0.15, 0.85]
gini = 0.361
samples = 11
value = [1.05, 3.4]
gini = 0.255
samples = 8
value = [0.6, 3.4]
gini = 0.453
samples = 12
value = [1.35, 2.55]
CCAvg <= 2.4
gini = 0.474
samples = 10
value = [1.35, 0.85]
gini = 0.0
samples = 1



In [286…

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

print(tree.export_text(estimator_2, feature_names=feature_names, show_weights=True))
```

```
|--- Income <= 98.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [374.10, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- CD_Account <= 0.50
|   |   |   |--- CCAvg <= 3.95
|   |   |   |   |--- Income <= 81.50
|   |   |   |   |   |--- Age <= 36.50
|   |   |   |   |   |   |--- Education_Undergraduate <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.15, 1.70] class: 1
|   |   |   |   |   |   |--- Education_Undergraduate >  0.50
|   |   |   |   |   |   |   |--- weights: [0.60, 0.00] class: 0
|   |   |   |   |   |--- Age >  36.50
|   |   |   |   |   |   |--- ZIPCode_91 <= 0.50
|   |   |   |   |   |   |   |--- weights: [6.15, 0.00] class: 0
|   |   |   |   |   |   |--- ZIPCode_91 >  0.50
|   |   |   |   |   |   |   |--- Age <= 46.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |--- Age >  46.00
|   |   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |--- Income >  81.50
|   |   |   |   |   |--- Mortgage <= 152.00
|   |   |   |   |   |   |--- Securities_Account <= 0.50
|   |   |   |   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |   |   |   |--- CCAvg <= 3.85
|   |   |   |   |   |   |   |   |   |--- Income <= 93.50
|   |   |   |   |   |   |   |   |   |   |--- Age <= 45.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- Age >  45.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |--- Income >  93.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  3.85
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.55] class: 1
|   |   |   |   |   |   |--- Securities_Account >  0.50
|   |   |   |   |   |   |   |--- weights: [0.60, 0.00] class: 0
|   |   |   |   |   |--- Mortgage >  152.00
|   |   |   |   |   |   |--- weights: [1.05, 0.00] class: 0
|   |   |   |--- CCAvg >  3.95
```

```
|   |   |   |   |--- weights: [6.75, 0.00] class: 0
|   |   |--- CD_Account >  0.50
|   |   |   |--- weights: [0.15, 6.80] class: 1
|--- Income >  98.50
|   |--- Education_Undergraduate <= 0.50
|   |   |--- Income <= 116.50
|   |   |   |--- CCAvg <= 2.80
|   |   |   |   |--- Income <= 106.50
|   |   |   |   |   |--- weights: [5.40, 0.00] class: 0
|   |   |   |   |--- Income >  106.50
|   |   |   |   |   |--- Age <= 57.50
|   |   |   |   |   |   |--- Age <= 41.50
|   |   |   |   |   |   |   |--- Mortgage <= 51.50
|   |   |   |   |   |   |   |   |--- CCAvg <= 1.55
|   |   |   |   |   |   |   |   |   |--- weights: [1.05, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  1.55
|   |   |   |   |   |   |   |   |   |--- CCAvg <= 1.75
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |   |   |   |   |   |   |--- CCAvg >  1.75
|   |   |   |   |   |   |   |   |   |   |--- CCAvg <= 2.40
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.35, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- CCAvg >  2.40
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.85] class: 1
|   |   |   |   |   |   |   |--- Mortgage >  51.50
|   |   |   |   |   |   |   |   |--- weights: [1.65, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  41.50
|   |   |   |   |   |   |   |--- CCAvg <= 0.35
|   |   |   |   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  0.35
|   |   |   |   |   |   |   |   |--- weights: [0.60, 3.40] class: 1
|   |   |   |   |   |--- Age >  57.50
|   |   |   |   |   |   |--- weights: [1.35, 0.00] class: 0
|   |   |   |--- CCAvg >  2.80
|   |   |   |   |--- ZIPCode_93 <= 0.50
|   |   |   |   |   |--- Age <= 63.50
|   |   |   |   |   |   |--- weights: [0.90, 19.55] class: 1
|   |   |   |   |   |--- Age >  63.50
|   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |--- ZIPCode_93 >  0.50
|   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |--- Income >  116.50
```

```
|   |   |   |--- weights: [0.00, 188.70] class: 1
|   |--- Education_Undergraduate >  0.50
|   |   |--- Family <= 2.50
|   |   |   |--- Income <= 100.00
|   |   |   |   |--- CCAvg <= 4.20
|   |   |   |   |   |--- weights: [0.45, 0.00] class: 0
|   |   |   |   |--- CCAvg >  4.20
|   |   |   |   |   |--- weights: [0.00, 1.70] class: 1
|   |   |   |--- Income >  100.00
|   |   |   |   |--- Income <= 103.50
|   |   |   |   |   |--- Securities_Account <= 0.50
|   |   |   |   |   |   |--- weights: [2.10, 0.00] class: 0
|   |   |   |   |   |--- Securities_Account >  0.50
|   |   |   |   |   |   |--- weights: [0.15, 0.85] class: 1
|   |   |   |   |--- Income >  103.50
|   |   |   |   |   |--- weights: [64.95, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- Income <= 108.50
|   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |--- weights: [1.05, 0.00] class: 0
|   |   |   |   |--- Family >  3.50
|   |   |   |   |   |--- weights: [0.15, 0.85] class: 1
|   |   |   |--- Income >  108.50
|   |   |   |   |--- weights: [0.45, 45.05] class: 1
```

In [287…

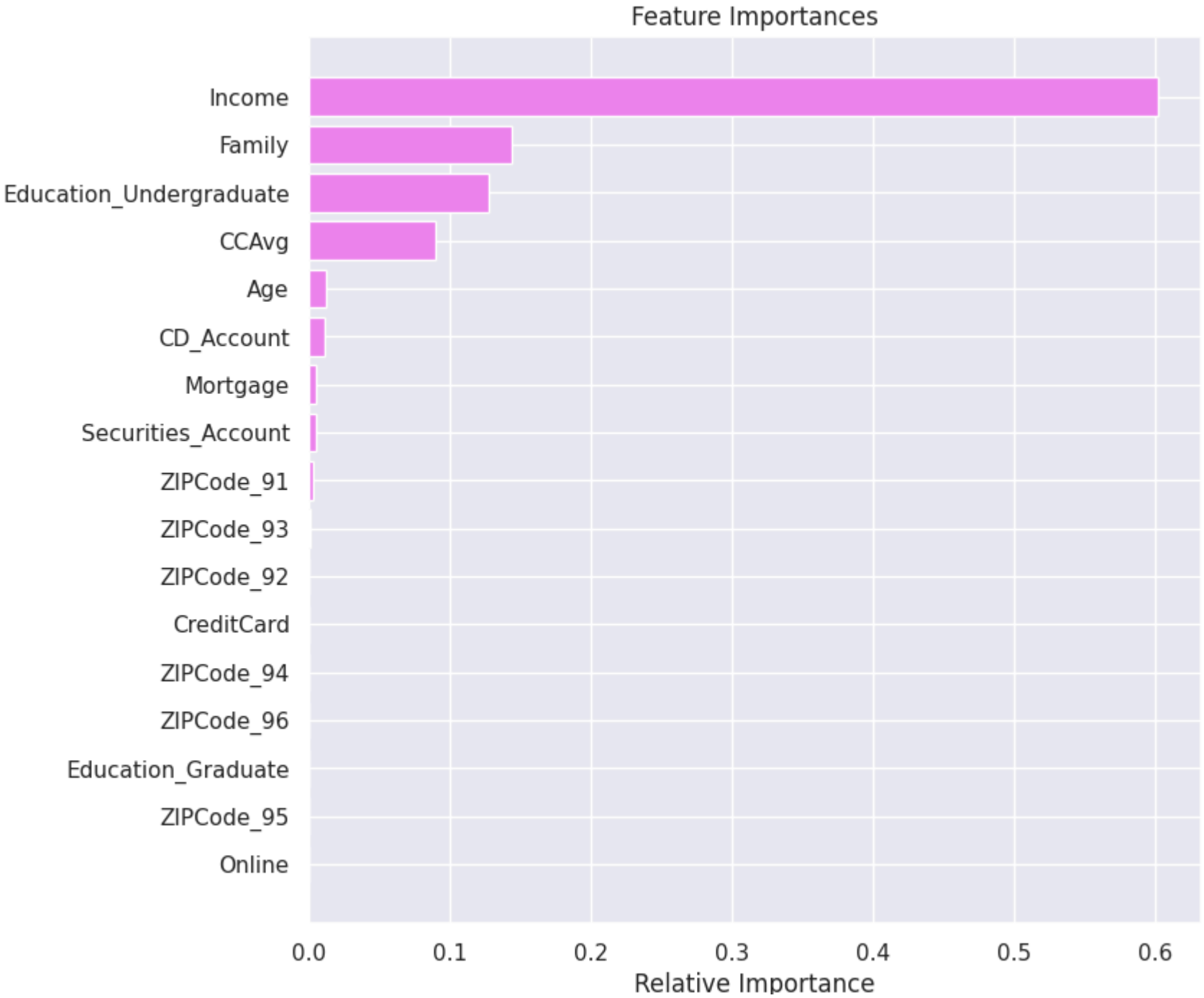
```python
# #Create and print a DataFrame of feature importances
# 1. Extract feature importances from the model
# 2. Create a DataFrame with importances and feature names computed as the  Gini importance
# 3. Display the result
print(
    pd.DataFrame(
        estimator_2.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

	Imp
Income	0.602209
Family	0.144151
Education_Undergraduate	0.127332
CCAvg	0.089356
Age	0.011925
CD_Account	0.011166
Mortgage	0.004929
Securities_Account	0.004788
ZIPCode_91	0.002635
ZIPCode_93	0.001508
Online	0.000000
CreditCard	0.000000
ZIPCode_92	0.000000
ZIPCode_94	0.000000
ZIPCode_95	0.000000
ZIPCode_96	0.000000
Education_Graduate	0.000000

In [288…

```
# Calculate feature importances from the trained estimator
importances = estimator_2.feature_importances_

# Sort feature indices based on their importances
indices = np.argsort(importances)

# Create a figure for plotting with a specific size
plt.figure(figsize=(8, 8))

# Set the title of the plot
plt.title("Feature Importances")

# Create a horizontal bar plot to visualize feature importances
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")

# Set y-axis tick labels to display feature names corresponding to their indices
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])

# Set the label for the x-axis
plt.xlabel("Relative Importance")
```

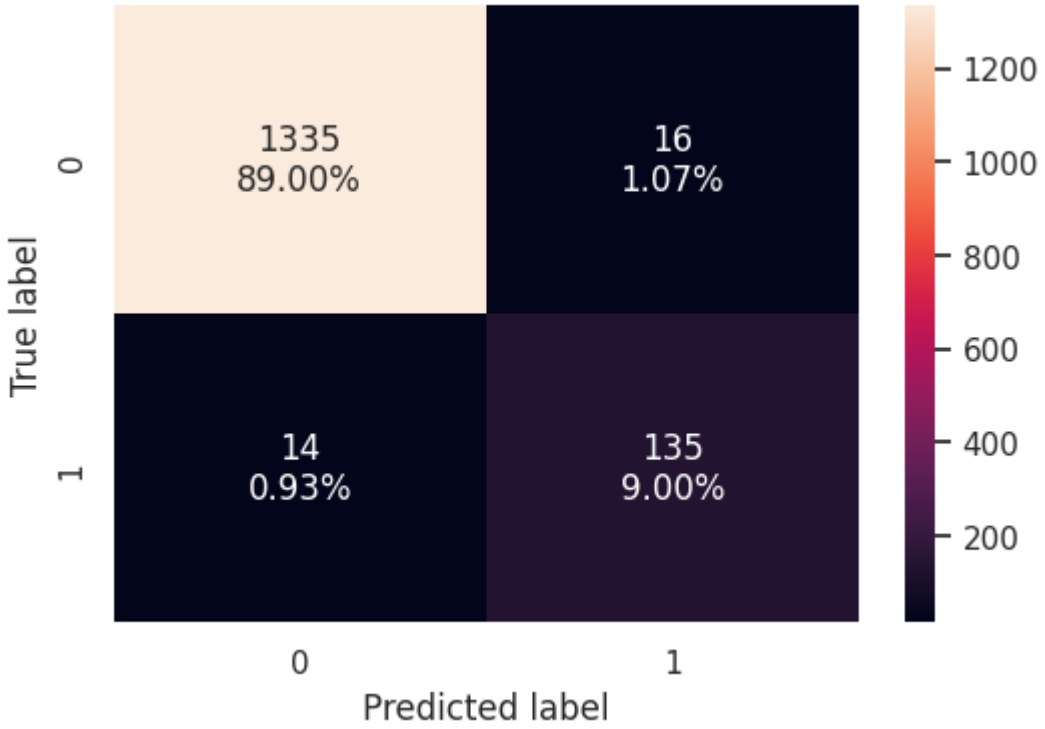
```
# Display the plot
plt.show()
```


Feature Importances
Income
Family
Education_Undergraduate
CCAvg
Age
CD_Account
Mortgage
Securities_Account
ZIPCode_91
ZIPCode_93
ZIPCode_92
CreditCard
ZIPCode_94
ZIPCode_96
Education_Graduate
ZIPCode_95
Online
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Relative Importance

- The most important feature is Income by a large margin (o.6)
- The other important features in order if importance are Education_2, CCAvg, Educatyion_3 and Family.

Checking performance on the test data

In [289…
```
# Compute and display the confusion matrix for the test set
confusion_matrix_sklearn(estimator_2, X_test, y_test)
```



In [290…
```
# Compute the performance metrics for the test set
decision_tree_perf_test = model_performance_classification_sklearn(
    estimator_2, X_test, y_test
)
```

```
# Display the performance metrics
decision_tree_perf_test
```

Out[290…

	Accuracy	Recall	Precision	F1
0	0.98	0.90604	0.89404	0.9

Model Performance Comparison and Final Model Selection

In [291…

```
# Training performance comparison
# Combine the performance metrics of the original decision tree and the tuned (pre-pruned) decision tree
models_train_comp_df = pd.concat(
    [decision_tree_perf_train.T, decision_tree_tune_perf_train.T], axis=1,
)

# Set column names for the combined dataframe
models_train_comp_df.columns = ["Decision Tree sklearn", "Decision Tree (Pre-Pruning)"]

# Print a header for the comparison
print("Training performance comparison:")

# Display the comparison dataframe
models_train_comp_df
```

```
Training performance comparison:
```

Out[291…

	Decision Tree sklearn	Decision Tree (Pre-Pruning)
Accuracy	0.994571	0.990286
Recall	1.000000	0.927492
Precision	0.945714	0.968454
F1	0.972100	0.947531

In [292…

```
# Compare the testing performance of the original decision tree and the tuned (pre-pruned) decision tree

models_test_comp_df = pd.concat(
```

```
    [decision_tree_perf_test.T, decision_tree_tune_perf_test.T], axis=1,)

# Set column names for the combined dataframe
models_test_comp_df.columns = ["Decision Tree sklearn", "Decision Tree (Pre-Pruning)"]

# Print a header for the comparison
print("Test performance comparison:")

# Display the comparison dataframe
models_test_comp_df
```

```
Test performance comparison:
```

Out[292...

	Decision Tree sklearn	Decision Tree (Pre-Pruning)
Accuracy	0.98000	0.980000
Recall	0.90604	0.865772
Precision	0.89404	0.928058
F1	0.90000	0.895833

Actionable Insights and Business Recommendations

Expolatory Data Analysis observations

- Income is right-skewed and has many outliers above the 3rd quartile.
- CCAvg is right-skewed and has many outliers above the 3rd quartile.
- Mortgage is heavily right-skewed and there are many outliers above the 3rd quartile in this variable.
- 29.4% of customers have one child, and only 20.2% of customers have three children.
- The majority (42%) of customers have an undergraduate level of education.
- 89.6% of customers do not have a securities account
- 94% of customers do not have a CD account

- 59% of users do their banking online
- 70% of customers do not have a credit card
- The majority of customers (29.4%) live in area 94.
- Very few customers (0.8%) live in area 96
- There is an extremely strong correlation (0.99) between age and experience.
- There is a strong correlation (.65) between income and credit card average.
- 90% of customers do not have a personal loan
- Customers with advanced/professional degrees are about three times more likely to have a personal loan than those with undergraduate degrees.
- Education of advanced/professional and graduates have similar rates of personal loans, both significantly higher than undergraduate.
- 90.4% of families do not have a personal loan
- Larger families (3 -4 persons) seem to have a higher percentage of personal loans compared to smaller families (1- 2 persons)
- 89.6% of customers do not have a securities account
- There is a slightly higher percentage of securities account holders among those with personal loans (12.5%) than those without (10.22%).
- 93% of customers do not have a CD account.
- There seems to be a higher proportion of individuals with personal loans among those with a CD account than those who do not have a CD account.
- 59.68% of customers use online banking
- Online banking is more popular than traditional banking in this dataset, regardless of loan status
- 70.6% of customers do not have a credit card.
- Credit card ownership rates are similar between those customers with personal loans and those without.
- ZIP code 94 has the highest population (1472, 29.44% of the sample).
- ZIP code 96 has the lowest population (40, 0.8% of the sample).
- Personal loan rates are relatively consistent across most ZIP codes, ranging from 9.37% to 10.31% for ZIP codes 90-95.
- Income, CCAvg, and Mortgage have a significant number of values above the 3rd quartile that are considered outliers. However, these values are legitimate and do not require outlier treatment.
- There is a drop of in personal loans as the customers have more experience.

Analysis of Model Performance for AllLife Bank

Importance of Recall

- **Recall is the most critical metric** for AllLife Bank's objective of expanding its borrower base and converting liability customers to personal loan customers.
- Recall measures the proportion of actual positive cases (potential loan customers) correctly identified.
- A high recall indicates the model excels at identifying potential loan customers.

Strategy Justification

- It's more beneficial to reach out to a larger pool of potential customers, even if some don't take loans than to miss potential loan customers.

This will maximize the number of customers who are likely to take out a personal loan, which should lead to a higher number of customers taking out a personal loan.

Model Comparison: Post Pruning vs Pre-Pruning

The Post Pruning model outperforms the Pre-Pruning model:

1. **Higher Recall**: 0.852349 vs 0.785235
   - Most crucial for the bank's objectives
2. **Better F1 Score**: 0.891228 vs 0.879699
   - Indicates a superior balance between precision and recall

- These factors make the Post Pruning model more aligned with AllLife Bank's goal of rapidly expanding its borrower base. This is the model that should be used.

3. See model visualization and feature importance below

Recommendations

Feature Importance:

- Focus marketing efforts on the most influential features for loan acceptance. Based on the feature importance of this model, income, and family size are top factors, tailoring marketing messages around financial stability and family financial planning.

Address Class Imbalance: Given that borrowers are a small portion of the customer base, consider techniques to address class imbalance, such as:

- Oversampling the minority class (borrowers)
- Undersampling the majority class (non-borrowers)

Targeted Marketing:

- Use the insights from the model to create more targeted marketing campaigns. For example, if certain age groups or income levels are more likely to take loans, focus efforts there.

Cross-Selling Strategy:

- Develop a strategy to cross-sell personal loans to existing liability customers. Use the model to identify which depositors are most likely interested in loans.

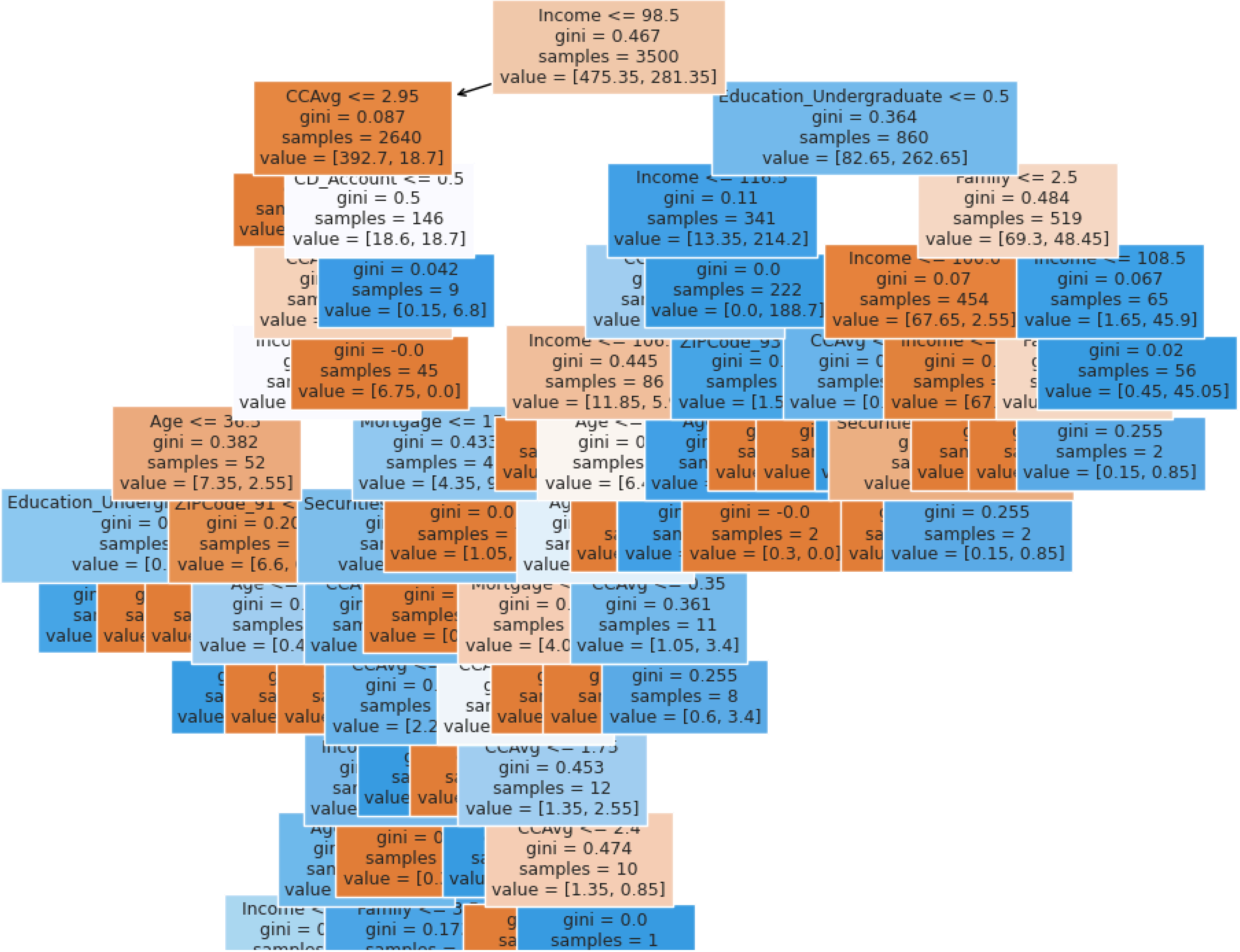
Customer Segmentation:

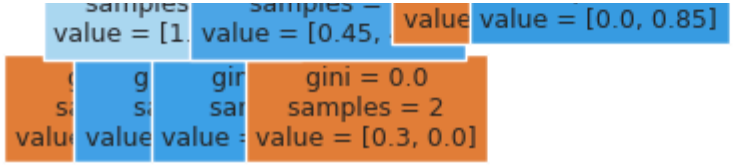
- Use the model insights to create customer segments. Tailor product offerings and marketing messages to each segment's characteristics and needs.

In [293…

```
#This is the visualiztion of the Post Prune tree that is the model that should be used
# Set up the figure size for the plot
plt.figure(figsize=(10, 10))

# Plot the decision tree using the specified estimator and configuration
out = tree.plot_tree(
    estimator_2,                # The decision tree classifier to plot
    feature_names=feature_names,  # Names of the features used in the decision tree
    filled=True,                # Color the nodes to reflect the majority class
    fontsize=9,                 # Set the font size for the text in the plot
    node_ids=False,             # Do not display node IDs
```

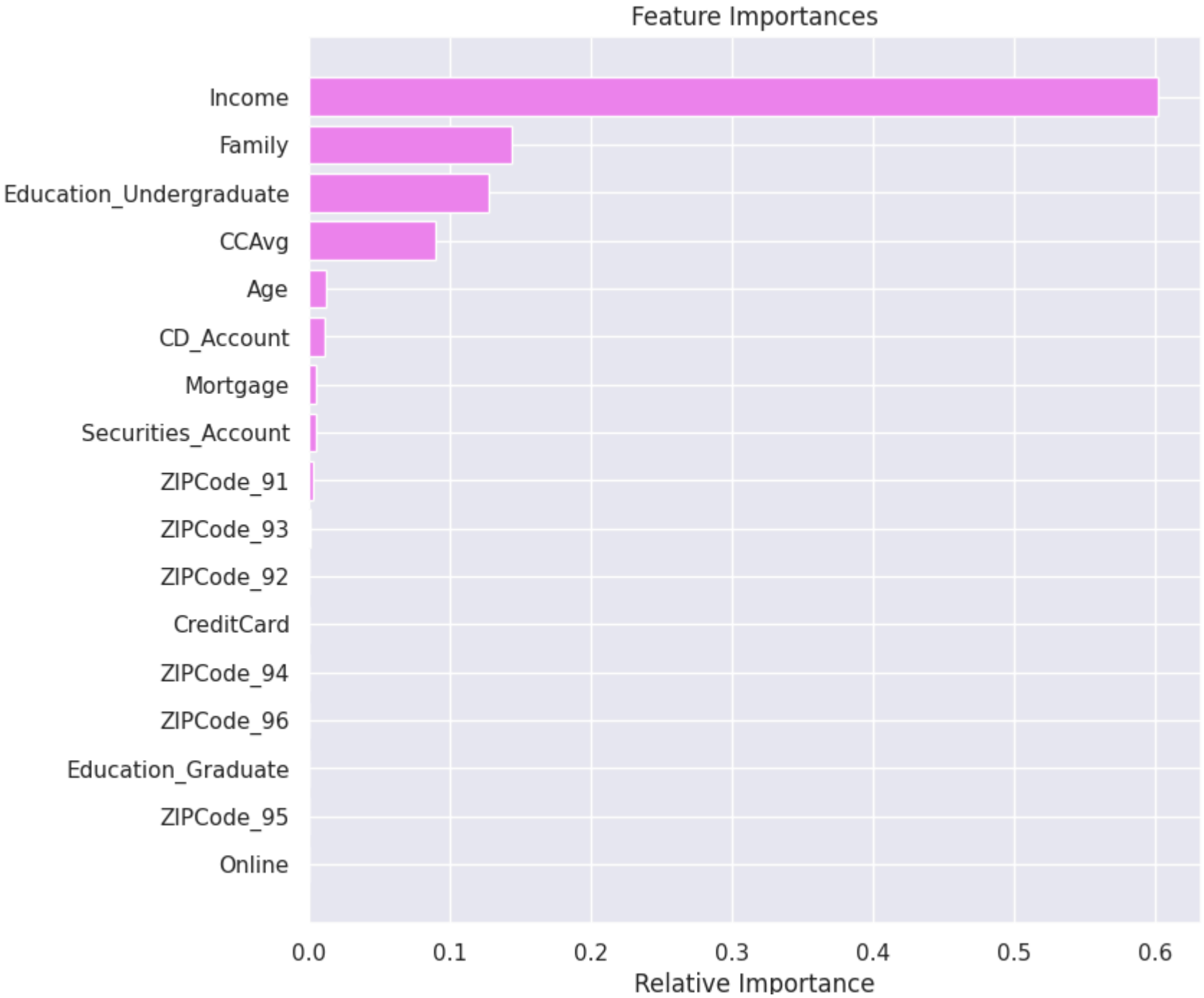
```
    class_names=None            # Do not display class names
)

# Loop through the plotted tree elements to 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 the edge color of the arrows to black
        arrow.set_linewidth(1)        # Set the line width of the arrows to 1

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


Income <= 98.5
gini = 0.467
samples = 3500
value = [475.35, 281.35]
CCAvg <= 2.95
gini = 0.087
samples = 2640
value = [392.7, 18.7]
Education_Undergraduate <= 0.5
gini = 0.364
samples = 860
value = [82.65, 262.65]
gini = 0.5
samples = 146
value = [18.6, 18.7]
gini = 0.11
samples = 341
value = [13.35, 214.2]
gini = 0.484
samples = 519
value = [69.3, 48.45]
gini = 0.042
samples = 9
value = [0.15, 6.8]
gini = 0.0
samples = 222
value = [0.0, 188.7]
gini = 0.07
samples = 454
value = [67.65, 2.55]
gini = 0.067
samples = 65
value = [1.65, 45.9]
gini = -0.0
samples = 45
value = [6.75, 0.0]
gini = 0.445
samples = 86
gini = 0.02
samples = 56
value = [0.45, 45.05]
gini = 0.382
samples = 52
value = [7.35, 2.55]
gini = 0.255
samples = 2
value = [0.15, 0.85]
gini = -0.0
samples = 2
value = [0.3, 0.0]
gini = 0.255
samples = 2
value = [0.15, 0.85]
gini = 0.361
samples = 11
value = [1.05, 3.4]
gini = 0.255
samples = 8
value = [0.6, 3.4]
gini = 0.453
samples = 12
value = [1.35, 2.55]
gini = 0.474
samples = 10
value = [1.35, 0.85]
gini = 0.0
samples = 1



In [294…

```
# This is the importance for the post prune model
# Calculate feature importances from the trained estimator
importances = estimator_2.feature_importances_

# Sort feature indices based on their importances
indices = np.argsort(importances)

# Create a figure for plotting with a specific size
plt.figure(figsize=(8, 8))

# Set the title of the plot
plt.title("Feature Importances")

# Create a horizontal bar plot to visualize feature importances
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")

# Set y-axis tick labels to display feature names corresponding to their indices
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])

# Set the label for the x-axis
plt.xlabel("Relative Importance")

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


Feature Importances
Income
Family
Education_Undergraduate
CCAvg
Age
CD_Account
Mortgage
Securities_Account
ZIPCode_91
ZIPCode_93
ZIPCode_92
CreditCard
ZIPCode_94
ZIPCode_96
Education_Graduate
ZIPCode_95
Online
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Relative Importance