Copy of AIML_ML_Project_low_code_notebook

February 17, 2025

0.1 Problem Statement

0.1.1 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.

0.1.2 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.

0.1.3 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)

0.2 Please read the instructions carefully before starting the project.

This is a commented Jupyter IPython Notebook file in which all the instructions and tasks to be performed are mentioned. * Blanks '__________' are provided in the notebook that needs to be filled with an appropriate code to get the correct result. With every '__________' blank, there is a comment that briefly describes what needs to be filled in the blank space. * Identify the task to be performed correctly, and only then proceed to write the required code. * Fill the code wherever asked by the commented lines like "# write your code here" or "# complete the code". Running incomplete code may throw error. * Please run the codes in a sequential manner from the beginning to avoid any unnecessary errors. * Add the results/observations (wherever mentioned) derived from the analysis in the presentation and submit the same.

0.3 Importing necessary libraries

[ ]:
```
# Installing the libraries with the specified version.
!pip install numpy==1.24.3
!pip install pandas==2.0.3
!pip install matplotlib==3.7.1
```

```
Collecting numpy==1.24.3
  Downloading numpy-1.24.3.tar.gz (10.9 MB)
     ---------------------------------------- 0.0/10.9 MB ? eta -:--:--
     ---------------------------------------- 0.0/10.9 MB ? eta -:--:--
     - -------------------------------------- 0.5/10.9 MB 2.4 MB/s eta 0:00:05
     ---- ----------------------------------- 1.3/10.9 MB 2.6 MB/s eta 0:00:04
     -------- ------------------------------- 2.4/10.9 MB 3.4 MB/s eta 0:00:03
     ------------ --------------------------- 3.4/10.9 MB 3.7 MB/s eta 0:00:03
     ---------------- ----------------------- 4.5/10.9 MB 3.9 MB/s eta 0:00:02
     ------------------- -------------------- 5.2/10.9 MB 4.1 MB/s eta 0:00:02
     ----------------------- ---------------- 6.3/10.9 MB 4.2 MB/s eta 0:00:02
     -------------------------- ------------- 7.3/10.9 MB 4.3 MB/s eta 0:00:01
     ------------------------------ --------- 8.4/10.9 MB 4.3 MB/s eta 0:00:01
     ----------------------------------- ---- 9.7/10.9 MB 4.5 MB/s eta 0:00:01
     ------------------------------------- -- 10.2/10.9 MB 4.5 MB/s eta 0:00:01
     ---------------------------------------- 10.9/10.9 MB 4.4 MB/s eta 0:00:00
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'error'

  error: subprocess-exited-with-error

  Getting requirements to build wheel did not run successfully.
  exit code: 1
```

```
  [33 lines of output]
  Traceback (most recent call last):
    File "C:\Users\Aubrey\anaconda3\Lib\site-
packages\pip\_vendor\pyproject_hooks\_in_process\_in_process.py", line 353, in
<module>
      main()
    File "C:\Users\Aubrey\anaconda3\Lib\site-
packages\pip\_vendor\pyproject_hooks\_in_process\_in_process.py", line 335, in
main
      json_out['return_val'] = hook(**hook_input['kwargs'])
                               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    File "C:\Users\Aubrey\anaconda3\Lib\site-
packages\pip\_vendor\pyproject_hooks\_in_process\_in_process.py", line 112, in
get_requires_for_build_wheel
      backend = _build_backend()
                ^^^^^^^^^^^^^^^^
    File "C:\Users\Aubrey\anaconda3\Lib\site-
packages\pip\_vendor\pyproject_hooks\_in_process\_in_process.py", line 77, in
_build_backend
      obj = import_module(mod_path)
            ^^^^^^^^^^^^^^^^^^^^^^^
    File "C:\Users\Aubrey\anaconda3\Lib\importlib\__init__.py", line 90, in
import_module
      return _bootstrap._gcd_import(name[level:], package, level)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    File "<frozen importlib._bootstrap>", line 1387, in _gcd_import
    File "<frozen importlib._bootstrap>", line 1360, in _find_and_load
    File "<frozen importlib._bootstrap>", line 1310, in _find_and_load_unlocked
    File "<frozen importlib._bootstrap>", line 488, in _call_with_frames_removed
    File "<frozen importlib._bootstrap>", line 1387, in _gcd_import
    File "<frozen importlib._bootstrap>", line 1360, in _find_and_load
    File "<frozen importlib._bootstrap>", line 1331, in _find_and_load_unlocked
    File "<frozen importlib._bootstrap>", line 935, in _load_unlocked
    File "<frozen importlib._bootstrap_external>", line 995, in exec_module
    File "<frozen importlib._bootstrap>", line 488, in _call_with_frames_removed
    File "C:\Users\Aubrey\AppData\Local\Temp\pip-build-env-
jhbw16zf\overlay\Lib\site-packages\setuptools\__init__.py", line 16, in <module>
      import setuptools.version
    File "C:\Users\Aubrey\AppData\Local\Temp\pip-build-env-
jhbw16zf\overlay\Lib\site-packages\setuptools\version.py", line 1, in <module>
      import pkg_resources
    File "C:\Users\Aubrey\AppData\Local\Temp\pip-build-env-
jhbw16zf\overlay\Lib\site-packages\pkg_resources\__init__.py", line 2172, in
<module>
      register_finder(pkgutil.ImpImporter, find_on_path)
                      ^^^^^^^^^^^^^^^^^^^
  AttributeError: module 'pkgutil' has no attribute 'ImpImporter'. Did you mean:
'zipimporter'?
```

```
  [end of output]

  note: This error originates from a subprocess, and is likely not a problem
with pip.
error: subprocess-exited-with-error

Getting requirements to build wheel did not run successfully.
exit code: 1

See above for output.

note: This error originates from a subprocess, and is likely not a problem with
pip.
```

[1]:
```
!pip install uszipcode
```

```
Collecting uszipcode
  Downloading uszipcode-1.0.1-py2.py3-none-any.whl.metadata (8.9 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.11/dist-packages
(from uszipcode) (25.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-
packages (from uszipcode) (2.32.3)
Collecting pathlib-mate (from uszipcode)
  Downloading pathlib_mate-1.3.2-py3-none-any.whl.metadata (8.4 kB)
Collecting atomicwrites (from uszipcode)
  Downloading atomicwrites-1.4.1.tar.gz (14 kB)
  Preparing metadata (setup.py) … done
Collecting fuzzywuzzy (from uszipcode)
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl.metadata (4.9 kB)
Collecting haversine>=2.5.0 (from uszipcode)
  Downloading haversine-2.9.0-py2.py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: SQLAlchemy>=1.4.0 in
/usr/local/lib/python3.11/dist-packages (from uszipcode) (2.0.38)
Collecting sqlalchemy-mate>=1.4.28.3 (from uszipcode)
  Downloading sqlalchemy_mate-2.0.0.3-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: greenlet!=0.4.17 in
/usr/local/lib/python3.11/dist-packages (from SQLAlchemy>=1.4.0->uszipcode)
(3.1.1)
Requirement already satisfied: typing-extensions>=4.6.0 in
/usr/local/lib/python3.11/dist-packages (from SQLAlchemy>=1.4.0->uszipcode)
(4.12.2)
Requirement already satisfied: prettytable<4.0.0,>=3.0.0 in
/usr/local/lib/python3.11/dist-packages (from sqlalchemy-
mate>=1.4.28.3->uszipcode) (3.14.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->uszipcode) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests->uszipcode) (3.10)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->uszipcode) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->uszipcode) (2025.1.31)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.11/dist-
packages (from prettytable<4.0.0,>=3.0.0->sqlalchemy-mate>=1.4.28.3->uszipcode)
(0.2.13)
Downloading uszipcode-1.0.1-py2.py3-none-any.whl (35 kB)
Downloading haversine-2.9.0-py2.py3-none-any.whl (7.7 kB)
Downloading sqlalchemy_mate-2.0.0.3-py3-none-any.whl (59 kB)
                         59.9/59.9 kB
2.0 MB/s eta 0:00:00
Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Downloading pathlib_mate-1.3.2-py3-none-any.whl (56 kB)
                         57.0/57.0 kB
2.1 MB/s eta 0:00:00
Building wheels for collected packages: atomicwrites
  Building wheel for atomicwrites (setup.py) … done
  Created wheel for atomicwrites: filename=atomicwrites-1.4.1-py2.py3-none-
any.whl size=6941
sha256=90f377caf4fd5263caba43448750afbd7ed62d9488db954b18884ff1296cc8aa
  Stored in directory: /root/.cache/pip/wheels/f7/99/9c/d24e98c35f30eba0c367ad1e
7888d396d676abb35fe1e7611c
Successfully built atomicwrites
Installing collected packages: fuzzywuzzy, pathlib-mate, haversine,
atomicwrites, sqlalchemy-mate, uszipcode
Successfully installed atomicwrites-1.4.1 fuzzywuzzy-0.18.0 haversine-2.9.0
pathlib-mate-1.3.2 sqlalchemy-mate-2.0.0.3 uszipcode-1.0.1
```

Note:

1. After running the above cell, kindly restart the notebook kernel (for Jupyter Notebook) or runtime (for Google Colab) and run all cells sequentially from the next cell.
2. On executing the above line of code, you might see a warning regarding package dependencies. This error message can be ignored as the above code ensures that all necessary libraries and their dependencies are maintained to successfully execute the code in this notebook.

[82]:
```python
# 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

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

# to suppress unnecessary warnings
import warnings
warnings.filterwarnings("ignore")
```

0.4 Loading the dataset

[83]:
```
# uncomment the following lines if Google Colab is being used
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[84]:
```
Loan = pd.read_csv("/content/drive/My Drive/data_files/Loan_Modelling.csv")  ␣
↪##  Complete the code to read the data
```

[85]:
```
# copying data to another variable to avoid any changes to original data
data = Loan.copy()
```

0.5 Data Overview

0.5.1 View the first and last 5 rows of the dataset.

[86]:
```
data.head(5)  ##  Complete the code to view top 5 rows of the data
```

[86]:
```
   ID  Age  Experience  Income  ZIPCode  Family  CCAvg  Education  Mortgage  \
0   1   25           1      49    91107       4    1.6          1         0
1   2   45          19      34    90089       3    1.5          1         0
2   3   39          15      11    94720       1    1.0          1         0
3   4   35           9     100    94112       1    2.7          2         0
4   5   35           8      45    91330       4    1.0          2         0

   Personal_Loan  Securities_Account  CD_Account  Online  CreditCard
0              0                   1           0       0           0
1              0                   1           0       0           0
```

```
2              0                    0           0       0            0
3              0                    0           0       0            0
4              0                    0           0       0            1
```

[87]:
```
data.tail(5)  ##  Complete the code to view last 5 rows of the data
```

[87]:
```
        ID  Age  Experience  Income  ZIPCode  Family  CCAvg  Education  \
4995  4996   29           3      40    92697       1    1.9          3
4996  4997   30           4      15    92037       4    0.4          1
4997  4998   63          39      24    93023       2    0.3          3
4998  4999   65          40      49    90034       3    0.5          2
4999  5000   28           4      83    92612       3    0.8          1

      Mortgage  Personal_Loan  Securities_Account  CD_Account  Online  \
4995         0              0                   0           0       1
4996        85              0                   0           0       1
4997         0              0                   0           0       0
4998         0              0                   0           0       1
4999         0              0                   0           0       1

      CreditCard
4995           0
4996           0
4997           0
4998           0
4999           1
```

0.5.2 Understand the shape of the dataset.

[88]:
```
data.shape  ## Complete the code to get the shape of the data
```

[88]:
```
(5000, 14)
```

The dataset has 5000 rows and 14 columns

0.5.3 Check the data types of the columns for the dataset

[89]:
```
data.info()  ##  Complete the code to view the datatypes of the data
```

```
<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
```

There are 14 numerical and 0 categorical variables in the data (before the conversion)

[90]:
```
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
```

6 numerical and 7 categorical (after the conversion)

0.5.4 Checking the Statistical Summary

[91]:
```
data.describe().T  ## Complete the code to print the statistical summary of the
 ↪data
```

[91]:
```
                     count         mean          std   min      25%  \
ID                  5000.0  2500.500000  1443.520003   1.0  1250.75
Age                 5000.0    45.338400    11.463166  23.0    35.00
```

```
Experience          5000.0     20.104600    11.467954     -3.0     10.00
Income              5000.0     73.774200    46.033729      8.0     39.00
ZIPCode             5000.0  93169.257000  1759.455086  90005.0  91911.00
Family              5000.0      2.396400     1.147663      1.0      1.00
CCAvg               5000.0      1.937938     1.747659      0.0      0.70
Education           5000.0      1.881000     0.839869      1.0      1.00
Mortgage            5000.0     56.498800   101.713802      0.0      0.00
Personal_Loan       5000.0      0.096000     0.294621      0.0      0.00
Securities_Account  5000.0      0.104400     0.305809      0.0      0.00
CD_Account          5000.0      0.060400     0.238250      0.0      0.00
Online              5000.0      0.596800     0.490589      0.0      0.00
CreditCard          5000.0      0.294000     0.455637      0.0      0.00

                        50%       75%      max
ID                   2500.5   3750.25   5000.0
Age                    45.0     55.00     67.0
Experience             20.0     30.00     43.0
Income                 64.0     98.00    224.0
ZIPCode             93437.0  94608.00  96651.0
Family                  2.0      3.00      4.0
CCAvg                   1.5      2.50     10.0
Education               2.0      3.00      3.0
Mortgage                0.0    101.00    635.0
Personal_Loan           0.0      0.00      1.0
Securities_Account      0.0      0.00      1.0
CD_Account              0.0      0.00      1.0
Online                  1.0      1.00      1.0
CreditCard              0.0      1.00      1.0
```

Observations:

On average, customers have an annual income of ~$74k. Income range is from $8k to $224k per year. 3rd quartile is $98k which indicates outliers.

The age of the customers range from 23 to 67 with a median age of ~45 years old

On average, customers spend ~$2k a month in credit card purchases

0.5.5 Dropping columns

[92]:
```
data = data.drop(['ID'], axis=1)  ## Complete the code to drop a column from
 ↪the dataframe
```

Dropped the ID column, as it is a unique identifier that does not describe the data

0.6 Data Preprocessing

0.6.1 Checking for Anomalous Values

[93]:
```
data["Experience"].unique()
```

[93]:
```
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])
```

Customers' professional experience ranges from 1 to 43 years

[94]:
```
# checking for experience <0
data[data["Experience"] < 0]["Experience"].unique()
```

[94]:
```
array([-1, -2, -3])
```

[95]:
```
# 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)
```

[96]:
```
data["Education"].unique()
```

[96]:
```
array([1, 2, 3])
```

0.6.2 Feature Engineering

[97]:
```
# checking the number of uniques in the zip code
data["ZIPCode"].nunique()
```

[97]:
```
467
```

[98]:
```
data["ZIPCode"] = data["ZIPCode"].astype(str)
print(
    "Number of unique values if we take first two digits of ZIPCode: ",
    data["ZIPCode"].str[0:2].nunique(),
)
data["ZIPCode"] = data["ZIPCode"].str[0:2]

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

```
Number of unique values if we take first two digits of ZIPCode:  7
```

[99]:
```
## Converting the data type of categorical features to 'category'
cat_cols = [
    "Education",
    "Personal_Loan",
    "Securities_Account",
    "CD_Account",
```

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

0.7 Exploratory Data Analysis (EDA)

0.7.1 Univariate Analysis

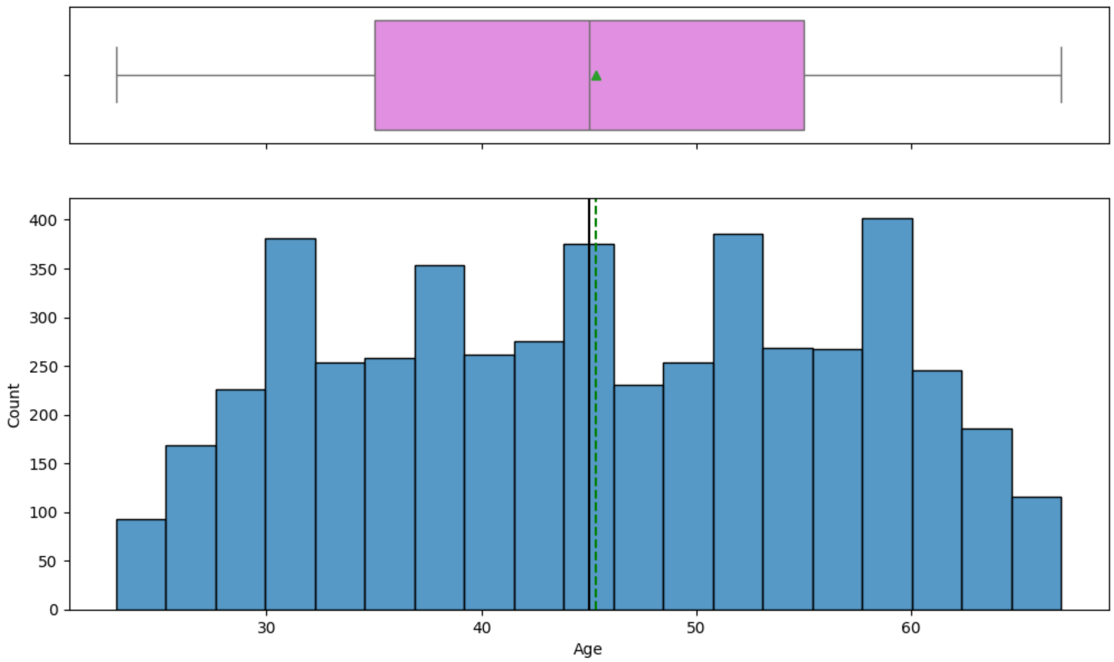
[100]:
```
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    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)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the␣
↪column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

[101]:
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
```

```
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all␣
↪levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

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

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

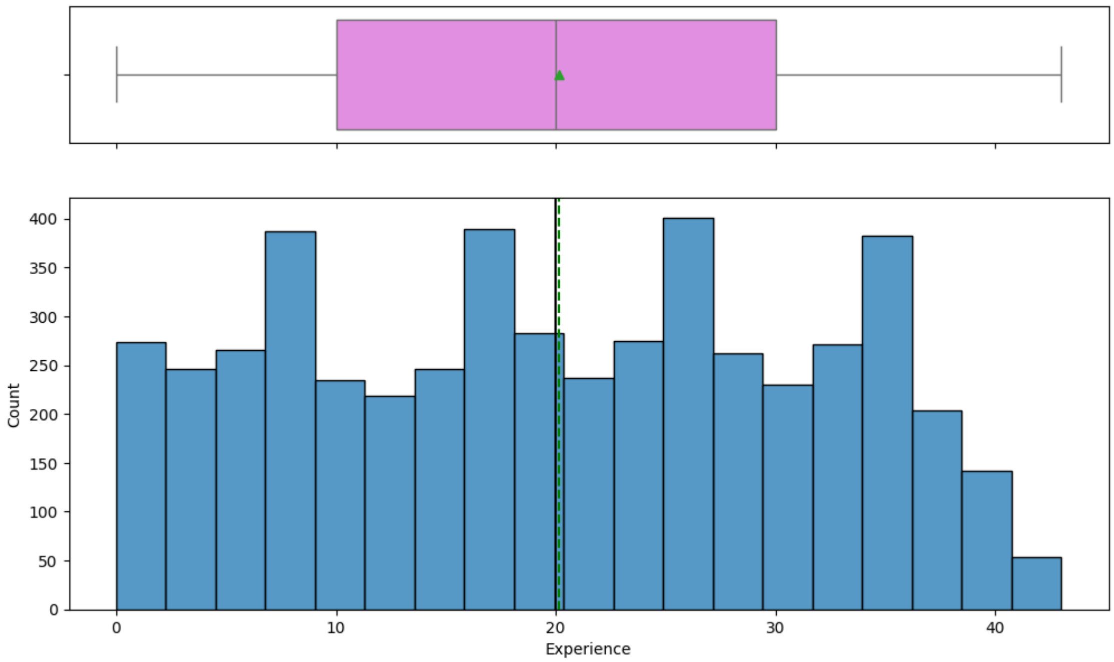
Observations on Age

[102]:
```
histogram_boxplot(data, "Age")
```



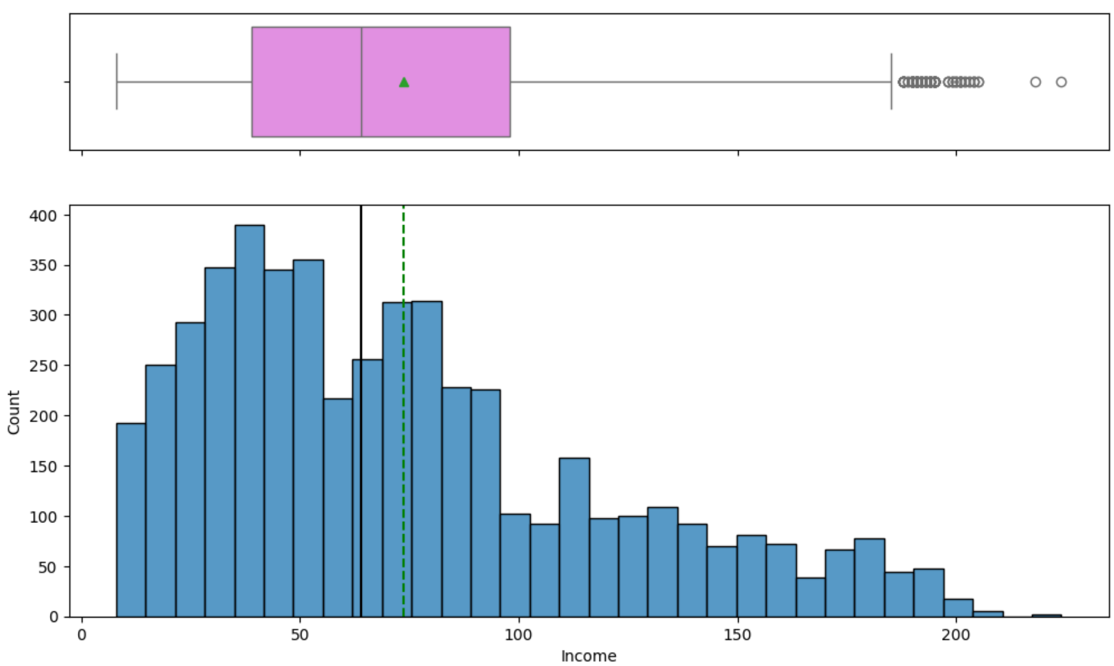
Observations on Experience

[103]:
```
histogram_boxplot(data, 'Experience') ## Complete the code to create␣
↪histogram_boxplot for experience
```



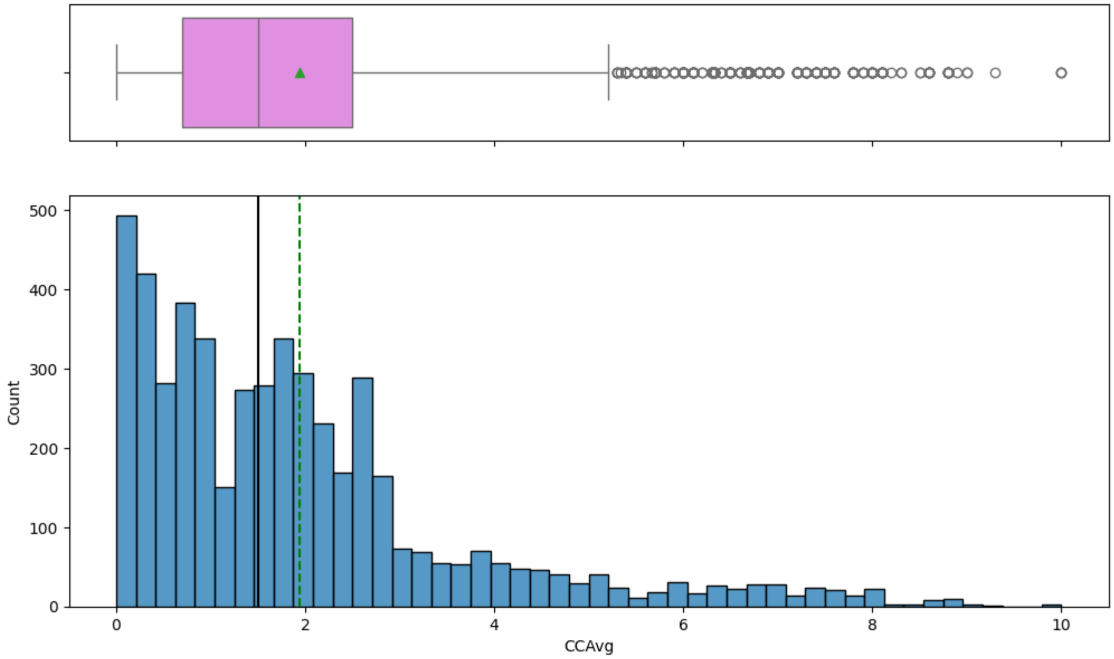
Observations on Income

[104]:
```
histogram_boxplot(data, 'Income')  ## Complete the code to create␣
↪histogram_boxplot for Income
```



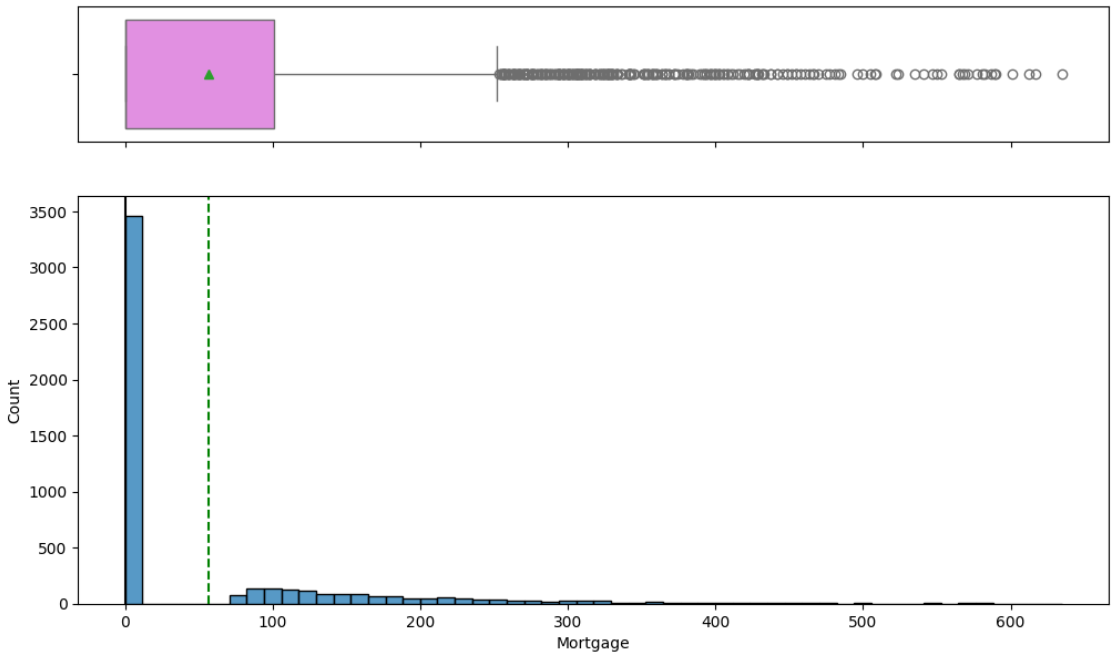
Observations on CCAvg

[105]:
```
histogram_boxplot(data, 'CCAvg')  ## Complete the code to create␣
↪histogram_boxplot for CCAvg
```



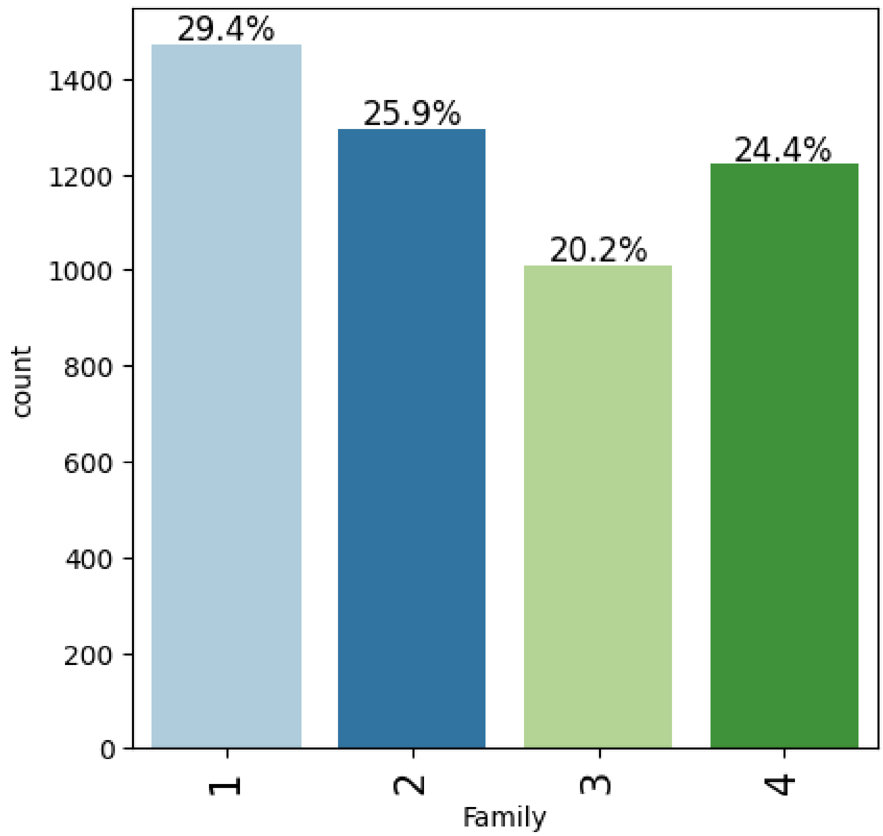
Observations on Mortgage

[106]:
```
histogram_boxplot(data, 'Mortgage')  ## Complete the code to create␣
↪histogram_boxplot for Mortgage
```



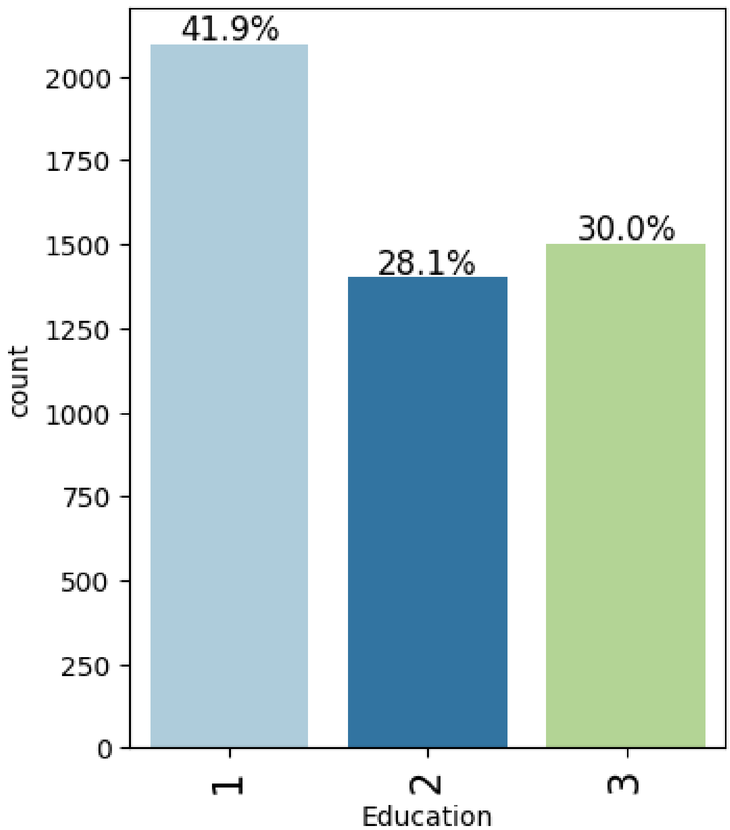
Observations on Family

[107]:
```
labeled_barplot(data, "Family", perc=True)
```



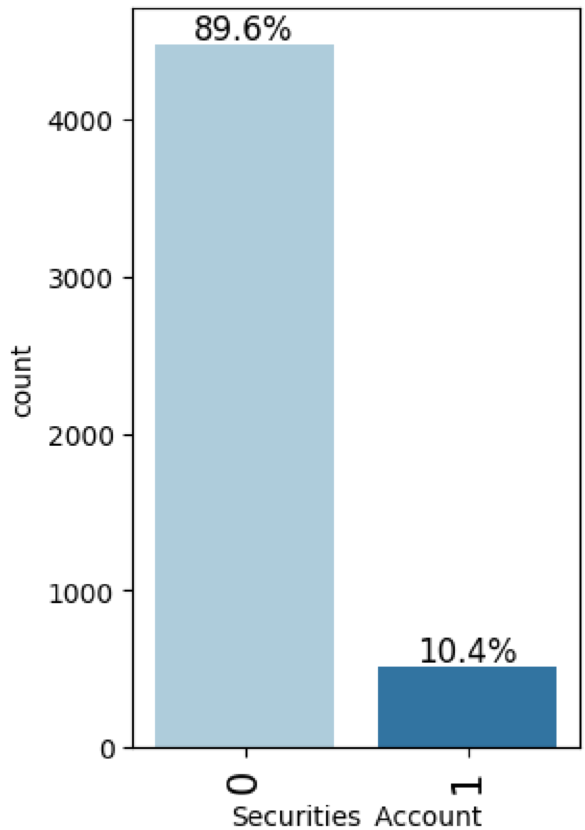
Observations on Education

[108]:
```
labeled_barplot(data, "Education", perc=True)    ## Complete the code to create␣
↪labeled_barplot for Education
```



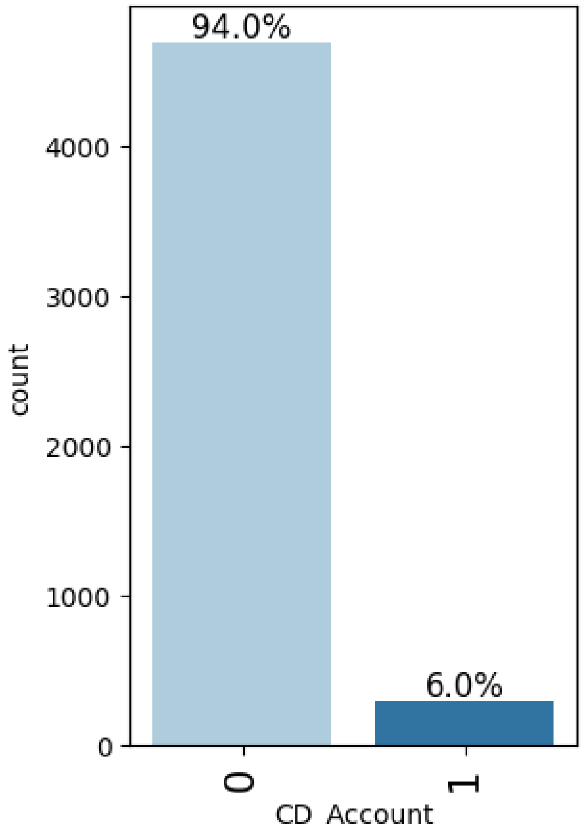
Observations on Securities_Account

```
[109]: labeled_barplot(data, "Securities_Account", perc=True)   ## Complete the code
       ↪to create labeled_barplot for Securities_Account
```



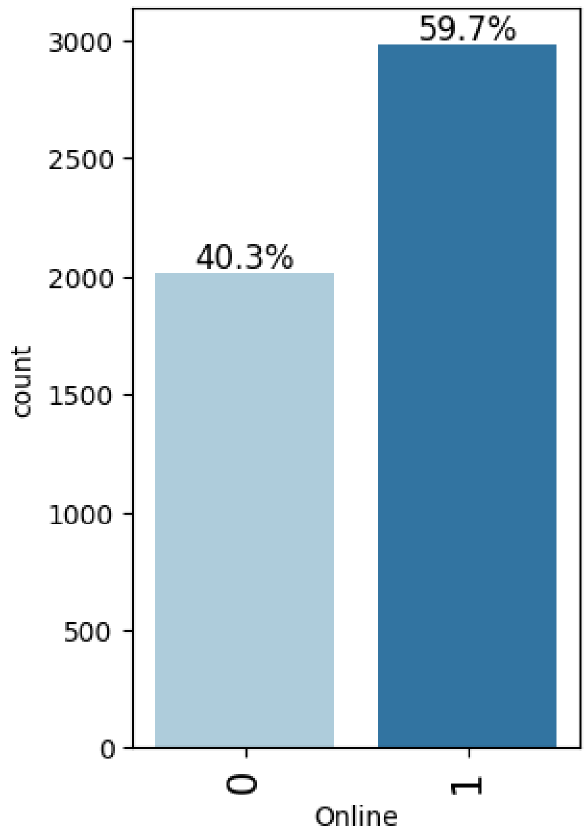
Observations on CD_Account

[110]:
```
labeled_barplot(data, "CD_Account", perc=True)    ## Complete the code to create
↪labeled_barplot for CD_Account
```



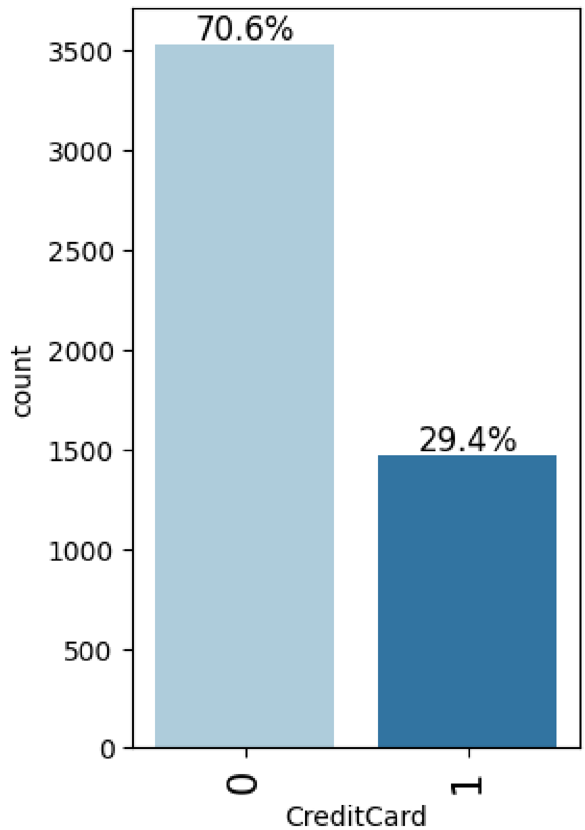
Observations on Online

[111]:
```
labeled_barplot(data, "Online", perc=True)   ## Complete the code to create␣
↪labeled_barplot for Online
```



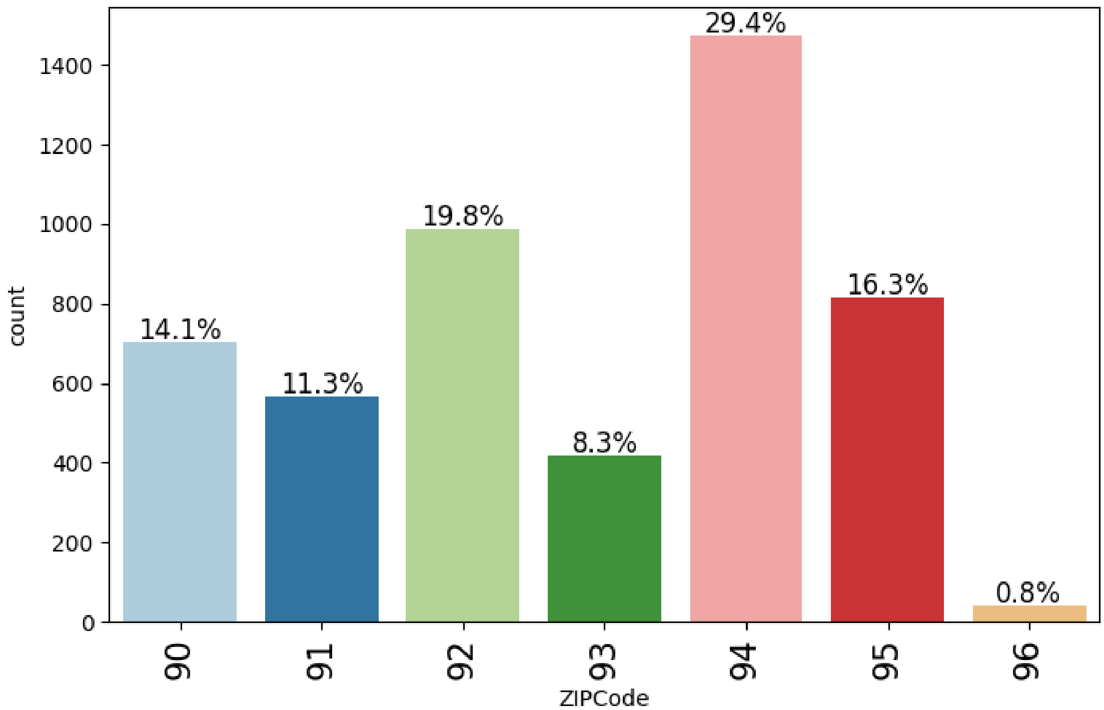
Observation on CreditCard

[112]:
```
labeled_barplot(data, "CreditCard", perc=True)    ## Complete the code to create
↪labeled_barplot for CreditCard
```



Observation on ZIPCode

[113]:
```
labeled_barplot(data, "ZIPCode", perc=True)   ## Complete the code to create␣
↪labeled_barplot for ZIPCode
```



0.7.2 Bivariate Analysis

[114]:
```
def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").
 ↪sort_values(
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 5, 5))
    plt.legend(
```

```
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

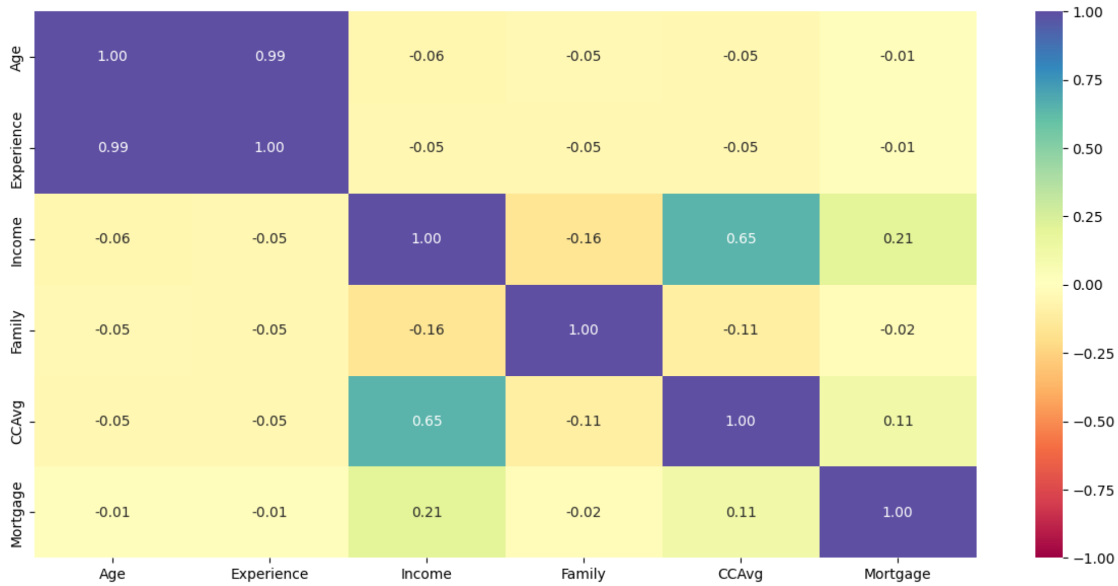
[115]:

```
### function to plot distributions wrt target


def distribution_plot_wrt_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

    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",
    )

    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",
    )

    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")

    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",
)

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

Correlation check

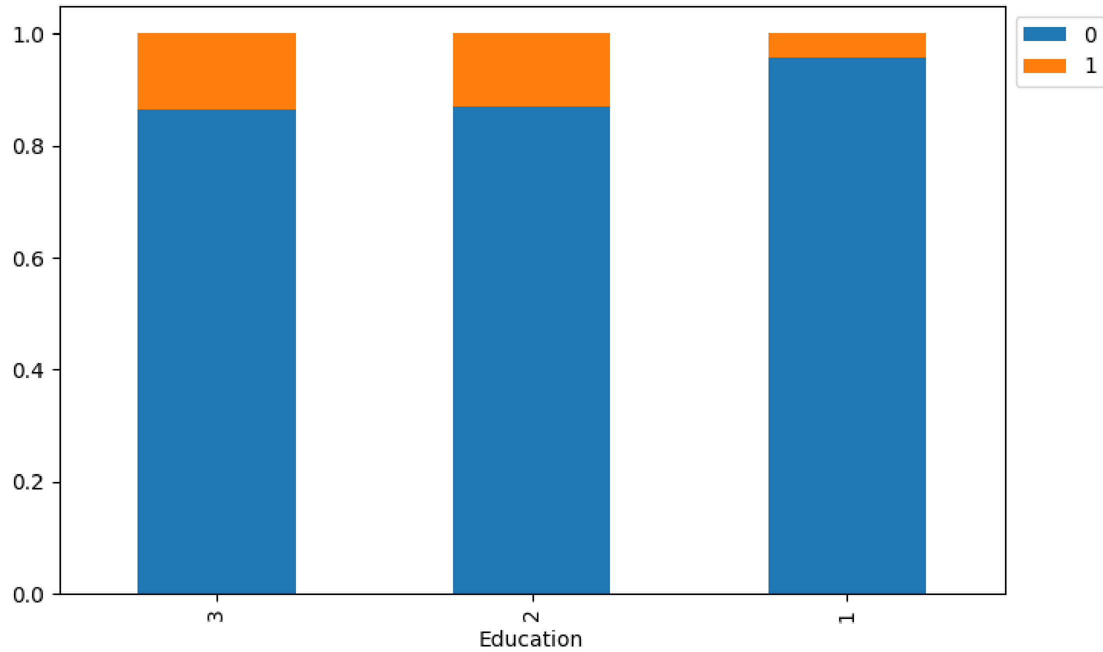
[116]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(data.corr(numeric_only=True), annot=True, vmin=-1, vmax=1, fmt=".
↪2f", cmap="Spectral") # Complete the code to get the heatmap of the data
plt.show()
```



Let's check how a customer's interest in purchasing a loan varies with their education

[117]:
```
stacked_barplot(data, "Education", "Personal_Loan")
```

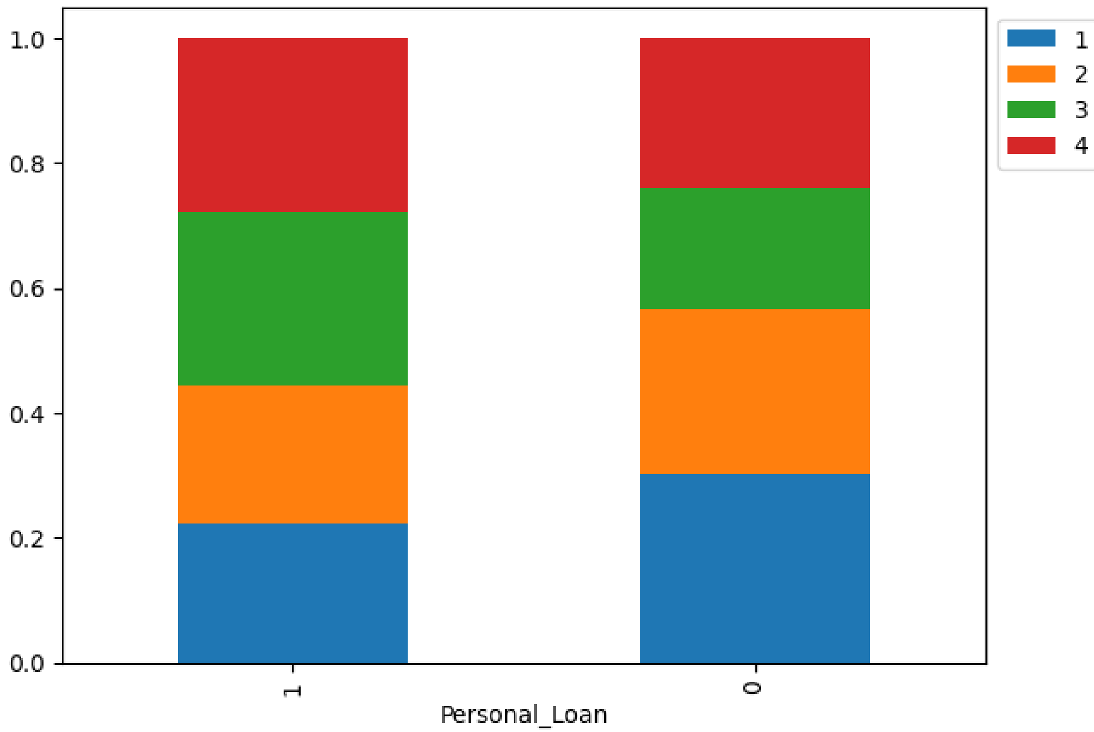
```
Personal_Loan     0    1   All
Education
All            4520  480  5000
3              1296  205  1501
2              1221  182  1403
1              2003   93  2096
--------------------------------------------------------------------------------
----------------------------------------
```



Personal_Loan vs Family

[118]:
```
stacked_barplot(data, "Personal_Loan", "Family")  ## Complete the code to plot
 ↪stacked barplot for Personal Loan and Family
```

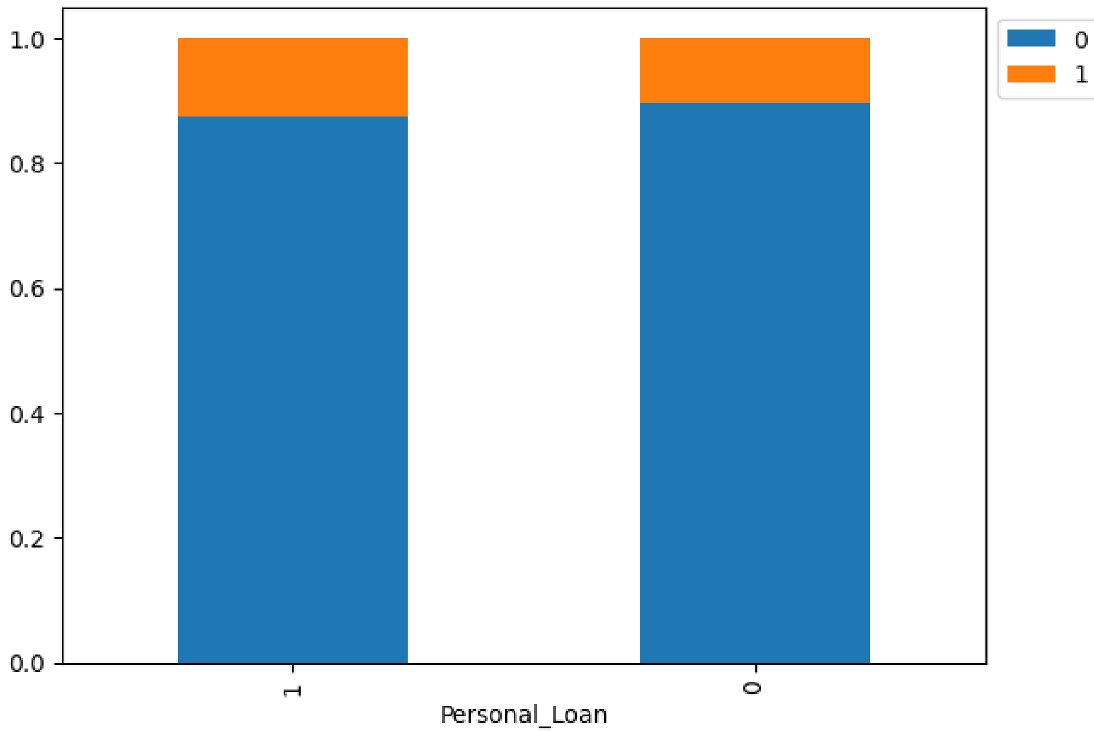
```
Family            1     2     3     4   All
Personal_Loan
All            1472  1296  1010  1222  5000
0              1365  1190   877  1088  4520
1               107   106   133   134   480
--------------------------------------------------------------------------------
----------------------------------------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



Personal_Loan vs Securities_Account

[119]:
```
stacked_barplot(data, "Personal_Loan", "Securities_Account") ## Complete the
↪code to plot stacked barplot for Personal Loan and Securities_Account
```

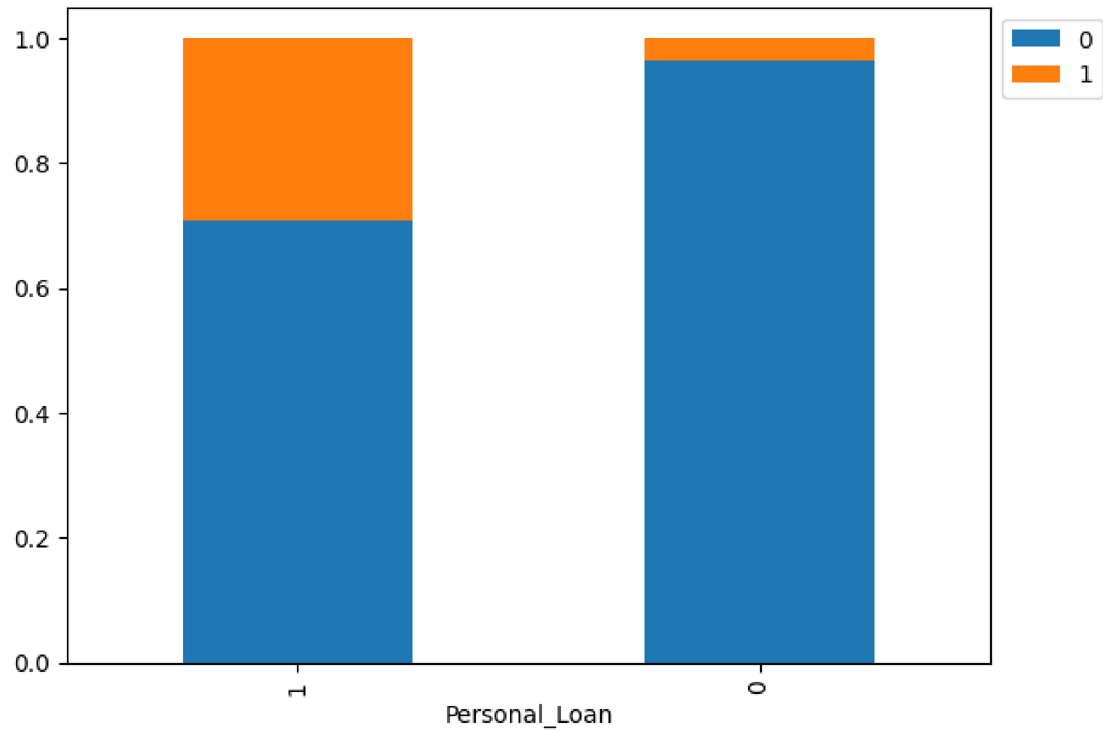
```
Securities_Account     0    1   All
Personal_Loan
All                 4478  522  5000
0                   4058  462  4520
1                    420   60   480
------------------------------------------------------------------------------------------------------------------------
```



Personal_Loan vs CD_Account

[120]:
```
stacked_barplot(data, "Personal_Loan", "CD_Account") ## Complete the code to
↪plot stacked barplot for Personal Loan and CD_Account
```

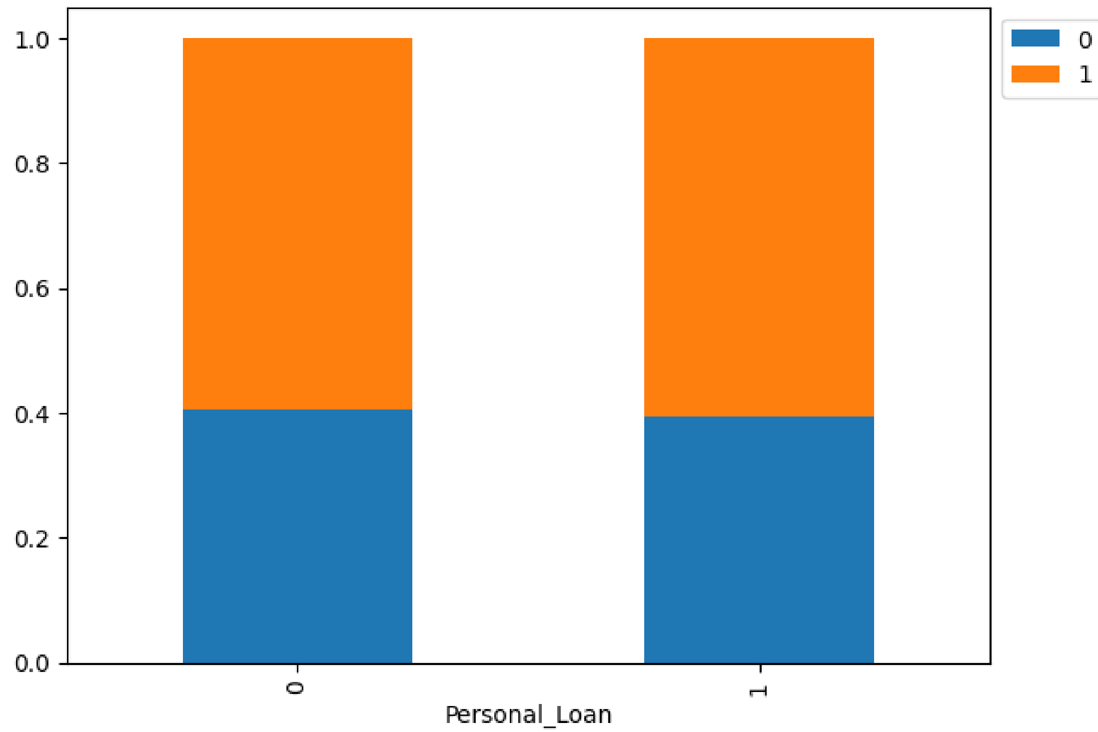
```
CD_Account        0    1   All
Personal_Loan
All            4698  302  5000
0              4358  162  4520
1               340  140   480
--------------------------------------------------------------------------------
----------------------------------------
```



Personal_Loan vs Online

[121]:
```
stacked_barplot(data, "Personal_Loan", "Online") ## Complete the code to plot␣
↪stacked barplot for Personal Loan and Online
```

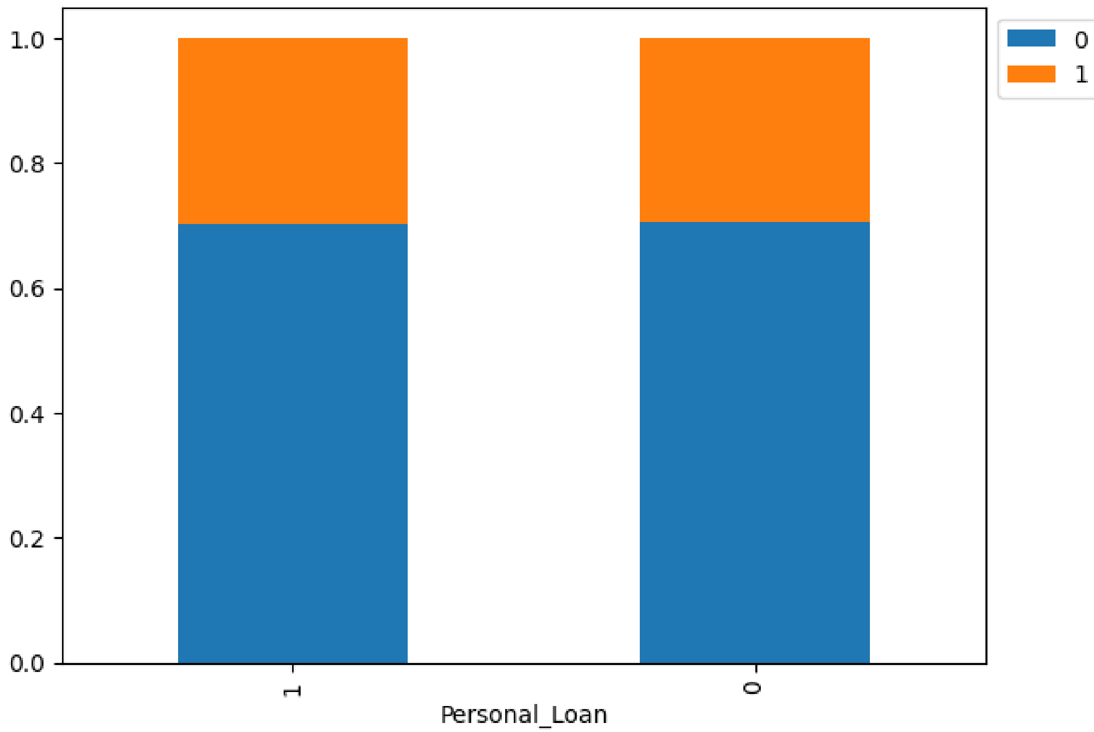
```
Online            0     1   All
Personal_Loan
All            2016  2984  5000
0              1827  2693  4520
1               189   291   480
--------------------------------------------------------------------------------
----------------------------------------
```



Personal_Loan vs CreditCard

[122]:
```
stacked_barplot(data, "Personal_Loan", "CreditCard") ## Complete the code to␣
↪plot stacked barplot for Personal Loan and CreditCard
```

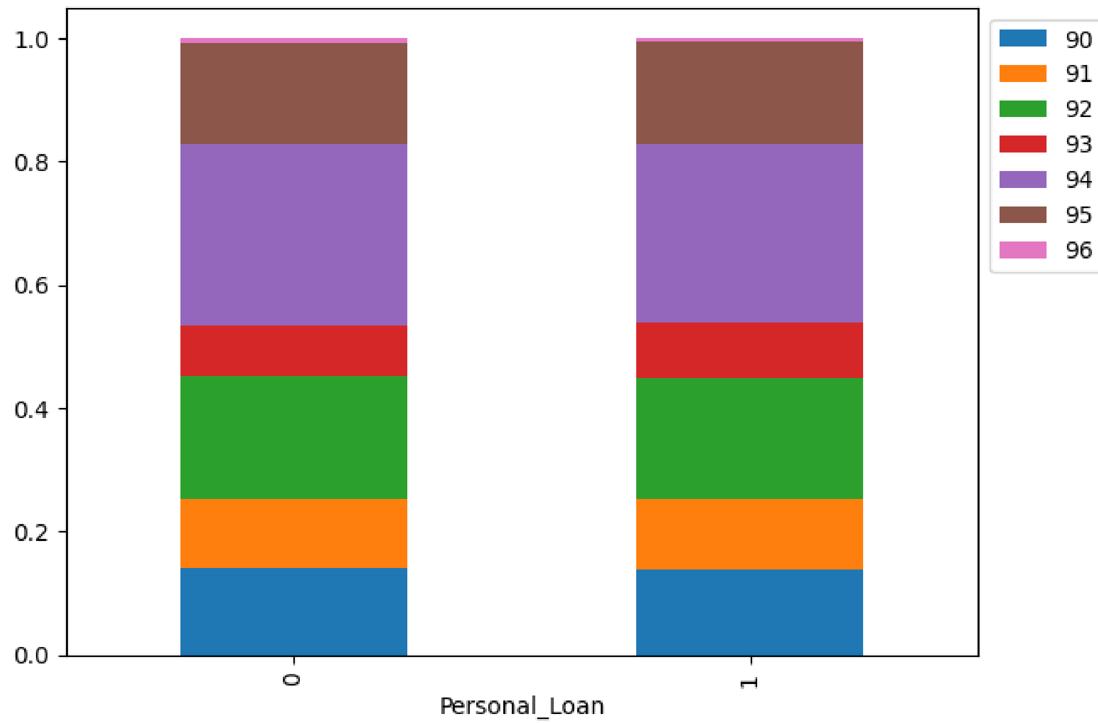
```
CreditCard        0     1   All
Personal_Loan
All            3530  1470  5000
0              3193  1327  4520
1               337   143   480
--------------------------------------------------------------------------------
----------------------------------------
```



Personal_Loan vs ZIPCode

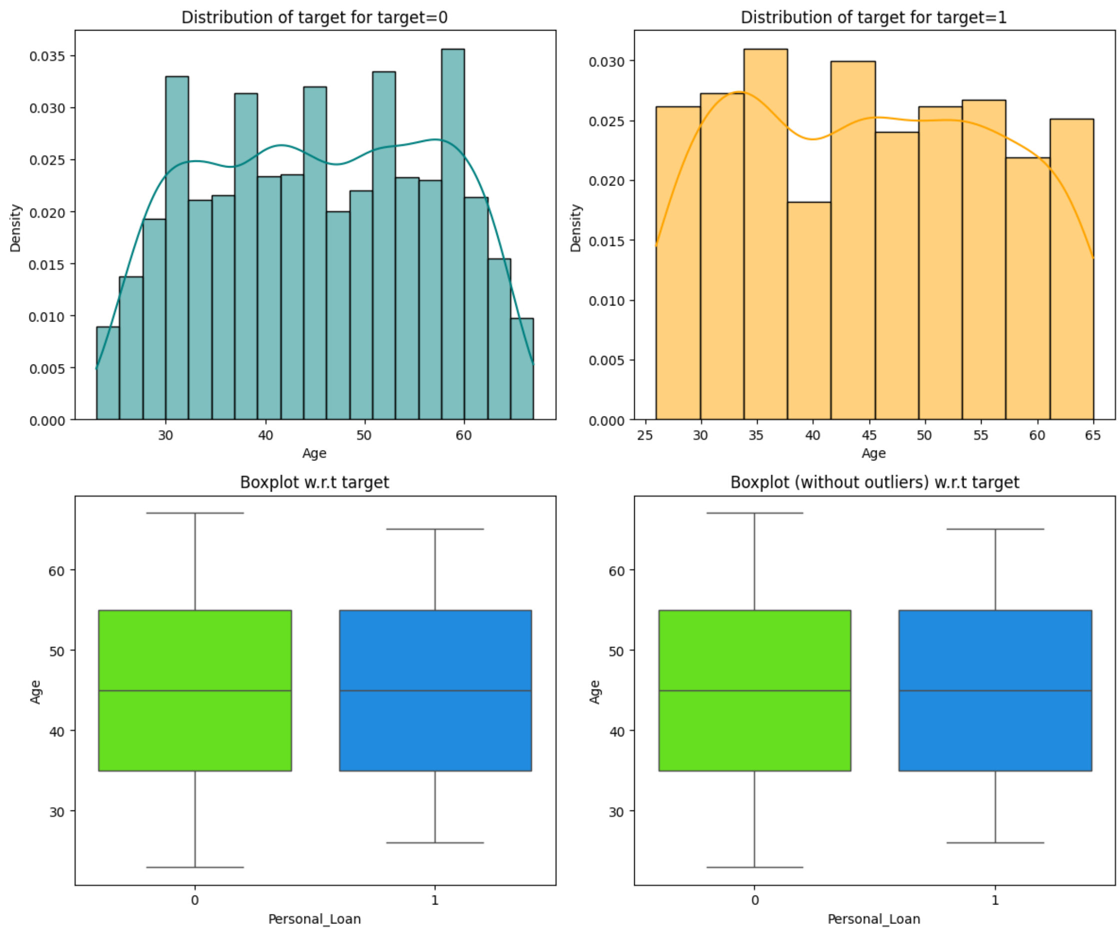
[123]:
```
stacked_barplot(data, "Personal_Loan", "ZIPCode") ## Complete the code to plot␣
↪stacked barplot for Personal Loan and ZIPCode
```

```
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
------------------------------------------------------------------------------------------------------------------------
```



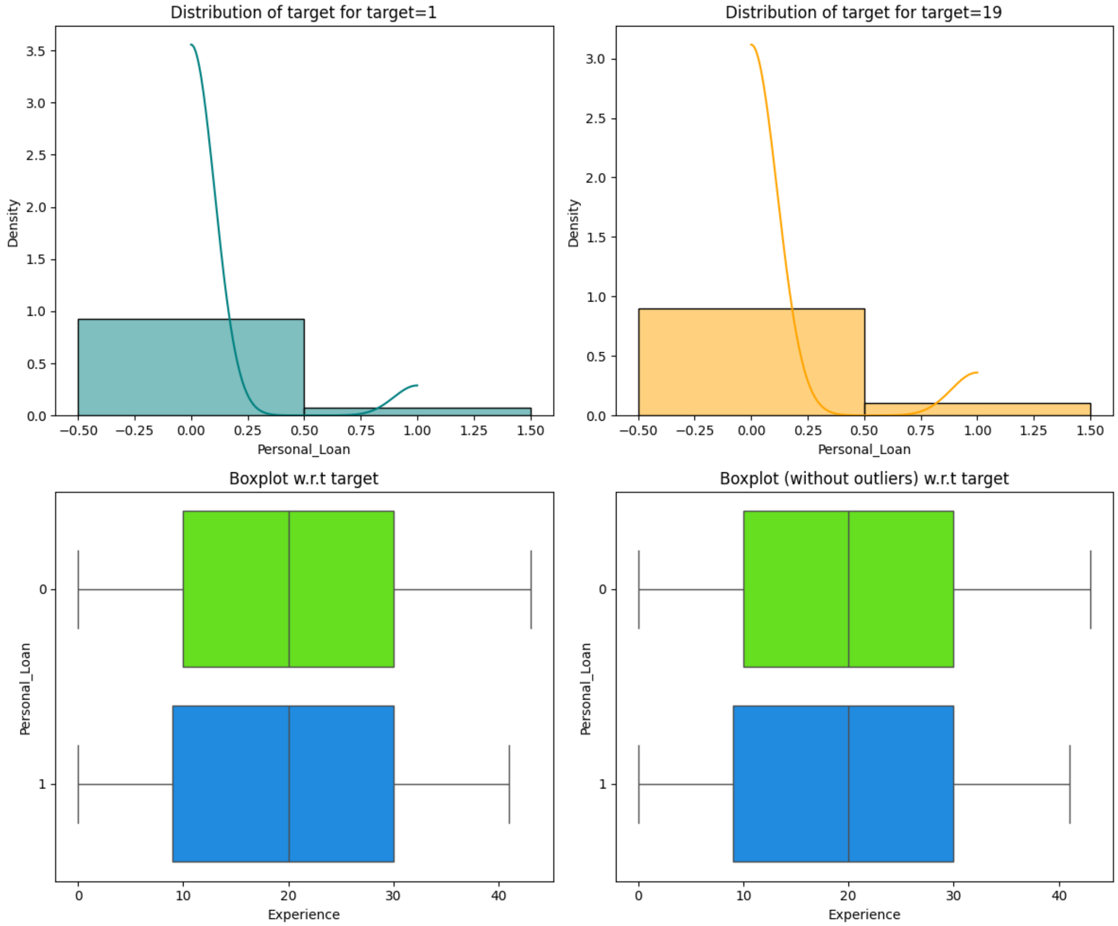
Let's check how a customer's interest in purchasing a loan varies with their age

[124]:
```
distribution_plot_wrt_target(data, "Age", "Personal_Loan")
```



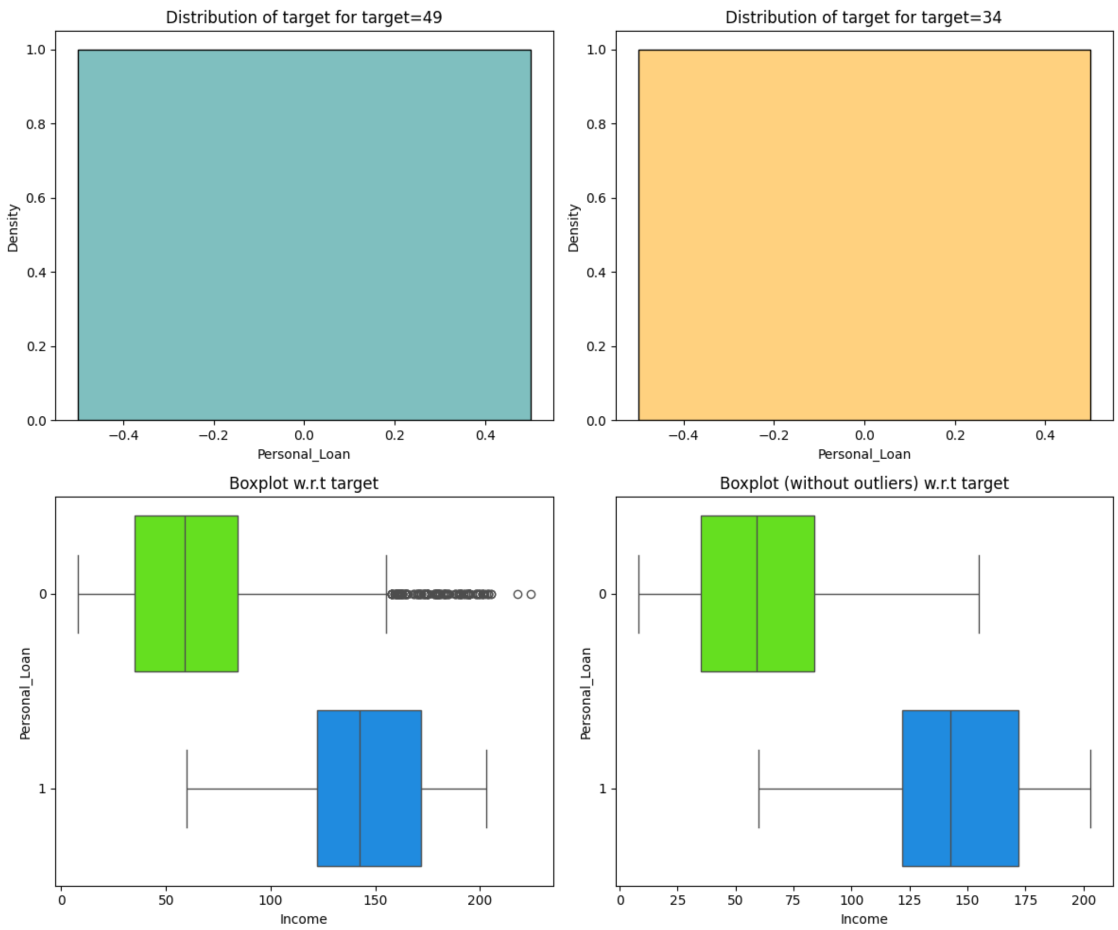
Personal Loan vs Experience

[125]:
```
distribution_plot_wrt_target(data, "Personal_Loan", "Experience") ## Complete
↪the code to plot stacked barplot for Personal Loan and Experience
```



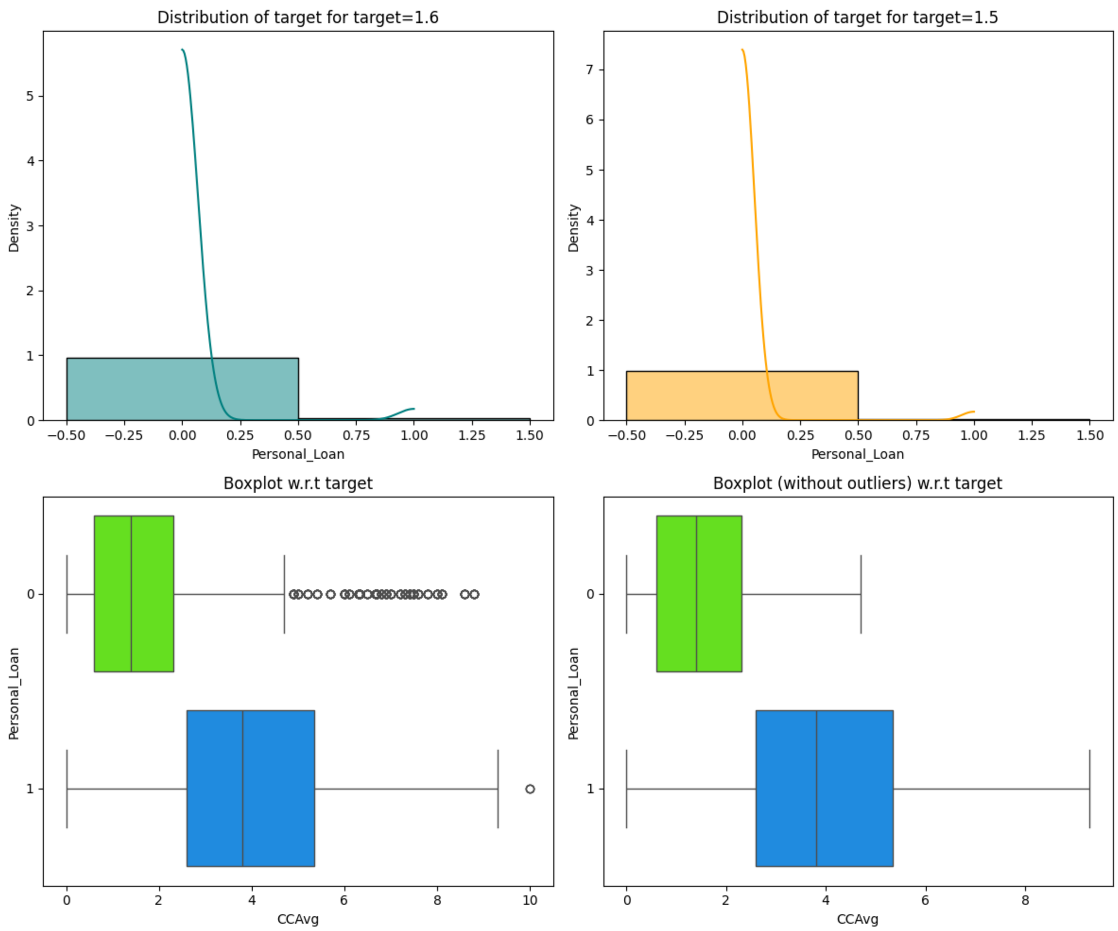
Personal Loan vs Income

[126]:
```
distribution_plot_wrt_target(data, "Personal_Loan", "Income") ## Complete the␣
↪code to plot stacked barplot for Personal Loan and Income
```



Personal Loan vs CCAvg

[127]:
```
distribution_plot_wrt_target(data, "Personal_Loan", "CCAvg") ## Complete the␣
↪code to plot stacked barplot for Personal Loan and CCAvg
```



0.8 Data Preprocessing (contd.)

0.8.1 Outlier Detection

[128]:
```
Q1 = data.select_dtypes(include=["float64", "int64"]).quantile(0.25)  # To find␣
↪the 25th percentile and 75th percentile.
Q3 = data.select_dtypes(include=["float64", "int64"]).quantile(0.75)

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

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

[129]:
```
(
    (data.select_dtypes(include=["float64", "int64"]) < lower)
```

```
    | (data.select_dtypes(include=["float64", "int64"]) > upper)
).sum() / len(data) * 100
```

[129]:
```
Age           0.00
Experience    0.00
Income        1.92
Family        0.00
CCAvg         6.48
Mortgage      5.82
dtype: float64
```

0.8.2 Data Preparation for Modeling

[130]:
```
# dropping Experience as it is perfectly correlated with Age
X = data.drop(["Personal_Loan", "Experience"], axis=1)
Y = data["Personal_Loan"]

X = pd.get_dummies(X, columns=["ZIPCode", "Education"], drop_first=True)

X = X.astype(float)

# 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
)
```

[131]:
```
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:
Personal_Loan
0    0.905429
1    0.094571
Name: proportion, dtype: float64
Percentage of classes in test set:
Personal_Loan
0    0.900667
1    0.099333
Name: proportion, dtype: float64
```

0.9 Model Building

0.9.1 Model Evaluation Criterion

Model Evaluation Criterion: The gini criterion will help the decision tree find optimal splits in the data to classify loan acceptance

First, let's create functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

- The model_performance_classification_sklearn function will be used to check the model performance of models.
- The confusion_matrix_sklearnfunction will be used to plot confusion matrix.

[132]:
```
# defining a function to compute different metrics to check performance of a␣
↪classification model built using sklearn
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model␣
↪performance

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

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

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

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

    return df_perf
```

[133]:
```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

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

```
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().
↪sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

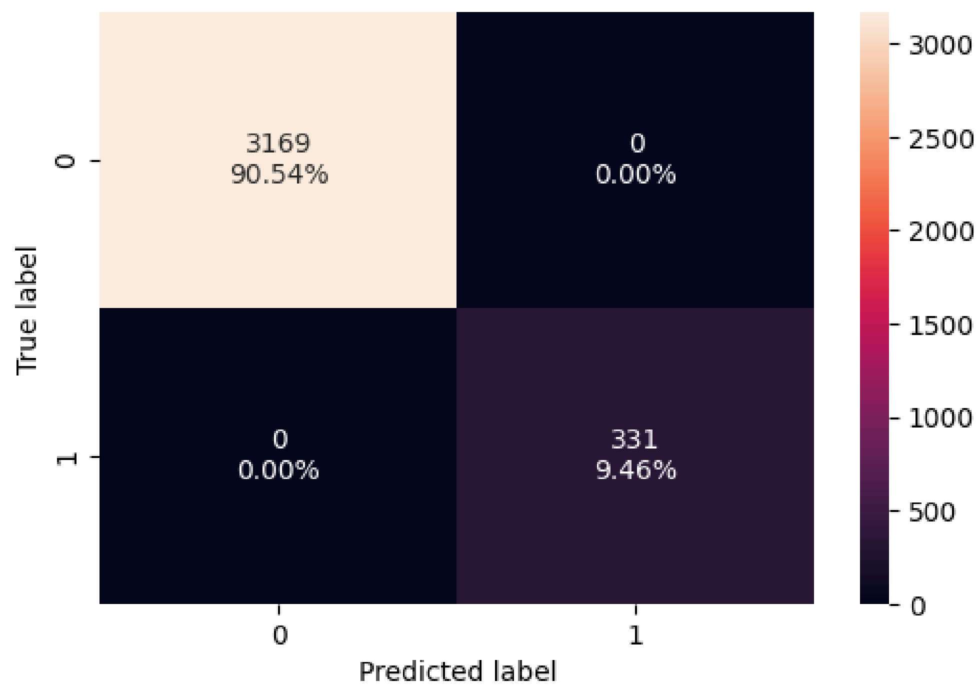
0.9.2 Decision Tree (sklearn default)

[134]:
```
model = DecisionTreeClassifier(criterion="gini", random_state=1)
model.fit(X_train, y_train)
```

[134]:
```
DecisionTreeClassifier(random_state=1)
```

Checking model performance on training data

[135]:
```
confusion_matrix_sklearn(model, X_train, y_train)
```



[136]:
```python
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, y_train
)
decision_tree_perf_train
```

[136]:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

1.0 for all values is a strong indicator of overfitting.

An overfitted model essentially memorizes the training data, including its noise and outliers. Instead of learning the underlying patterns that generalize to new, unseen data, it becomes too specialized to the training set.

While the model performs flawlessly on the data it has seen, it will likely perform poorly on new data (the test set).

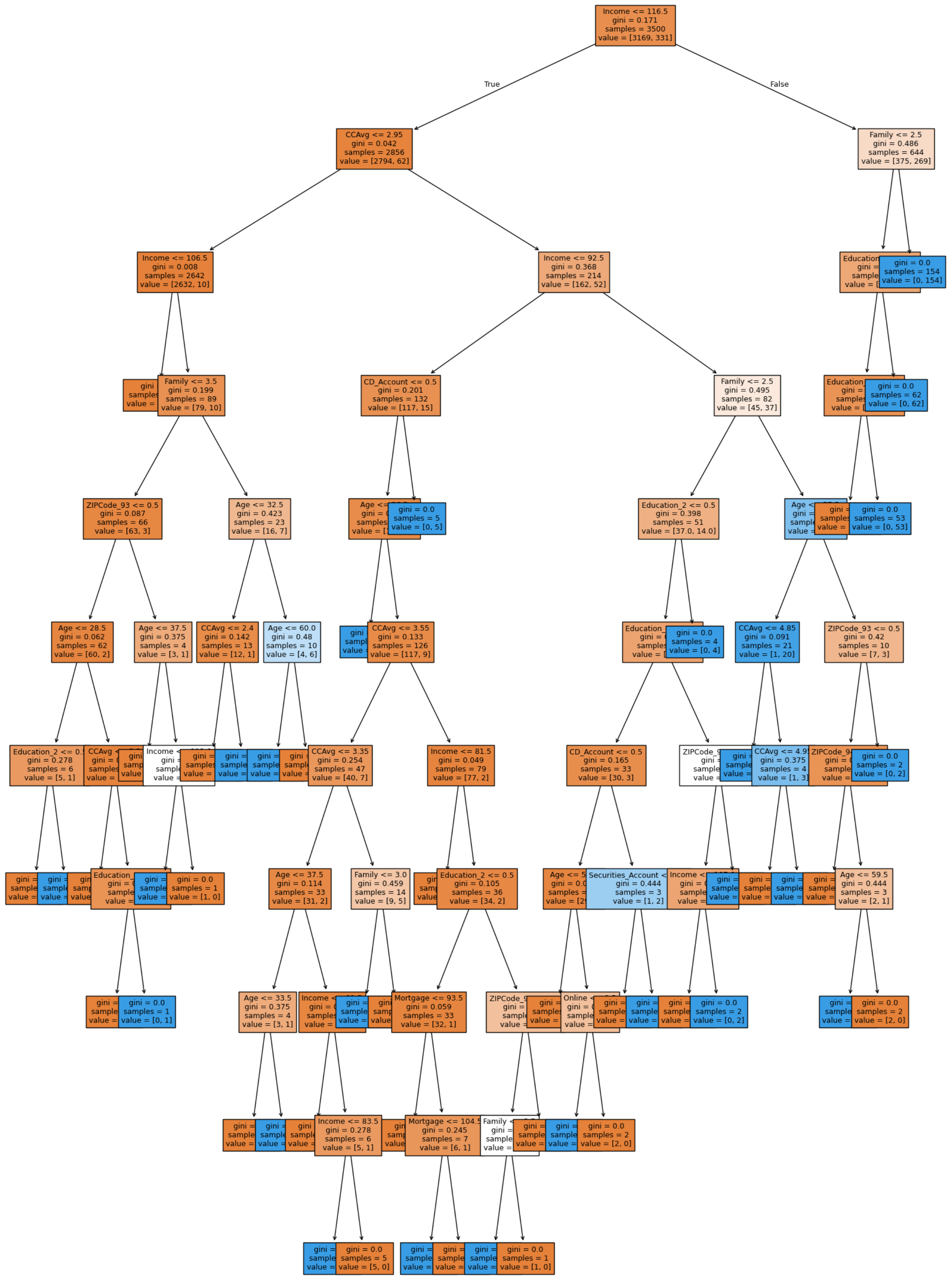
This means it lacks the ability to make accurate predictions in real-world scenarios.

Visualizing the Decision Tree

[137]:
```python
feature_names = list(X_train.columns)
print(feature_names)
```

```
['Age', 'Income', 'Family', 'CCAvg', 'Mortgage', 'Securities_Account',
'CD_Account', 'Online', 'CreditCard', 'ZIPCode_91', 'ZIPCode_92', 'ZIPCode_93',
'ZIPCode_94', 'ZIPCode_95', 'ZIPCode_96', 'Education_2', 'Education_3']
```

[138]:
```python
plt.figure(figsize=(20, 30))
out = tree.plot_tree(
    model,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# below code will add arrows to the decision tree split if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



[139]:
```python
# 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_2 <= 0.50
|   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |--- Education_2 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- Age >  28.50
|   |   |   |   |   |   |--- CCAvg <= 2.20
|   |   |   |   |   |   |   |--- weights: [48.00, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  2.20
|   |   |   |   |   |   |   |--- Education_3 <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Education_3 >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- ZIPCode_93 >  0.50
|   |   |   |   |   |--- Age <= 37.50
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |--- Age >  37.50
|   |   |   |   |   |   |--- Income <= 112.00
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- Income >  112.00
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- Family >  3.50
|   |   |   |   |--- Age <= 32.50
|   |   |   |   |   |--- CCAvg <= 2.40
|   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  2.40
|   |   |   |   |   |   |--- 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_2 <= 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_2 >  0.50
|   |   |   |   |   |   |   |   |--- ZIPCode_91 <= 0.50
|   |   |   |   |   |   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- Family >  3.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- ZIPCode_91 >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- CD_Account >  0.50
|   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |--- Income >  92.50
|   |   |   |--- Family <= 2.50
|   |   |   |   |--- Education_2 <= 0.50
|   |   |   |   |   |--- Education_3 <= 0.50
|   |   |   |   |   |   |--- CD_Account <= 0.50
```

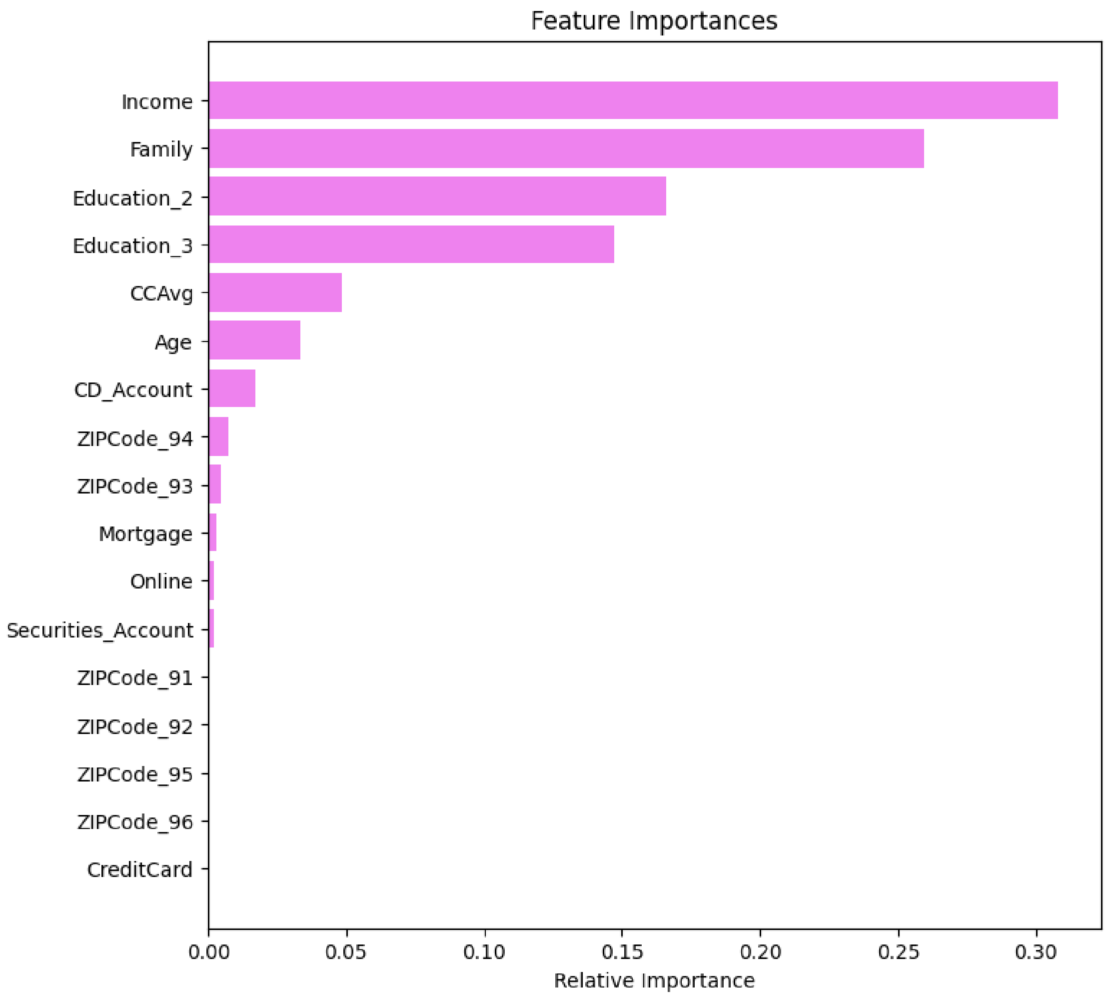
```
|   |   |   |   |   |   |   |--- Age <= 56.50
|   |   |   |   |   |   |   |   |--- weights: [27.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Age >  56.50
|   |   |   |   |   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- Online >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- CD_Account >  0.50
|   |   |   |   |   |   |   |--- Securities_Account <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Securities_Account >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |--- Education_3 >  0.50
|   |   |   |   |   |   |--- ZIPCode_94 <= 0.50
|   |   |   |   |   |   |   |--- Income <= 107.00
|   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Income >  107.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- ZIPCode_94 >  0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |--- Education_2 >  0.50
|   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |--- Family >  2.50
|   |   |   |   |--- Age <= 57.50
|   |   |   |   |   |--- CCAvg <= 4.85
|   |   |   |   |   |   |--- weights: [0.00, 17.00] class: 1
|   |   |   |   |   |--- CCAvg >  4.85
|   |   |   |   |   |   |--- CCAvg <= 4.95
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  4.95
|   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |--- Age >  57.50
|   |   |   |   |   |--- ZIPCode_93 <= 0.50
|   |   |   |   |   |   |--- ZIPCode_94 <= 0.50
|   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |--- ZIPCode_94 >  0.50
|   |   |   |   |   |   |   |--- Age <= 59.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- Age >  59.50
|   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |--- ZIPCode_93 >  0.50
|   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|--- Income >  116.50
|   |--- Family <= 2.50
|   |   |--- Education_3 <= 0.50
|   |   |   |--- Education_2 <= 0.50
|   |   |   |   |--- weights: [375.00, 0.00] class: 0
|   |   |   |--- Education_2 >  0.50
```

```
|   |   |   |   |--- weights: [0.00, 53.00] class: 1
|   |   |--- Education_3 >  0.50
|   |   |   |--- weights: [0.00, 62.00] class: 1
|   |--- Family >  2.50
|   |   |--- weights: [0.00, 154.00] class: 1
```

[140]:
```python
# importance of features in the tree building ( The importance of a feature is
 ↪computed as the
# (normalized) total reduction of the criterion brought by that feature. It is
 ↪also known as the Gini importance )

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

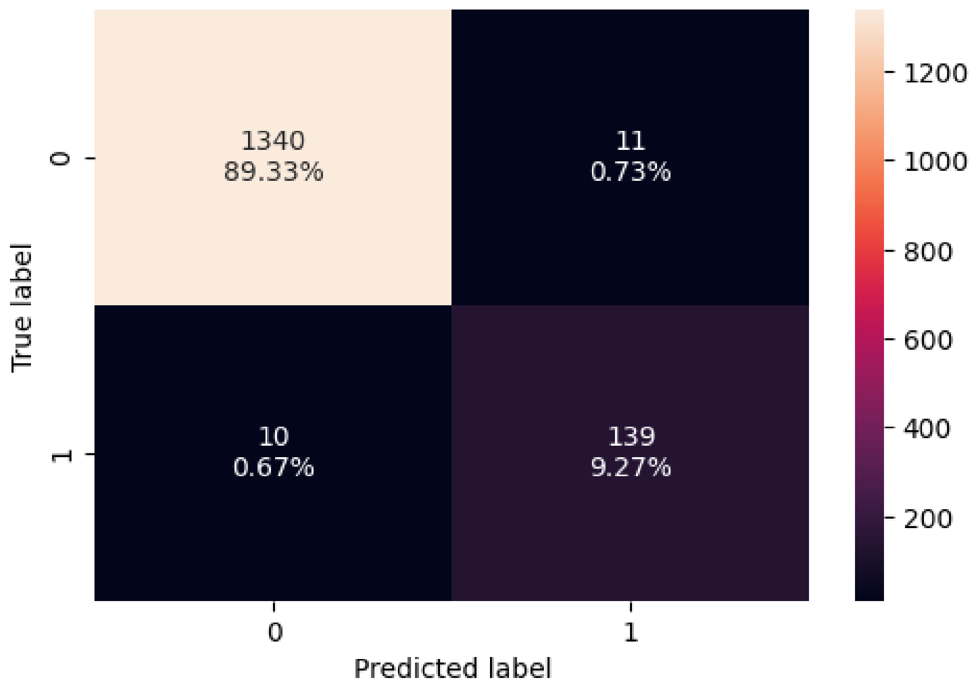
```
                        Imp
Income             0.308098
Family             0.259255
Education_2        0.166192
Education_3        0.147127
CCAvg              0.048798
Age                0.033150
CD_Account         0.017273
ZIPCode_94         0.007183
ZIPCode_93         0.004682
Mortgage           0.003236
Online             0.002224
Securities_Account 0.002224
ZIPCode_91         0.000556
ZIPCode_92         0.000000
ZIPCode_95         0.000000
ZIPCode_96         0.000000
CreditCard         0.000000
```

[141]:
```python
importances = model.feature_importances_
indices = np.argsort(importances)

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



Checking model performance on test data

[142]:
```
confusion_matrix_sklearn(model, X_test, y_test) ## Complete the code to create
 ↪confusion matrix for test data
```



[143]:
```
decision_tree_perf_test = model_performance_classification_sklearn(model,␣
    ↪X_test, y_test) ## Complete the code to check performance on test data
decision_tree_perf_test
```

[143]:

	Accuracy	Recall	Precision	F1
0	0.986	0.932886	0.926667	0.929766

0.10 Model Performance Improvement

Pre-pruning **Note**: The parameters provided below are a sample set. You can feel free to update the same and try out other combinations.

[144]:
```
# Define the parameters of the tree to iterate over
max_depth_values = np.arange(2, 7, 2)
max_leaf_nodes_values = [50, 75, 150, 250]
min_samples_split_values = [10, 30, 50, 70]

# Initialize variables to store the best model and its performance
best_estimator = None
best_score_diff = float('inf')
best_test_score = 0.0

# Iterate over all combinations of the specified parameter values
for max_depth in max_depth_values:
```

```
    for max_leaf_nodes in max_leaf_nodes_values:
        for min_samples_split in min_samples_split_values:

            # Initialize the tree with the current set of parameters
            estimator = DecisionTreeClassifier(
                max_depth=max_depth,
                max_leaf_nodes=max_leaf_nodes,
                min_samples_split=min_samples_split,
                class_weight='balanced',
                random_state=42
            )

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

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

            # Calculate recall scores for training and test sets
            train_recall_score = recall_score(y_train, y_train_pred)
            test_recall_score = recall_score(y_test, y_test_pred)

            # Calculate the absolute difference between training and test␣
 ↪recall scores
            score_diff = abs(train_recall_score - test_recall_score)

            # Update the best estimator and best score if the current one has a␣
 ↪smaller score difference
            if (score_diff < best_score_diff) & (test_recall_score >␣
 ↪best_test_score):
                best_score_diff = score_diff
                best_test_score = test_recall_score
                best_estimator = estimator

# Print the best parameters
print("Best parameters found:")
print(f"Max depth: {best_estimator.max_depth}")
print(f"Max leaf nodes: {best_estimator.max_leaf_nodes}")
print(f"Min samples split: {best_estimator.min_samples_split}")
print(f"Best test recall score: {best_test_score}")
```

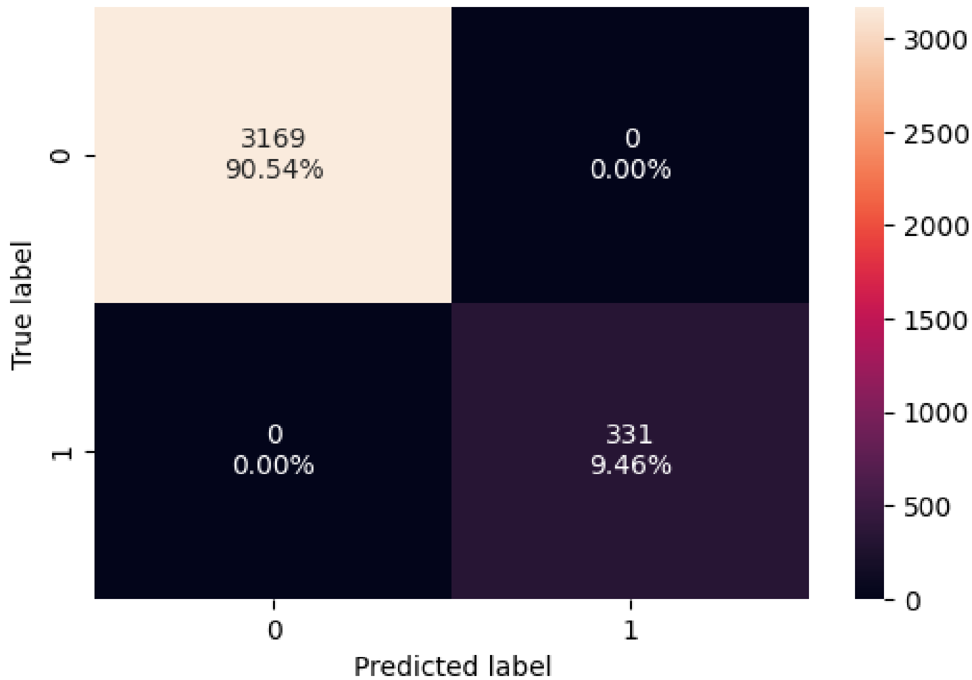
```
Best parameters found:
Max depth: 2
Max leaf nodes: 50
Min samples split: 10
Best test recall score: 1.0
```

```
[145]: # Fit the best algorithm to the data.
estimator = best_estimator
estimator.fit(X_train, y_train) ## Complete the code to fit model on train data
```

```
[145]: DecisionTreeClassifier(class_weight='balanced', max_depth=2, max_leaf_nodes=50,
                       min_samples_split=10, random_state=42)
```

Checking performance on training data

```
[146]: confusion_matrix_sklearn(model, X_train, y_train) ## Complete the code to
 ↪create confusion matrix for train data
```



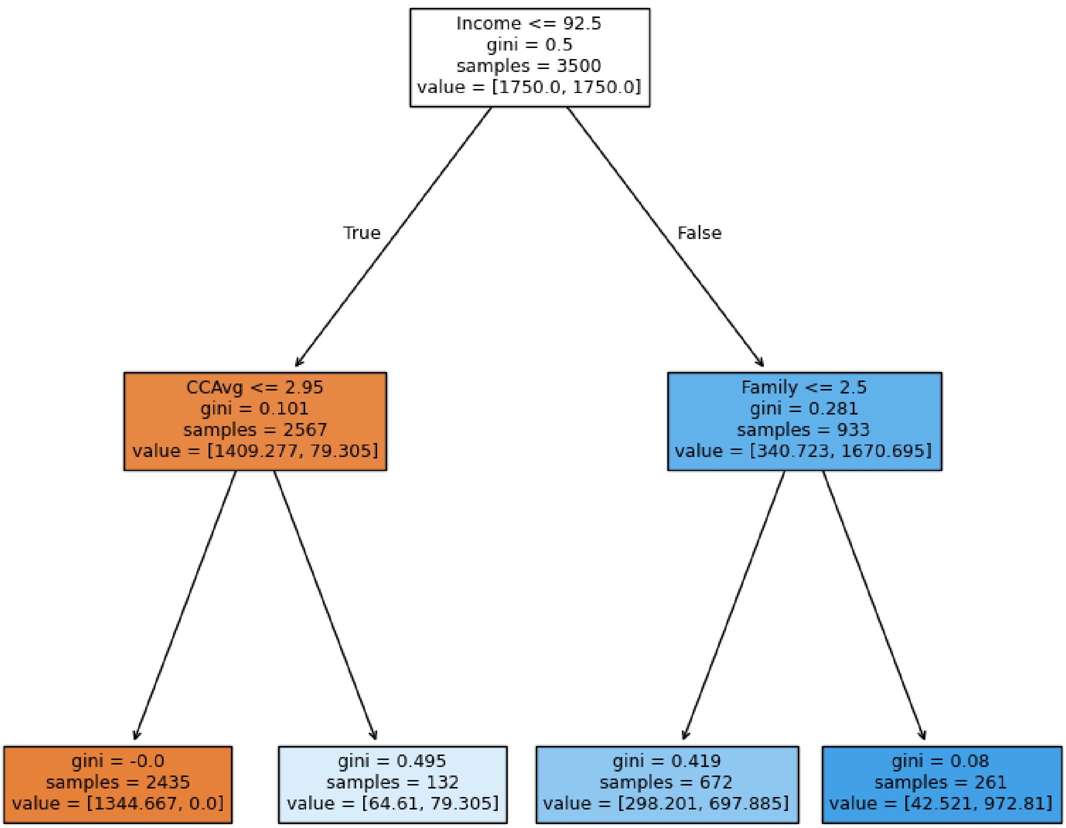

```
[147]: decision_tree_tune_perf_train = model_performance_classification_sklearn(model,
 ↪X_train, y_train) ## Complete the code to check performance on train data
decision_tree_tune_perf_train
```

```
[147]:    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

Visualizing the Decision Tree

```
[148]: plt.figure(figsize=(10, 10))
out = tree.plot_tree(
    estimator,
    feature_names=feature_names,
```

```
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# below code will add arrows to the decision tree split if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



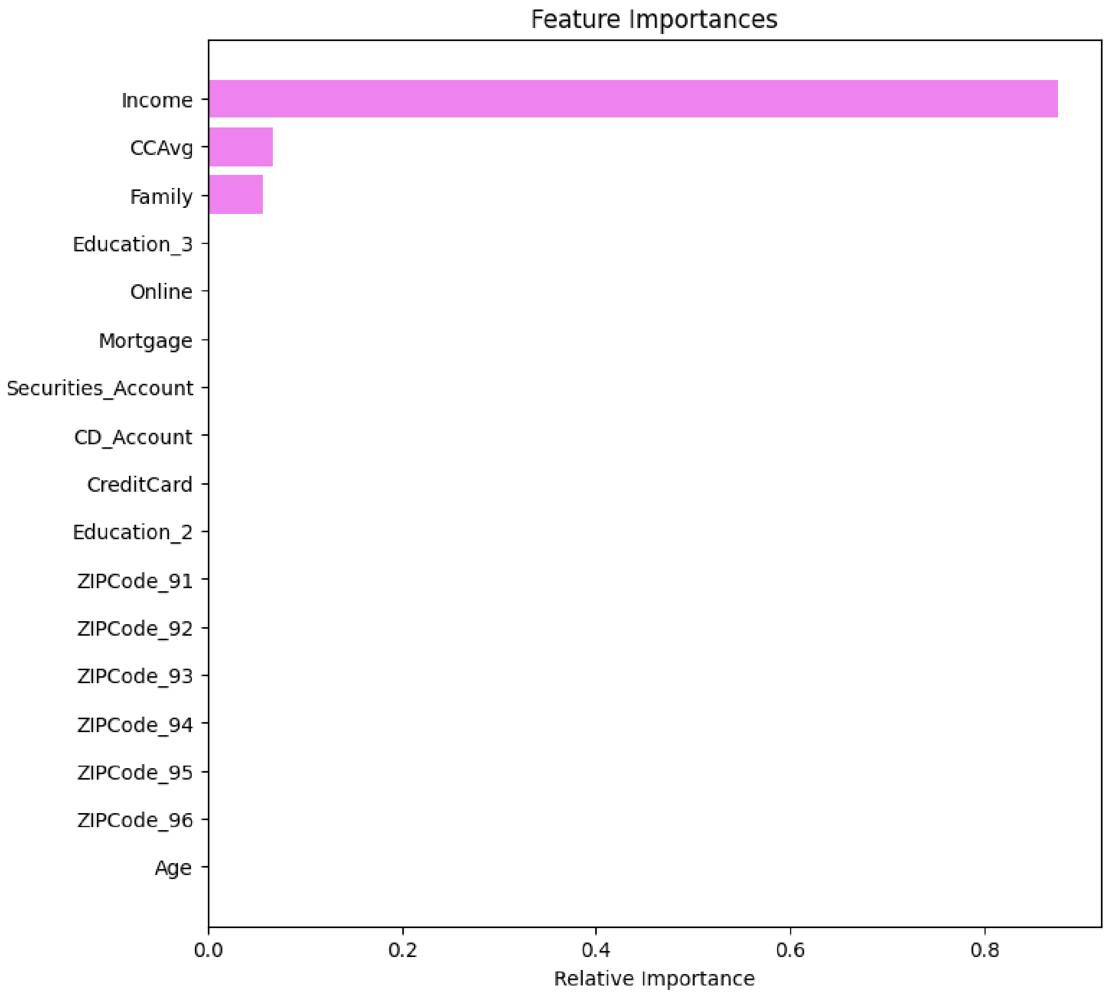
[149]:
```
# Text report showing the rules of a decision tree -

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

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [1344.67, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- weights: [64.61, 79.31] class: 1
|--- Income >  92.50
|   |--- Family <= 2.50
|   |   |--- weights: [298.20, 697.89] class: 1
|   |--- Family >  2.50
|   |   |--- weights: [42.52, 972.81] class: 1
```

[150]:
```
# importance of features in the tree building ( The importance of a feature is␣
 ↪computed as the
# (normalized) total reduction of the criterion brought by that feature. It is␣
 ↪also known as the Gini importance )

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

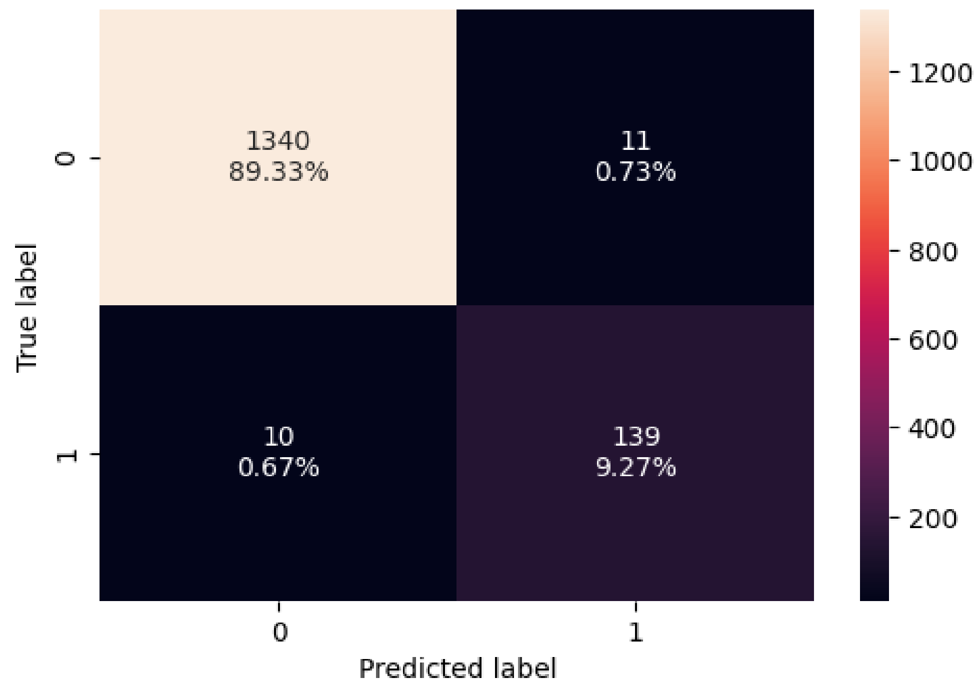
```
                        Imp
Income             0.876529
CCAvg              0.066940
Family             0.056531
Age                0.000000
ZIPCode_92         0.000000
Education_2        0.000000
ZIPCode_96         0.000000
ZIPCode_95         0.000000
ZIPCode_94         0.000000
ZIPCode_93         0.000000
CreditCard         0.000000
ZIPCode_91         0.000000
Online             0.000000
CD_Account         0.000000
Securities_Account 0.000000
Mortgage           0.000000
Education_3        0.000000
```

[151]:
```
importances = estimator.feature_importances_
indices = np.argsort(importances)

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



Checking performance on test data

[152]:
```
confusion_matrix_sklearn(model, X_test, y_test)  # Complete the code to get the␣
↪confusion matrix on test data
```



[153]:
```
decision_tree_tune_perf_test = model_performance_classification_sklearn(model,
    X_test, y_test) ## Complete the code to check performance on test data
decision_tree_tune_perf_test
```

[153]:

	Accuracy	Recall	Precision	F1
0	0.986	0.932886	0.926667	0.929766

Post-pruning

[154]:
```
clf = DecisionTreeClassifier(random_state=1)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

[155]:
```
pd.DataFrame(path)
```

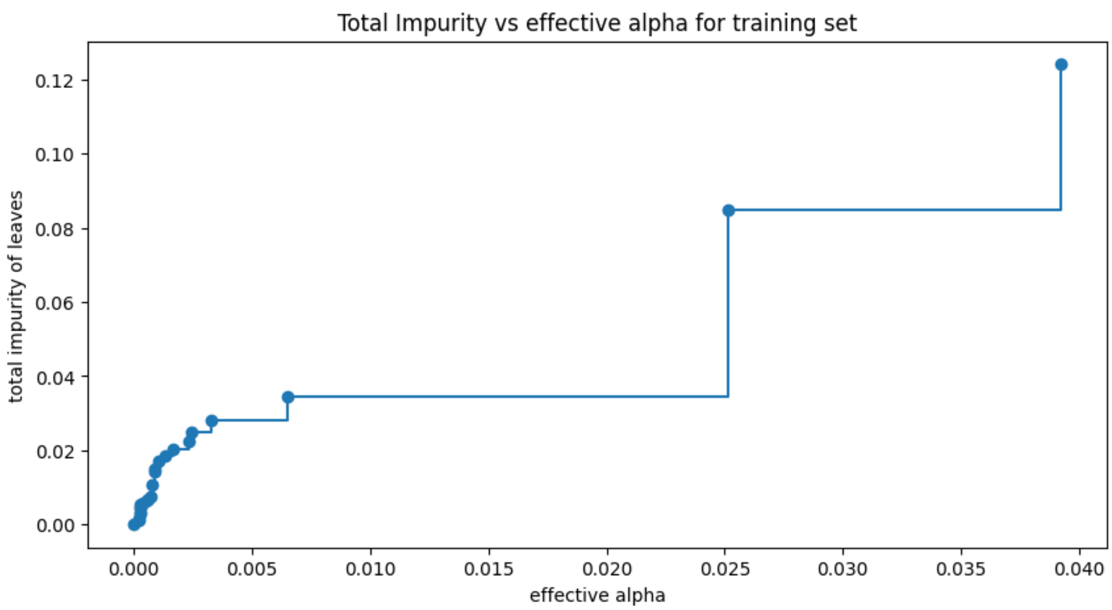
[155]:

	ccp_alphas	impurities
0	0.000000	0.000000
1	0.000186	0.001114
2	0.000214	0.001542
3	0.000242	0.002750
4	0.000250	0.003250
5	0.000268	0.004324
6	0.000272	0.004868
7	0.000276	0.005420
8	0.000381	0.005801

```
9      0.000527    0.006329
10     0.000625    0.006954
11     0.000700    0.007654
12     0.000769    0.010731
13     0.000882    0.014260
14     0.000889    0.015149
15     0.001026    0.017200
16     0.001305    0.018505
17     0.001647    0.020153
18     0.002333    0.022486
19     0.002407    0.024893
20     0.003294    0.028187
21     0.006473    0.034659
22     0.025146    0.084951
23     0.039216    0.124167
24     0.047088    0.171255
```

[156]:

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o", drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.show()
```

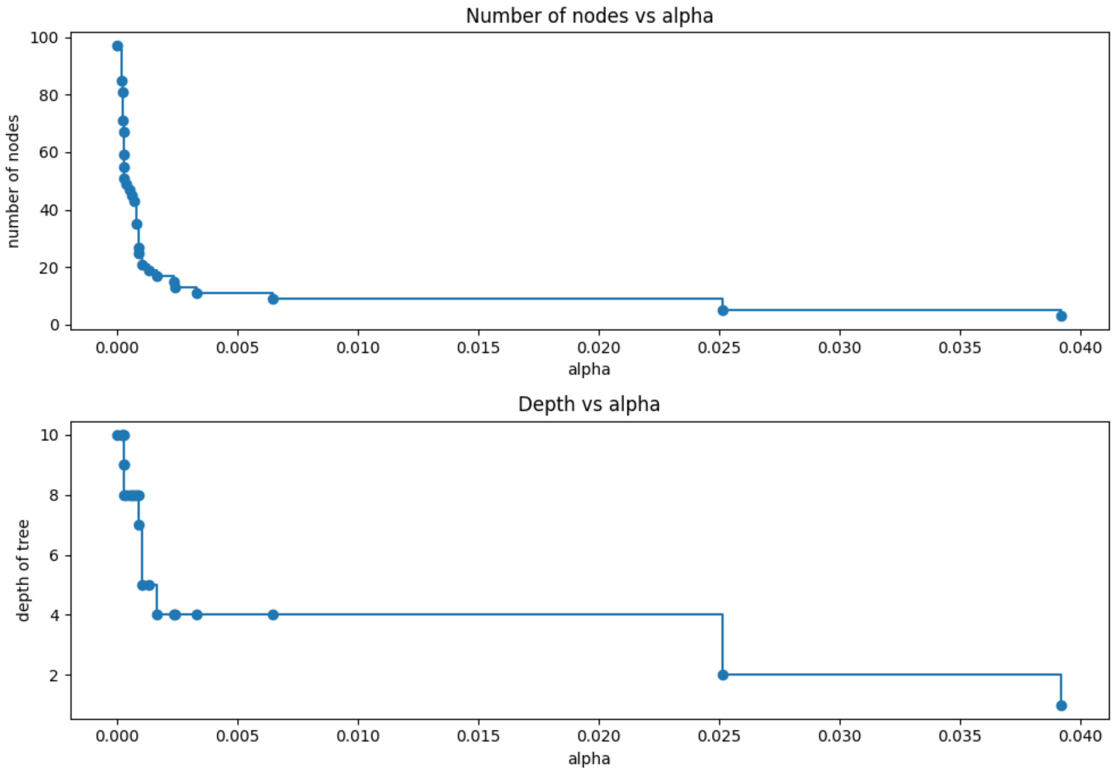


Next, we train a decision tree using effective alphas. The last value in `ccp_alphas` is the alpha value that prunes the whole tree, leaving the tree, `clfs[-1]`, with one node.

[157]:
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)      ## Complete the code to fit decision tree on␣
↪training data
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.047088341005967 66
```

[158]:
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

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

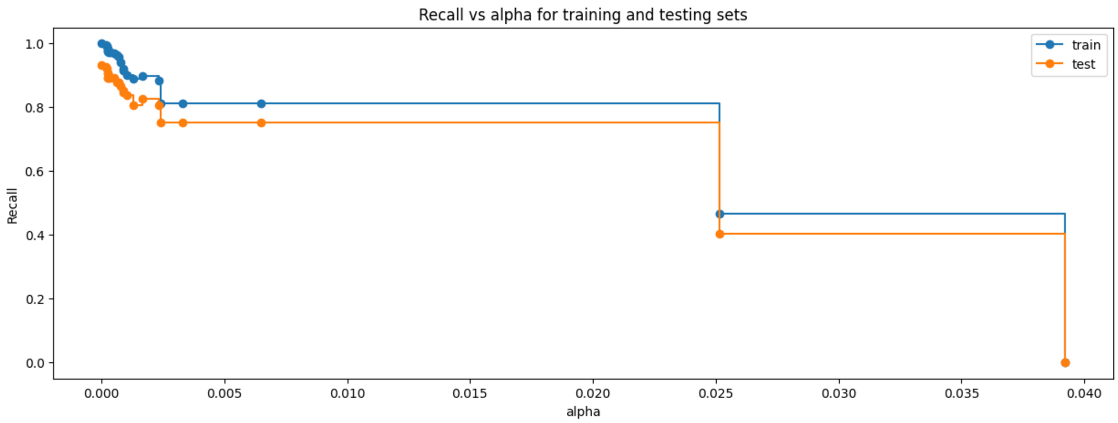


Recall vs alpha for training and testing sets

[159]:
```
recall_train = []
for clf in clfs:
    pred_train = clf.predict(X_train)
    values_train = recall_score(y_train, pred_train)
    recall_train.append(values_train)

recall_test = []
for clf in clfs:
    pred_test = clf.predict(X_test)
    values_test = recall_score(y_test, pred_test)
    recall_test.append(values_test)
```

[160]:
```
fig, ax = plt.subplots(figsize=(15, 5))
ax.set_xlabel("alpha")
ax.set_ylabel("Recall")
ax.set_title("Recall vs alpha for training and testing sets")
ax.plot(ccp_alphas, recall_train, marker="o", label="train",␣
 ↪drawstyle="steps-post")
ax.plot(ccp_alphas, recall_test, marker="o", label="test",␣
 ↪drawstyle="steps-post")
```

```
ax.legend()
plt.show()
```



[163]:
```
index_best_model = np.argmax(recall_test)
best_model = clfs[index_best_model]
print(best_model)
```

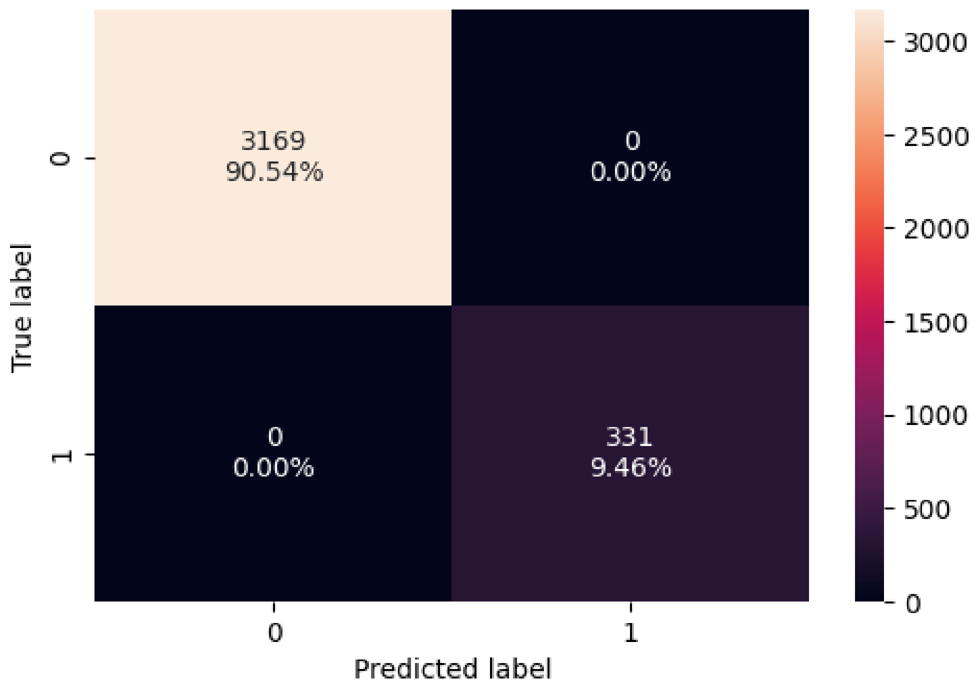
```
DecisionTreeClassifier(random_state=1)
```

[164]:
```
estimator_2 = DecisionTreeClassifier(
    ccp_alpha=.007, class_weight={0: 0.15, 1: 0.85}, random_state=1         ##␣
 ↪Complete the code by adding the correct ccp_alpha value
)
estimator_2.fit(X_train, y_train)
```

[164]:
```
DecisionTreeClassifier(ccp_alpha=0.007, class_weight={0: 0.15, 1: 0.85},
                       random_state=1)
```

Checking performance on training data

[166]:
```
confusion_matrix_sklearn(model, X_train, y_train) ## Complete the code to␣
 ↪create confusion matrix for train data
```



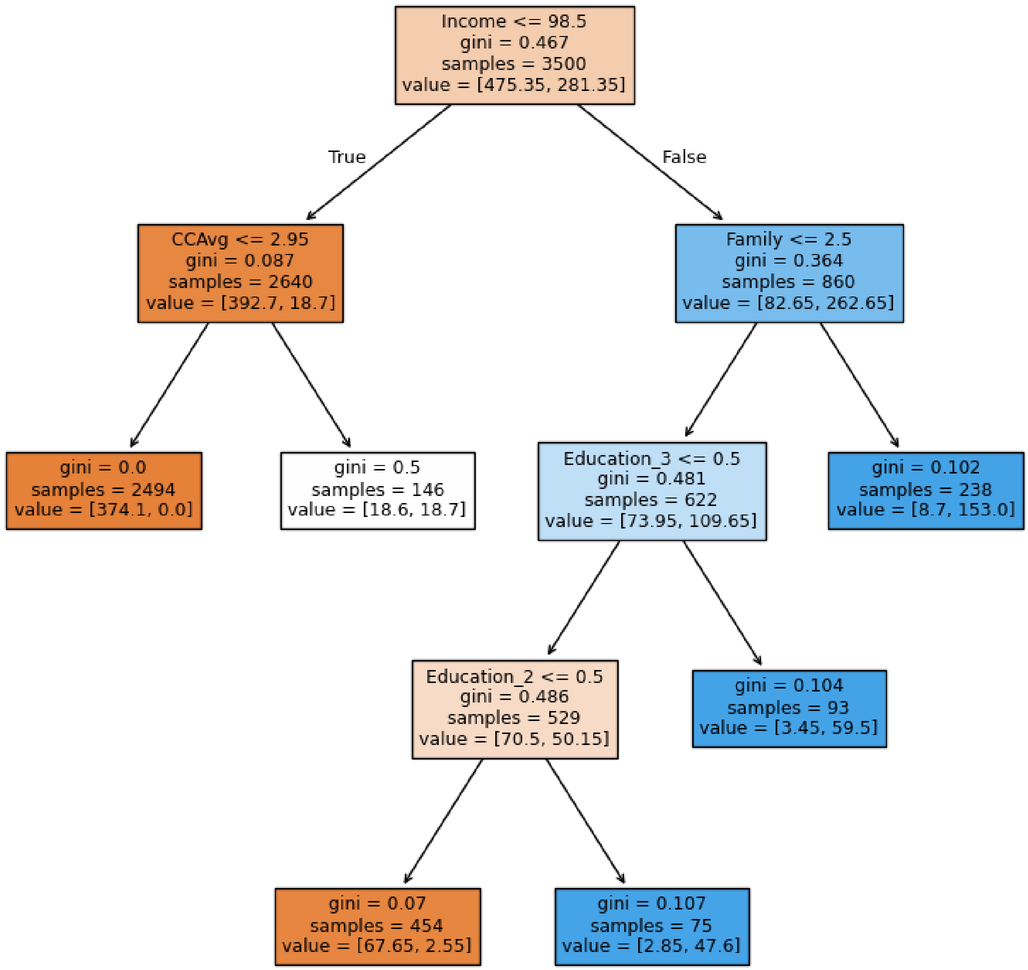
[167]:
```
decision_tree_tune_post_train = model_performance_classification_sklearn(model,
    X_train, y_train) ## Complete the code to check performance on train data
decision_tree_tune_post_train
```

[167]:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Visualizing the Decision Tree

[168]:
```
plt.figure(figsize=(10, 10))
out = tree.plot_tree(
    estimator_2,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# below code will add arrows to the decision tree split if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



[169]:

```
# 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
|   |   |--- weights: [18.60, 18.70] class: 1
|--- Income >  98.50
|   |--- Family <= 2.50
|   |   |--- Education_3 <= 0.50
```

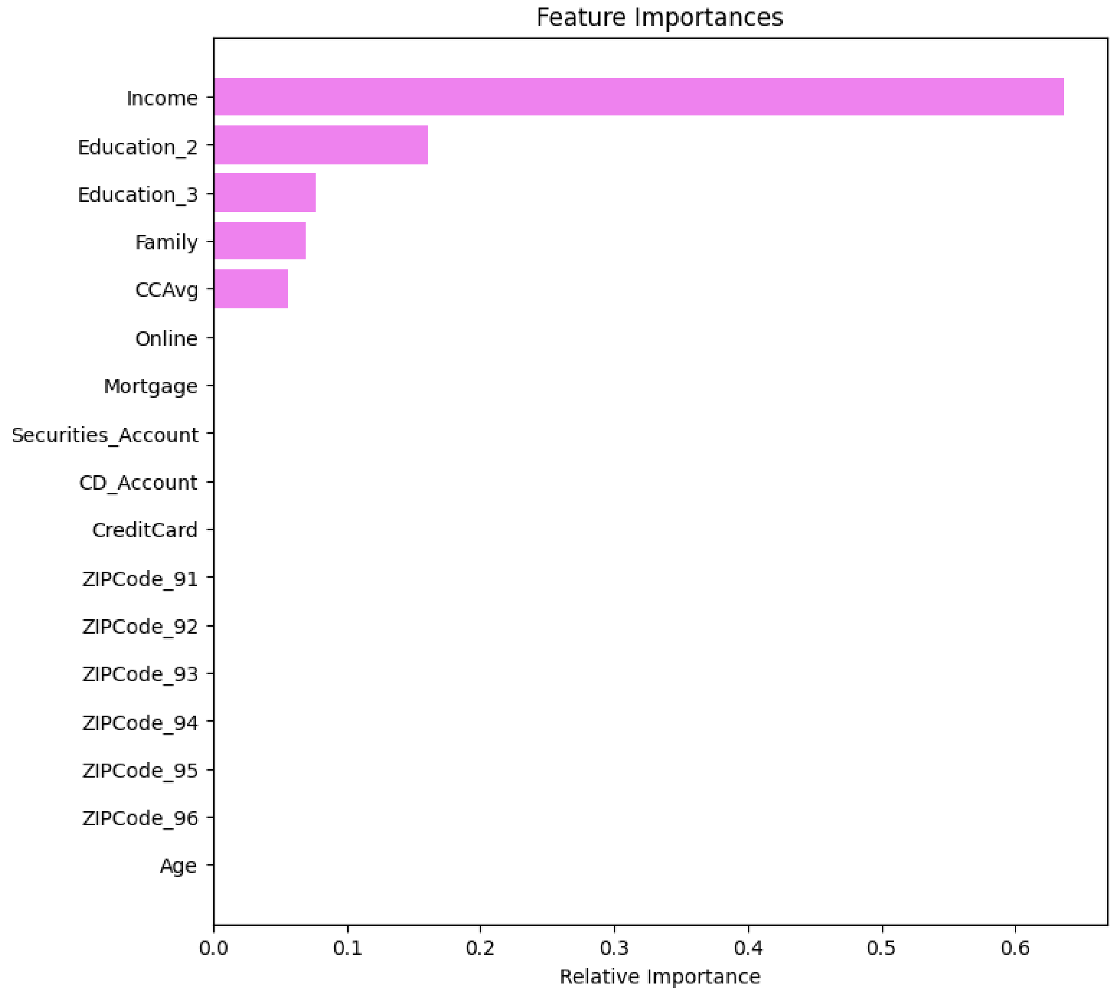
```
|   |   |   |--- Education_2 <= 0.50
|   |   |   |   |--- weights: [67.65, 2.55] class: 0
|   |   |   |--- Education_2 >  0.50
|   |   |   |   |--- weights: [2.85, 47.60] class: 1
|   |   |--- Education_3 >  0.50
|   |   |   |--- weights: [3.45, 59.50] class: 1
|   |--- Family >  2.50
|   |   |--- weights: [8.70, 153.00] class: 1
```

[170]:
```
# importance of features in the tree building ( The importance of a feature is␣
↪computed as the
# (normalized) total reduction of the criterion brought by that feature. It is␣
↪also known as the Gini importance )

print(
    pd.DataFrame(
        estimator_2.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                        Imp
Income             0.636860
Education_2        0.160224
Education_3        0.076930
Family             0.069445
CCAvg              0.056541
ZIPCode_92         0.000000
ZIPCode_96         0.000000
ZIPCode_95         0.000000
ZIPCode_94         0.000000
ZIPCode_93         0.000000
Age                0.000000
ZIPCode_91         0.000000
Online             0.000000
CD_Account         0.000000
Securities_Account 0.000000
Mortgage           0.000000
CreditCard         0.000000
```

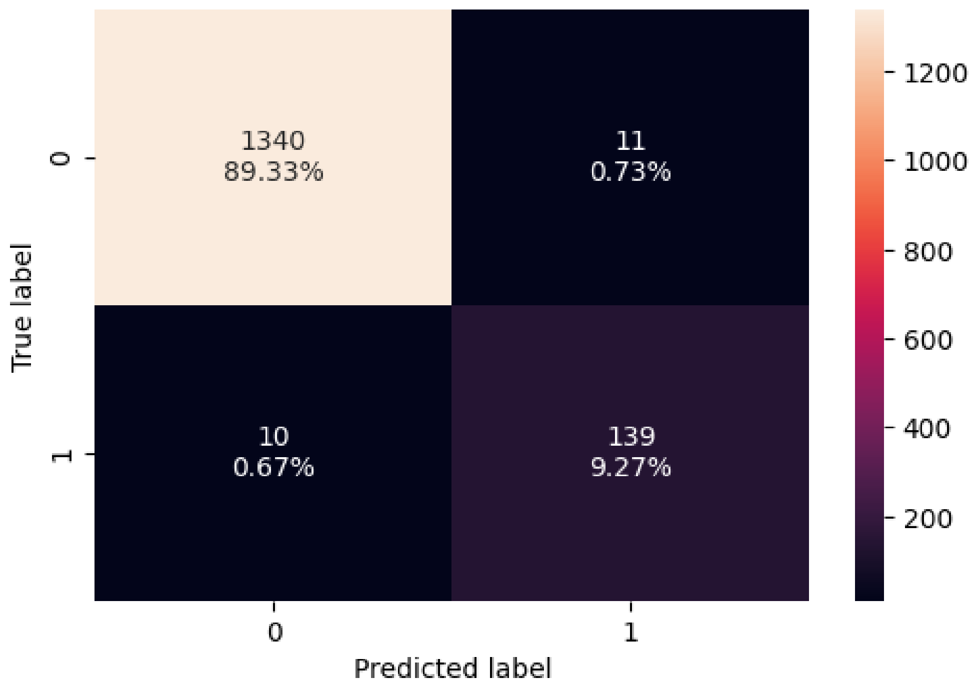
[171]:
```
importances = estimator_2.feature_importances_
indices = np.argsort(importances)

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

```
plt.xlabel("Relative Importance")
plt.show()
```



Checking performance on test data

[172]:
```
confusion_matrix_sklearn(model, X_test, y_test)  # Complete the code to get the␣
↪confusion matrix on test data
```



[174]:
```
decision_tree_tune_post_test = model_performance_classification_sklearn(model,␣
↪X_test, y_test) ## Complete the code to get the model performance on test␣
↪data
decision_tree_tune_post_test
```

[174]:

	Accuracy	Recall	Precision	F1
0	0.986	0.932886	0.926667	0.929766

0.11 Model Performance Comparison and Final Model Selection

[175]:
```
# training performance comparison

models_train_comp_df = pd.concat(
    [decision_tree_perf_train.T, decision_tree_tune_perf_train.T,␣
↪decision_tree_tune_post_train.T], axis=1,
)
models_train_comp_df.columns = ["Decision Tree (sklearn default)", "Decision␣
↪Tree (Pre-Pruning)", "Decision Tree (Post-Pruning)"]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

[175]:

```
           Decision Tree (sklearn default)  Decision Tree (Pre-Pruning)  \
Accuracy                               1.0                          1.0
Recall                                 1.0                          1.0
Precision                              1.0                          1.0
F1                                     1.0                          1.0

           Decision Tree (Post-Pruning)
Accuracy                            1.0
Recall                              1.0
Precision                           1.0
F1                                  1.0
```

[176]:

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [decision_tree_perf_test.T, decision_tree_tune_perf_test.T,␣
 ↪decision_tree_tune_post_test.T], axis=1,
)
models_test_comp_df.columns = ["Decision Tree (sklearn default)", "Decision␣
 ↪Tree (Pre-Pruning)", "Decision Tree (Post-Pruning)"]
print("Test set performance comparison:")
models_test_comp_df
```

```
Test set performance comparison:
```

[176]:

```
           Decision Tree (sklearn default)  Decision Tree (Pre-Pruning)  \
Accuracy                          0.986000                     0.986000
Recall                            0.932886                     0.932886
Precision                         0.926667                     0.926667
F1                                0.929766                     0.929766

           Decision Tree (Post-Pruning)
Accuracy                       0.986000
Recall                         0.932886
Precision                      0.926667
F1                             0.929766
```

0.12 Actionable Insights and Business Recommendations

What recommedations would you suggest to the bank?