Personal Loan Campaign

To predict whether a liability customer will buy a personal loan or not.

Which variables are most significant?

Which segment of customers should be targeted more?

In [1]:
```
%load_ext nb_black

import warnings

warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    plot_confusion_matrix,
    precision_recall_curve,
    roc_curve,
    make_scorer,
)
```

In [2]:
```
data = pd.read_csv("Loan_Modelling.csv")

df = data.copy()
```

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
df.tail()
```

Out[4]:

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

In [5]:
```
np.random.seed(2)
df.sample(10)
```

Out[5]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
3566	3567	57	33	80	92064	2	2.8	1	0	
4252	4253	54	29	81	91107	1	0.1	3	0	
1918	1919	39	9	118	93555	2	6.0	3	246	
4111	4112	43	17	21	95351	3	1.5	1	0	
1471	1472	52	26	180	94305	1	1.0	1	0	
929	930	55	30	22	92121	1	1.5	2	91	
1916	1917	57	32	64	95138	3	1.6	3	0	
3995	3996	53	28	34	92697	2	0.6	3	0	
1130	1131	58	32	191	94402	1	2.9	1	0	
1723	1724	39	15	55	95821	1	1.5	3	0	

In [6]:
```
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")
```

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

In [7]:
```
df[data.duplicated()].count()
```

Out[7]:
```
ID                    0
Age                   0
Experience            0
Income                0
```

```
ZIPCode              0
Family               0
CCAvg                0
Education            0
Mortgage             0
Personal_Loan        0
Securities_Account   0
CD_Account           0
Online               0
CreditCard           0
dtype: int64
```

In [8]:
```
df.info()
```

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

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
ID                  0
Age                 0
Experience          0
Income              0
ZIPCode             0
Family              0
CCAvg               0
Education           0
Mortgage            0
Personal_Loan       0
Securities_Account  0
CD_Account          0
Online              0
CreditCard          0
dtype: int64
```

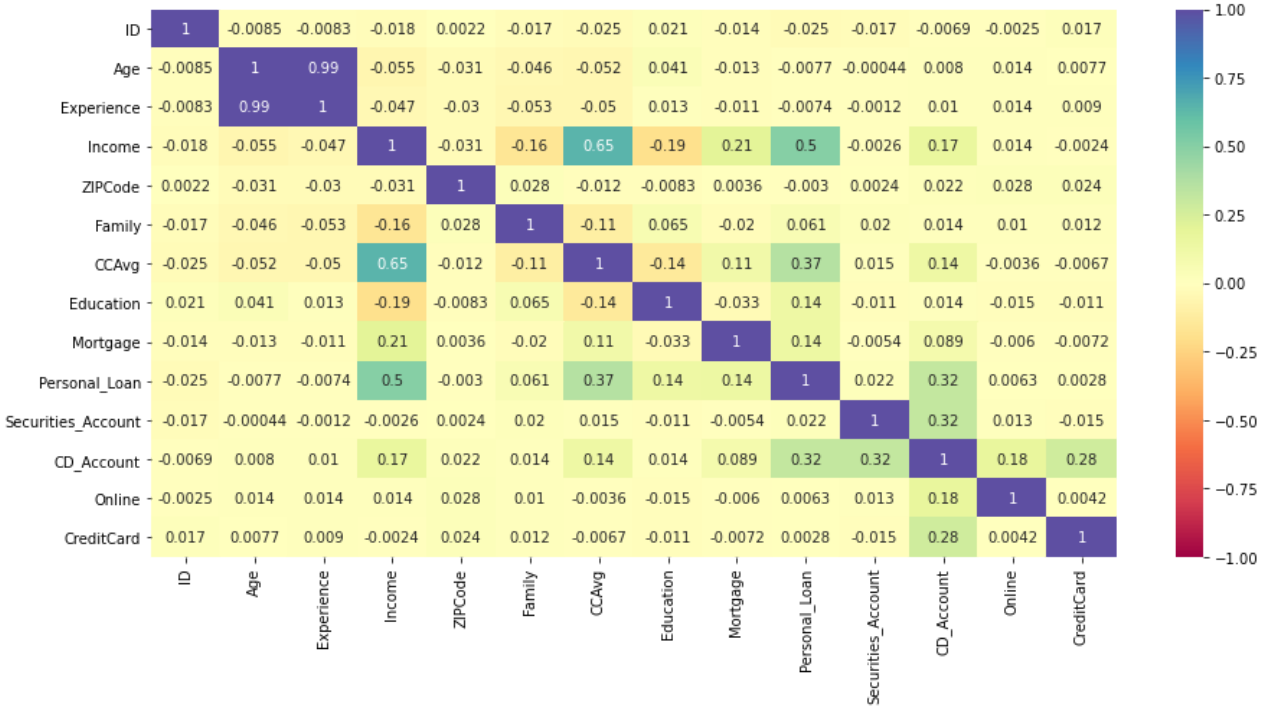
In [10]:
```
df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750.25	5(
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55.00	

	count	mean	std	min	25%	50%	75%	
Experience	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30.00	
Income	5000.0	73.774200	46.033729	8.0	39.00	64.0	98.00	2
ZIPCode	5000.0	93169.257000	1759.455086	90005.0	91911.00	93437.0	94608.00	96
Family	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	
CCAvg	5000.0	1.937938	1.747659	0.0	0.70	1.5	2.50	
Education	5000.0	1.881000	0.839869	1.0	1.00	2.0	3.00	
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101.00	6
Personal_Loan	5000.0	0.096000	0.294621	0.0	0.00	0.0	0.00	
Securities_Account	5000.0	0.104400	0.305809	0.0	0.00	0.0	0.00	
CD_Account	5000.0	0.060400	0.238250	0.0	0.00	0.0	0.00	
Online	5000.0	0.596800	0.490589	0.0	0.00	1.0	1.00	
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1.00	

In [11]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, cmap="Spectral")
plt.show()
```



In [12]:
```
my_tab = pd.crosstab(index=df["Age"], columns="count")
my_tab
```

Out[12]:

col_0	count
Age	
23	12

col_0	count
Age	
24	28
25	53
26	78
27	91
28	103
29	123
30	136
31	125
32	120
33	120
34	134
35	151
36	107
37	106
38	115
39	133
40	125
41	136
42	126
43	149
44	121
45	127
46	127
47	113
48	118
49	115
50	138
51	129
52	145
53	112
54	143
55	125
56	135
57	132

col_0	count
Age	
58	143
59	132
60	127
61	122
62	123
63	108
64	78
65	80
66	24
67	12

In [13]:
```
my_tab = pd.crosstab(index=df["Family"], columns="count")
my_tab
```

Out[13]:

col_0	count
Family	
1	1472
2	1296
3	1010
4	1222

In [14]:
```
my_tab = pd.crosstab(index=df["CCAvg"], columns="count")
my_tab
```

Out[14]:

col_0	count
CCAvg	
0.00	106
0.10	183
0.20	204
0.30	241
0.40	179
0.50	163
0.60	118
0.67	18
0.70	169
0.75	9

col_0	count
CCAvg	
0.80	187
0.90	106
1.00	231
1.10	84
1.20	66
1.30	128
1.33	9
1.40	136
1.50	178
1.60	101
1.67	18
1.70	158
1.75	9
1.80	152
1.90	106
2.00	188
2.10	100
2.20	130
2.30	58
2.33	18
2.40	92
2.50	107
2.60	87
2.67	36
2.70	58
2.75	1
2.80	110
2.90	54
3.00	53
3.10	20
3.20	22
3.25	1
3.30	45
3.33	1

col_0	count
CCAvg	
3.40	39
3.50	15
3.60	27
3.67	1
3.70	25
3.80	43
3.90	27
4.00	33
4.10	22
4.20	11
4.25	2
4.30	26
4.33	9
4.40	17
4.50	29
4.60	14
4.67	1
4.70	24
4.75	2
4.80	7
4.90	22
5.00	18
5.10	6
5.20	16
5.30	4
5.33	1
5.40	18
5.50	4
5.60	7
5.67	2
5.70	13
5.80	3
5.90	5
6.00	26

col_0	count
CCAvg	
6.10	14
6.20	2
6.30	13
6.33	10
6.40	3
6.50	18
6.60	4
6.67	9
6.70	9
6.80	10
6.90	14
7.00	14
7.20	13
7.30	10
7.40	13
7.50	12
7.60	9
7.80	9
7.90	4
8.00	12
8.10	10
8.20	1
8.30	2
8.50	2
8.60	8
8.80	9
8.90	1
9.00	2
9.30	1
10.00	3

In [15]:
```
my_tab = pd.crosstab(index=df["Education"], columns="count")
my_tab
```

Out[15]:

Education	col_0 count
1	2096
2	1403
3	1501

In [16]:
```
my_tab = pd.crosstab(index=df["Experience"], columns="count")
my_tab
```

Out[16]:

Experience	col_0 count
-3	4
-2	15
-1	33
0	66
1	74
2	85
3	129
4	113
5	146
6	119
7	121
8	119
9	147
10	118
11	116
12	102
13	117
14	127
15	119
16	127
17	125
18	137
19	135
20	148
21	113
22	124

Experience	col_0 count
23	144
24	131
25	142
26	134
27	125
28	138
29	124
30	126
31	104
32	154
33	117
34	125
35	143
36	114
37	116
38	88
39	85
40	57
41	43
42	8
43	3

In [17]:
```
my_tab = pd.crosstab(index=df["Personal_Loan"], columns="count")
my_tab
```

Out[17]:

Personal_Loan	col_0 count
0	4520
1	480

In [18]:
```
my_tab = pd.crosstab(index=df["Securities_Account"], columns="count")
my_tab
```

Out[18]:

Securities_Account	col_0 count
0	4478

Securities_Account	col_0 count
1	522

In [19]:
```
my_tab = pd.crosstab(index=df["CD_Account"], columns="count")
my_tab
```

Out[19]:

CD_Account	col_0 count
0	4698
1	302

In [20]:
```
my_tab = pd.crosstab(index=df["Online"], columns="count")
my_tab
```

Out[20]:

Online	col_0 count
0	2016
1	2984

In [21]:
```
my_tab = pd.crosstab(index=df["CreditCard"], columns="count")
my_tab
```

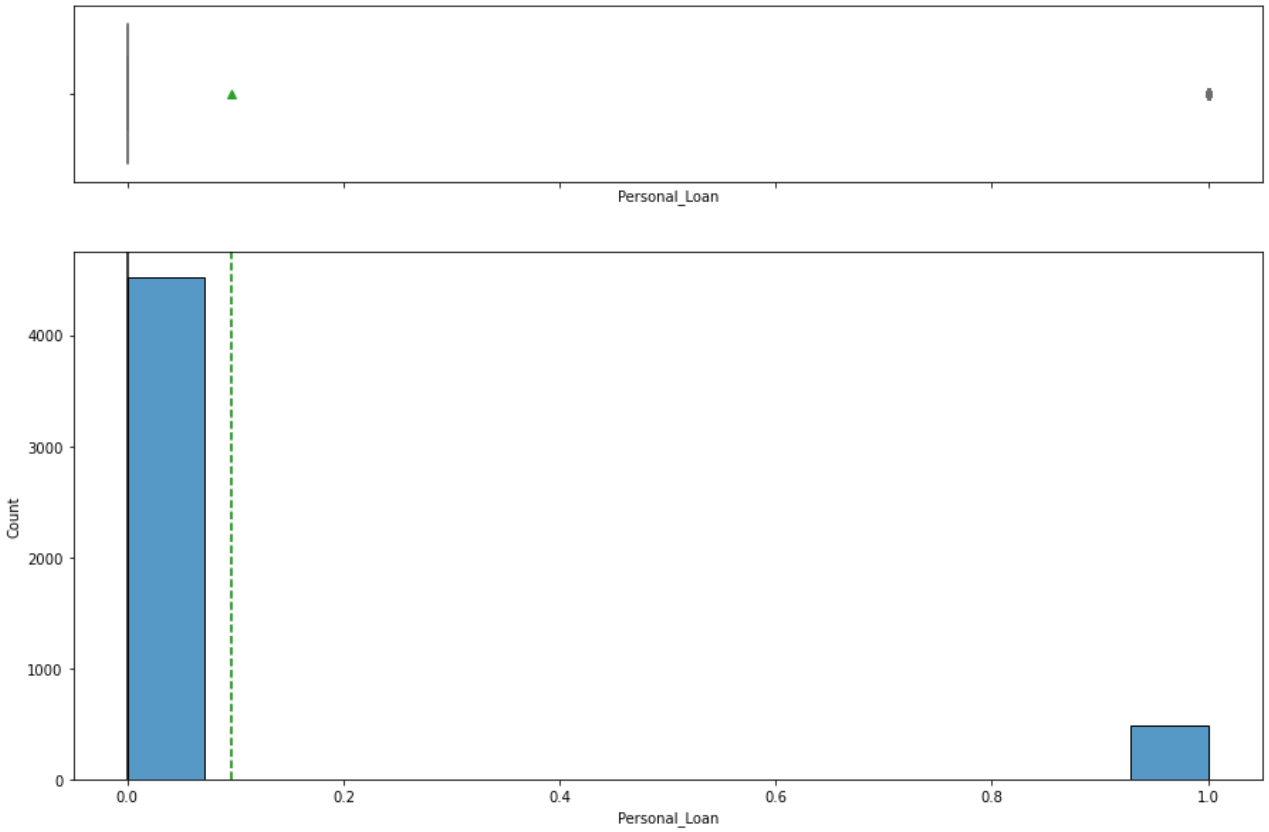
Out[21]:

CreditCard	col_0 count
0	3530
1	1470

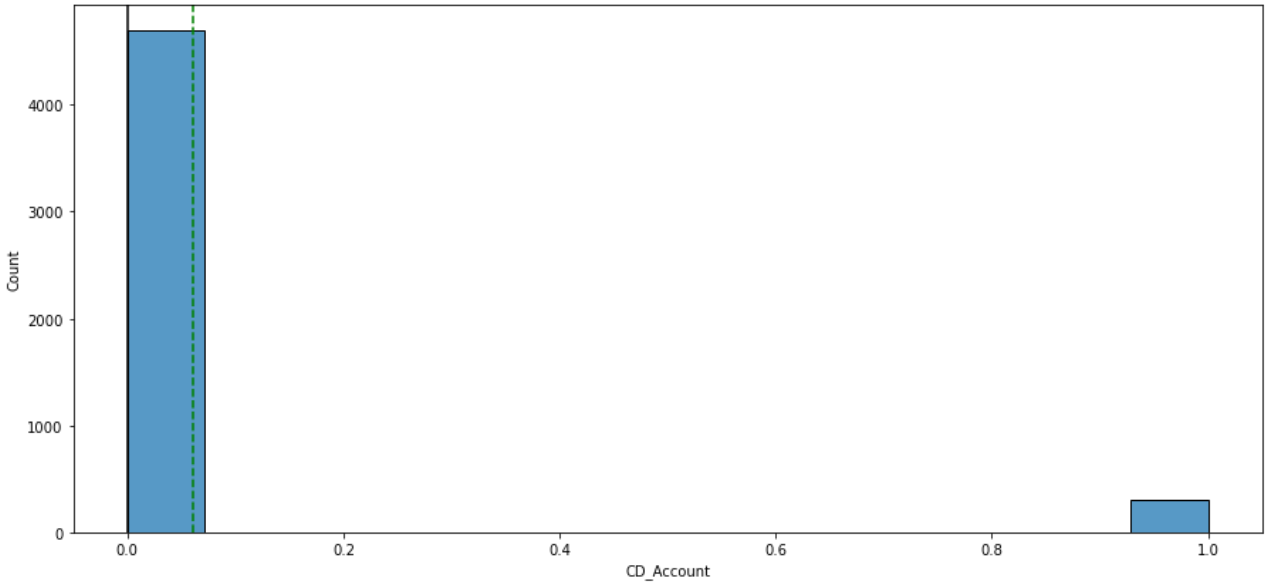
In [22]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    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,
        sharex=True,
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )
    sns.boxplot(data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
```

```
    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)
    ax_hist2.axvline(data[feature].mean(), color="green", linestyle="--")
    ax_hist2.axvline(data[feature].median(), color="black", linestyle="-")
```

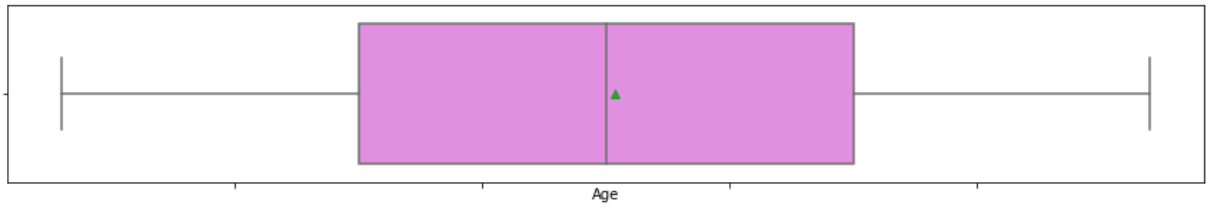
In [23]:
```
histogram_boxplot(data, "Personal_Loan")
```

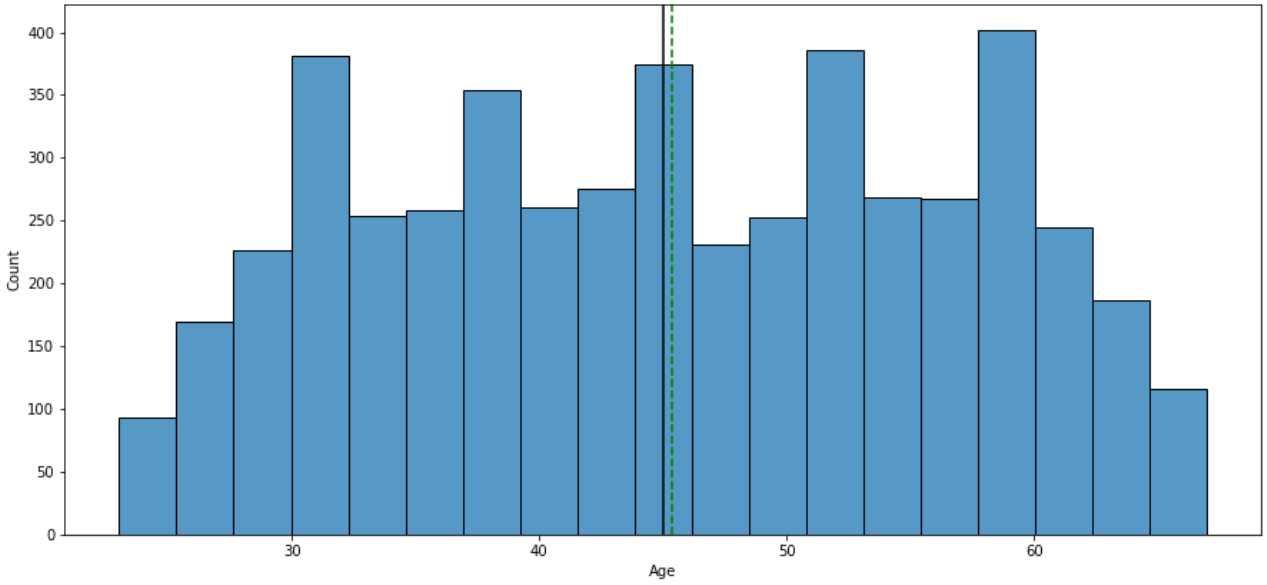


In [24]:
```
histogram_boxplot(data, "CD_Account")
```

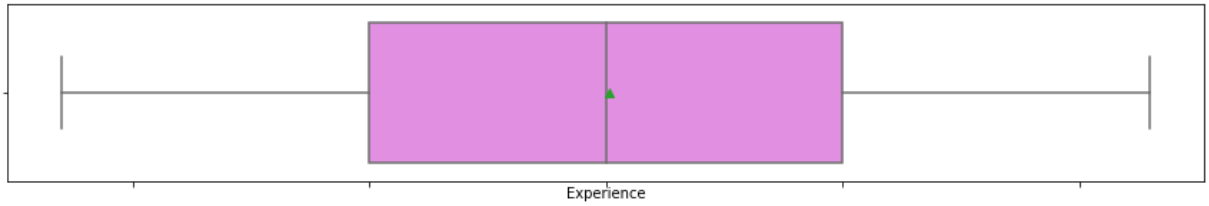


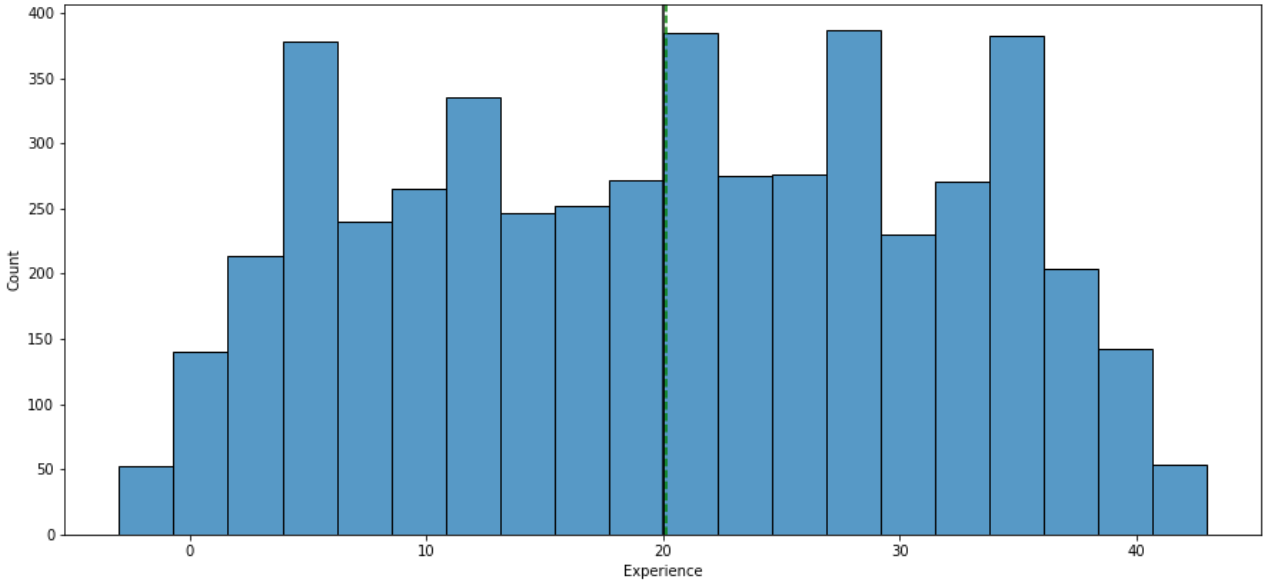
In [25]:
```
histogram_boxplot(data, "Age")
```



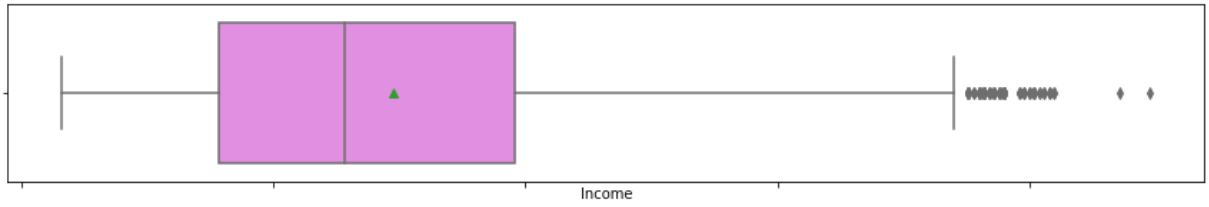


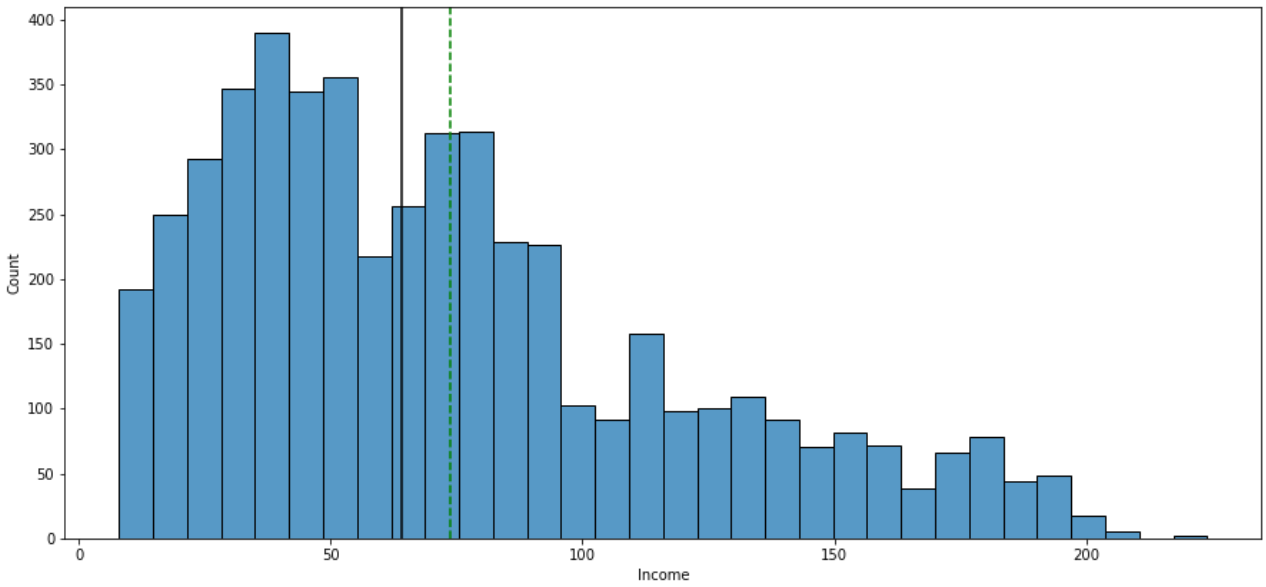
In [26]:
```
histogram_boxplot(data, "Experience")
```



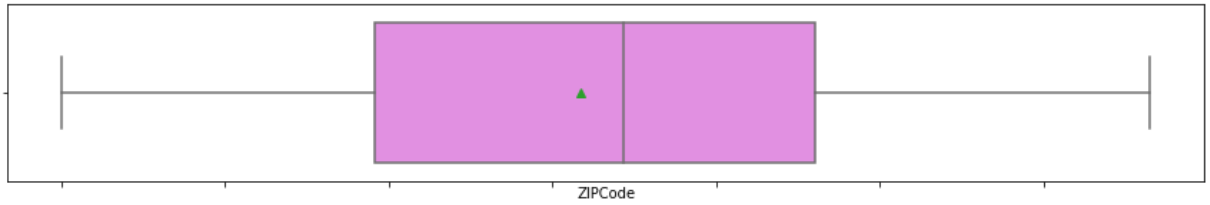


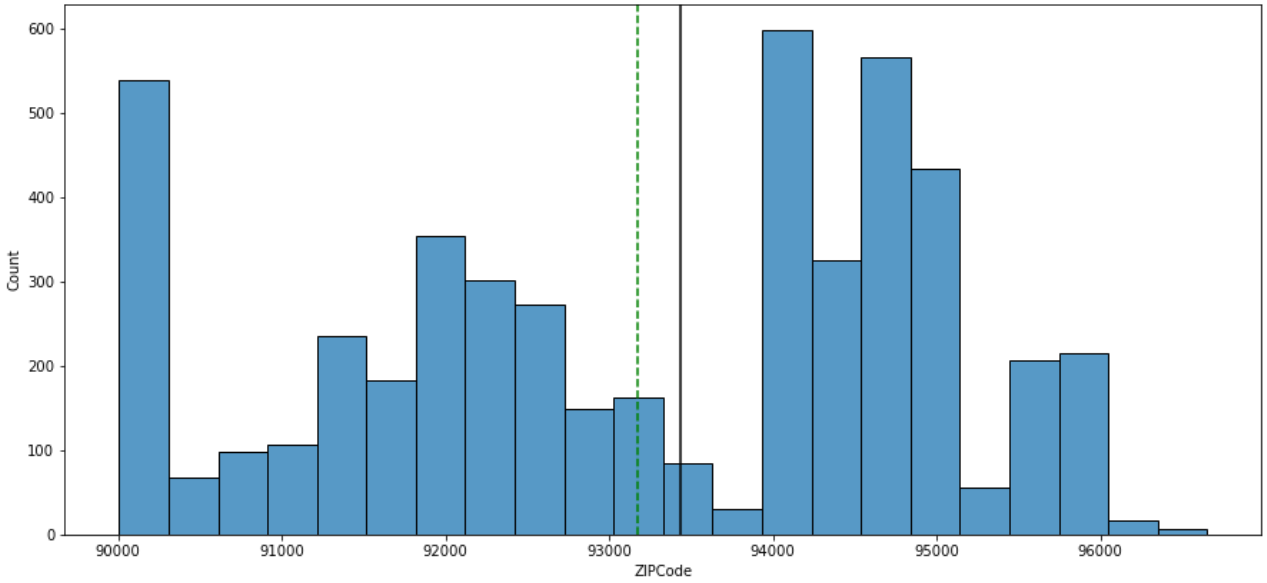
In [27]:
```
histogram_boxplot(data, "Income")
```



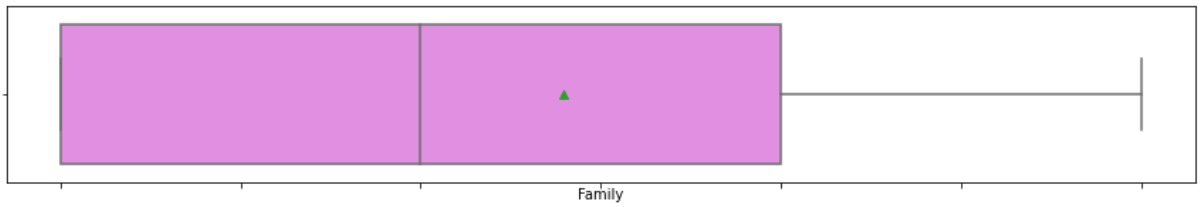


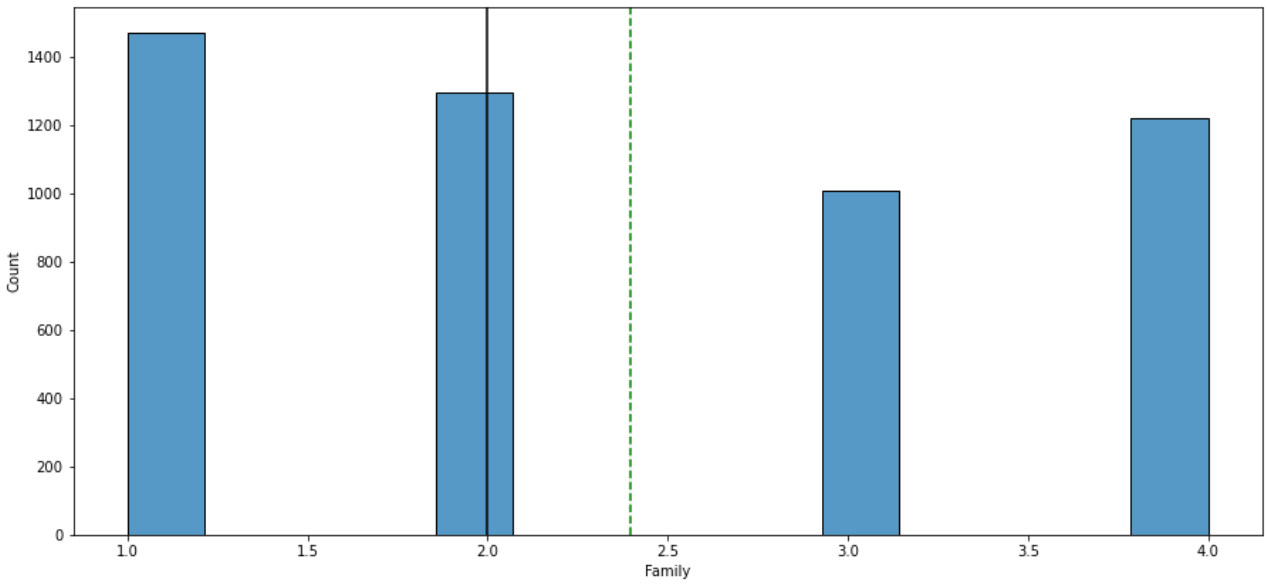
In [28]:
```
histogram_boxplot(data, "ZIPCode")
```



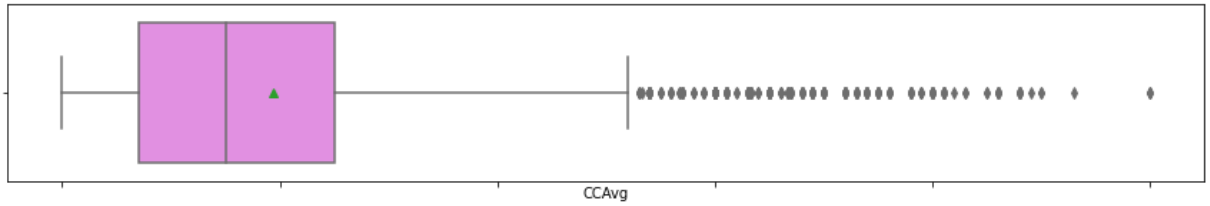


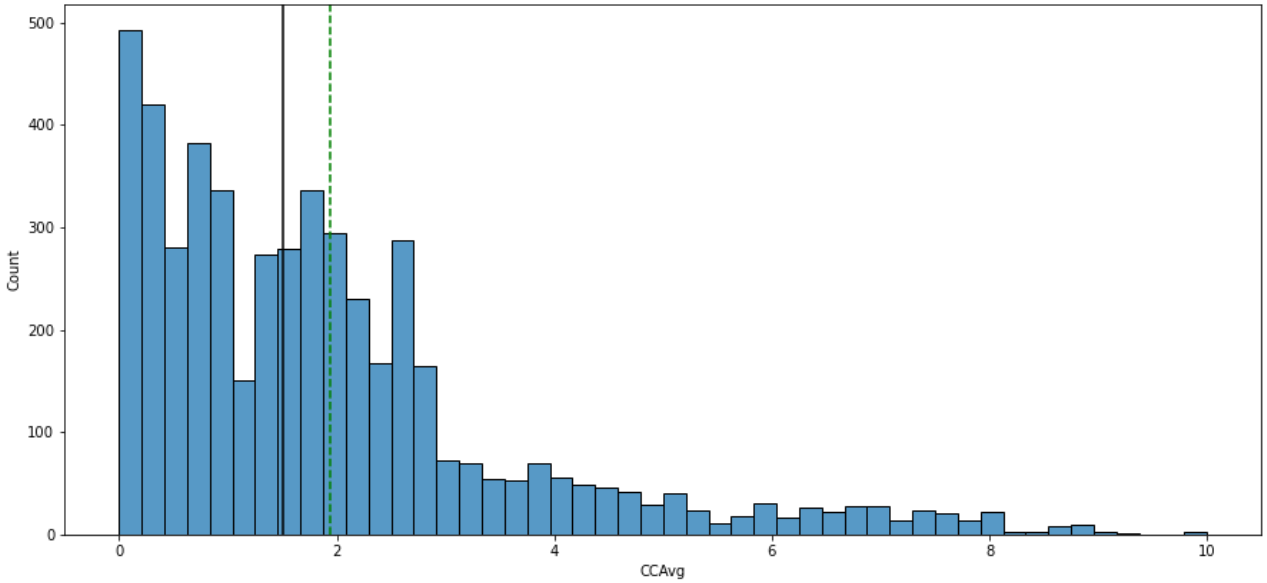
In [29]:
```
histogram_boxplot(data, "Family")
```



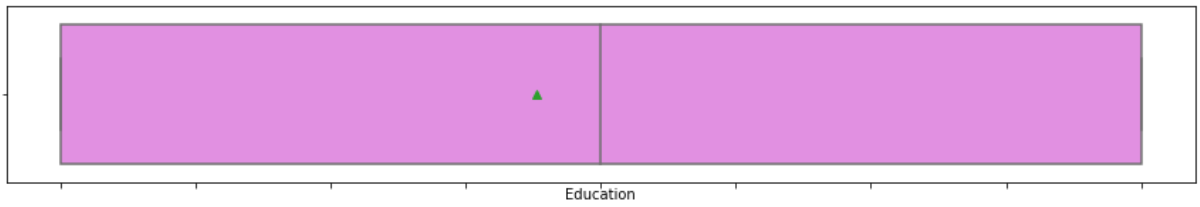


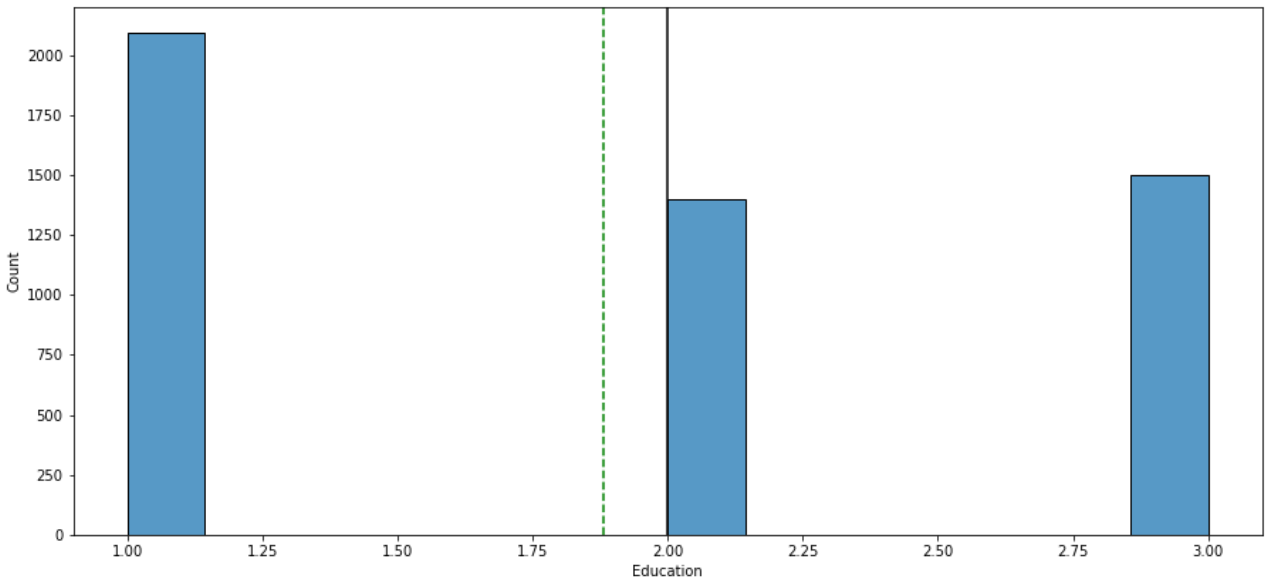
In [30]:
```
histogram_boxplot(data, "CCAvg")
```



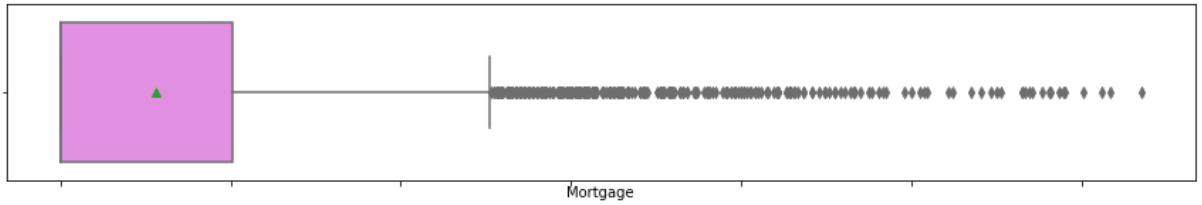


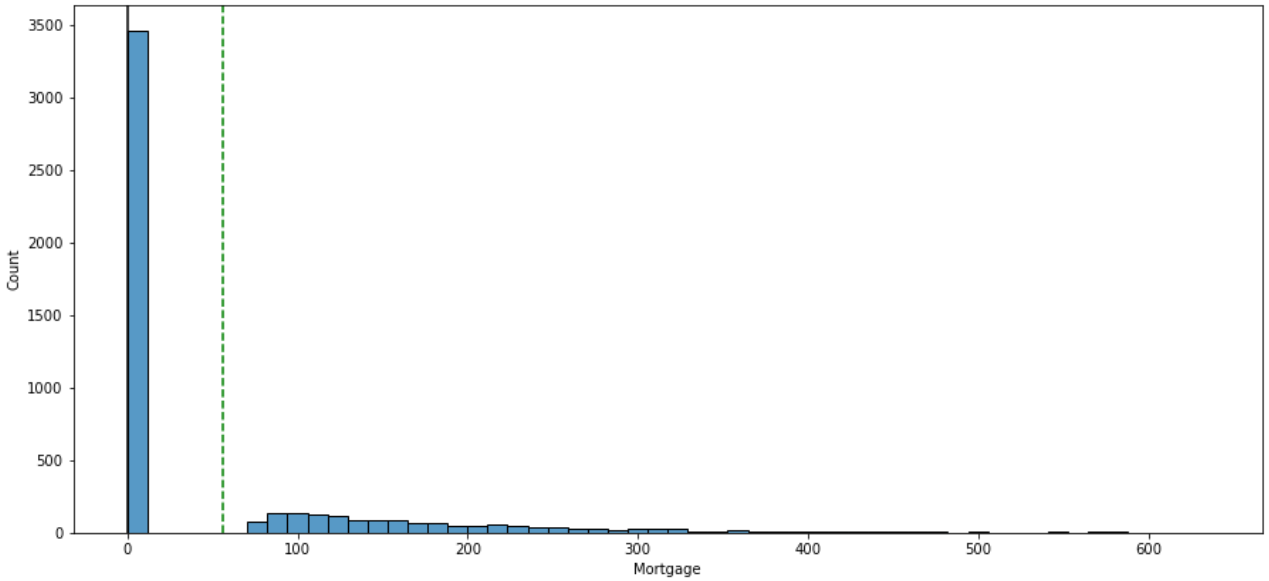
In [31]:
```
histogram_boxplot(data, "Education")
```



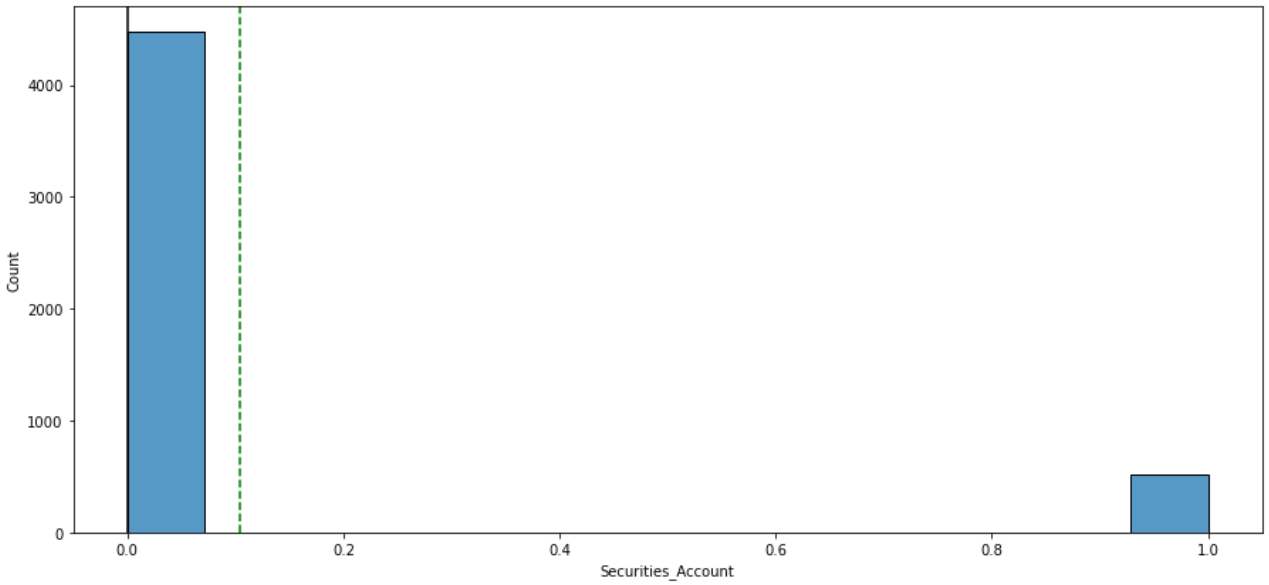


In [32]:
```
histogram_boxplot(data, "Mortgage")
```

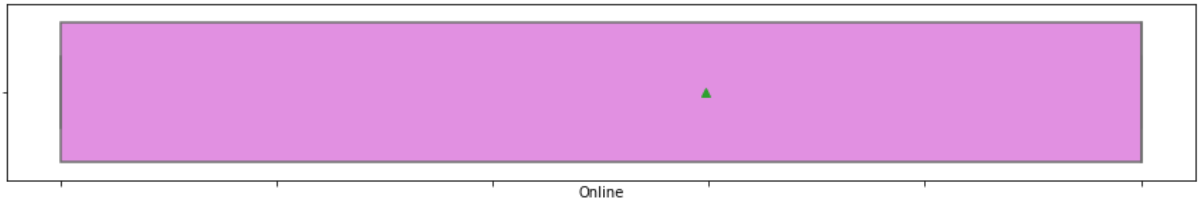


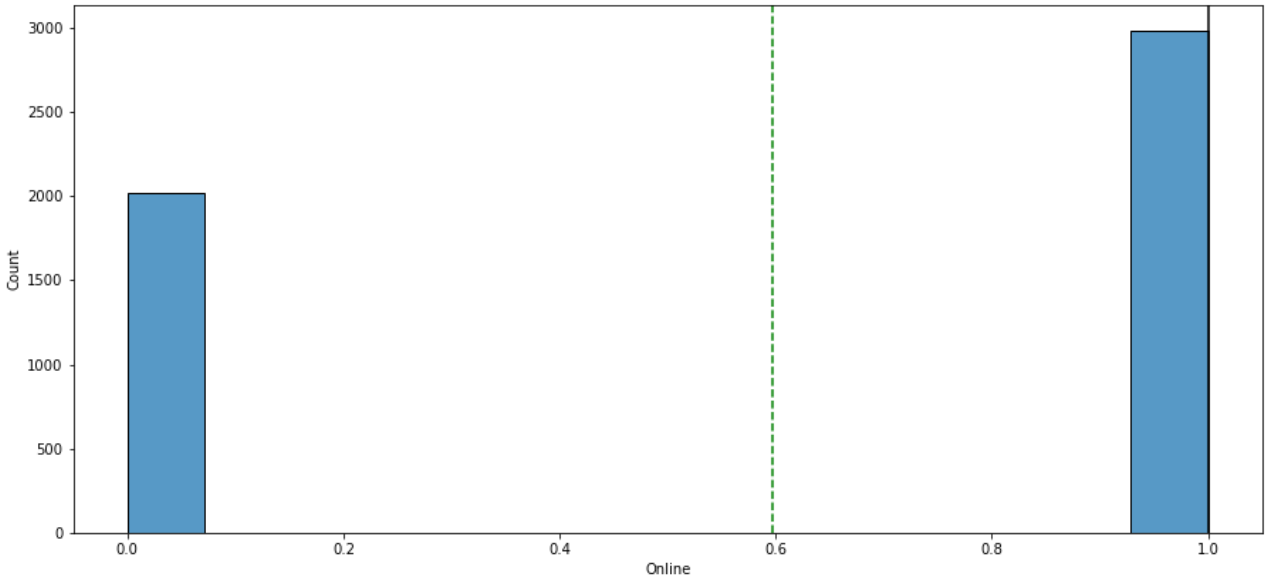


In [33]:
```
histogram_boxplot(data, "Securities_Account")
```

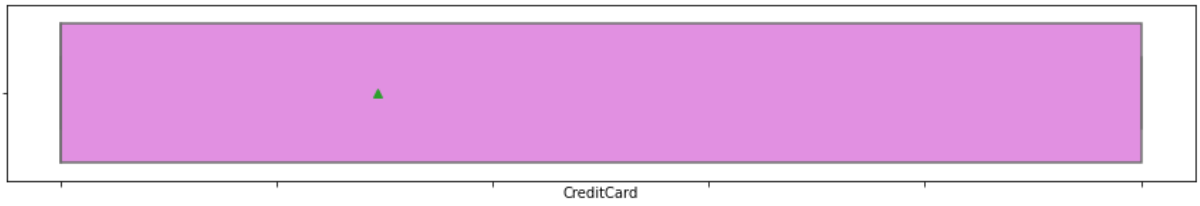


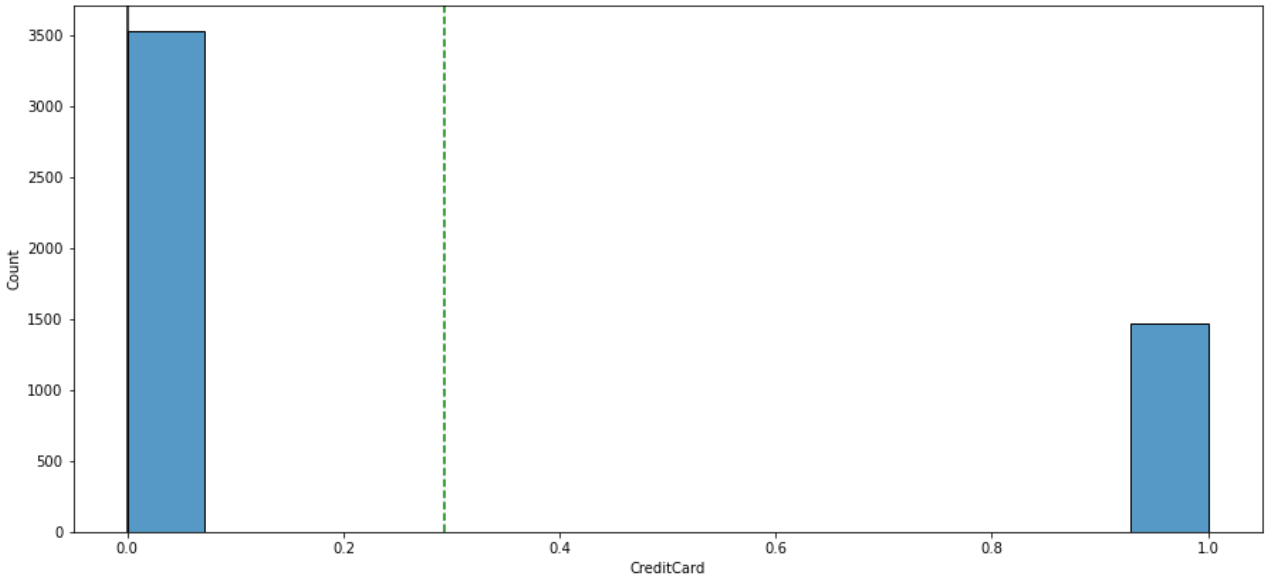
In [34]:
```
histogram_boxplot(data, "Online")
```





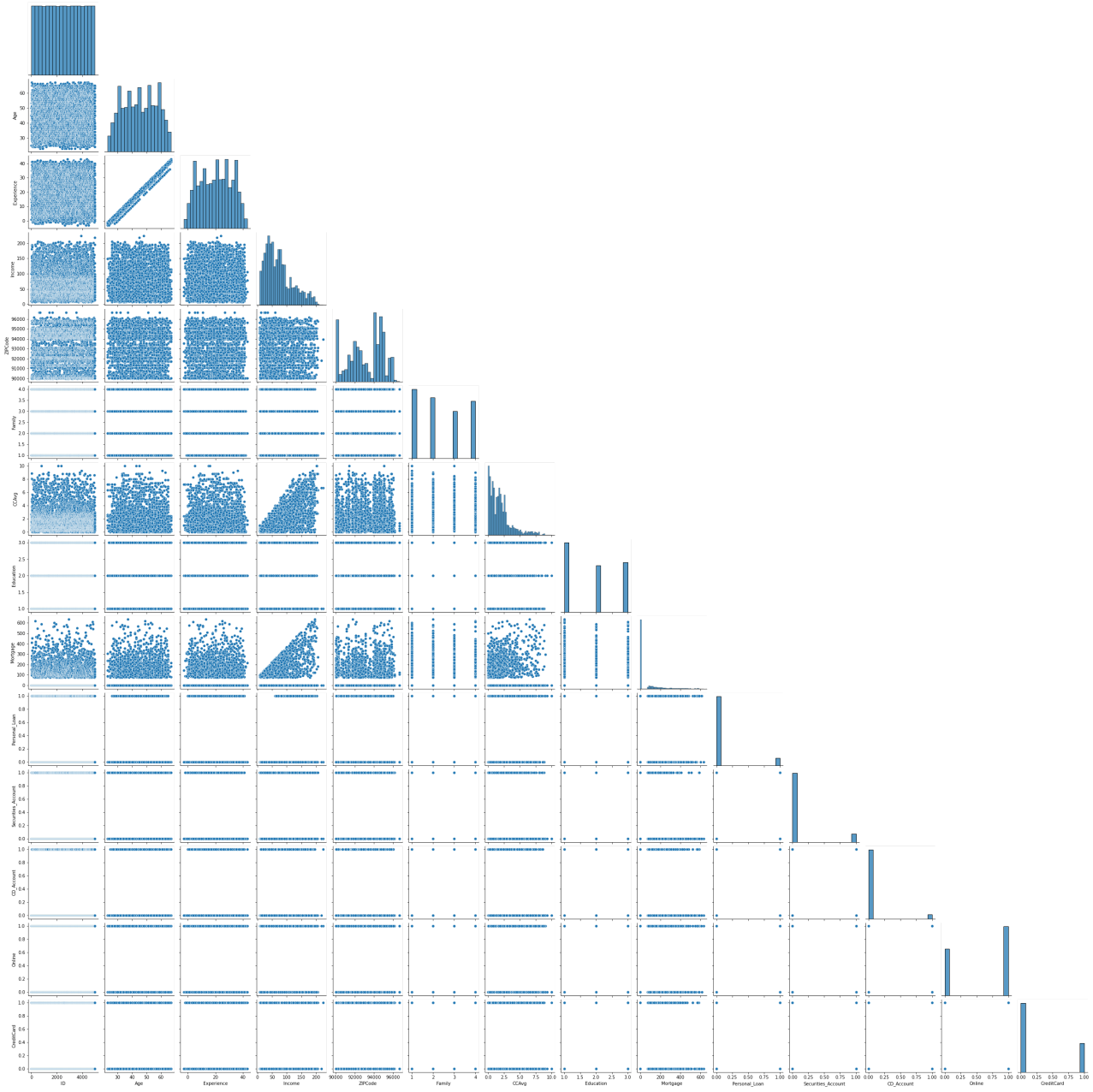
In [35]:
```
histogram_boxplot(data, "CreditCard")
```





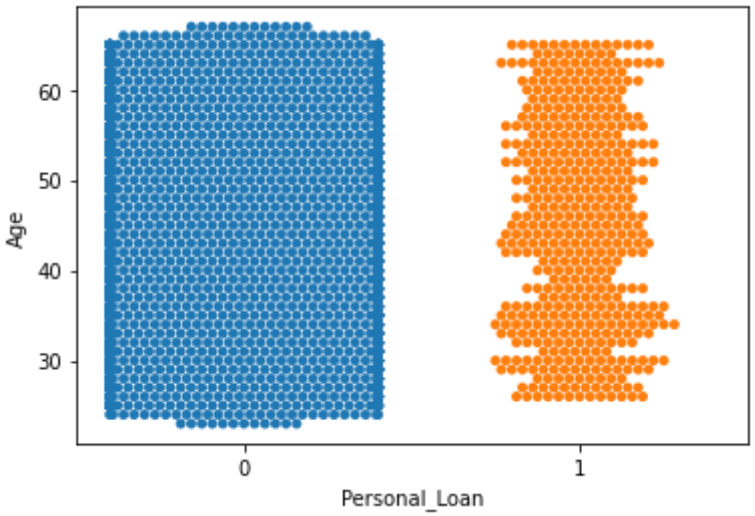
In [36]:
```
sns.pairplot(df, corner=True)
```

Out[36]: <seaborn.axisgrid.PairGrid at 0x1d5d3029af0>



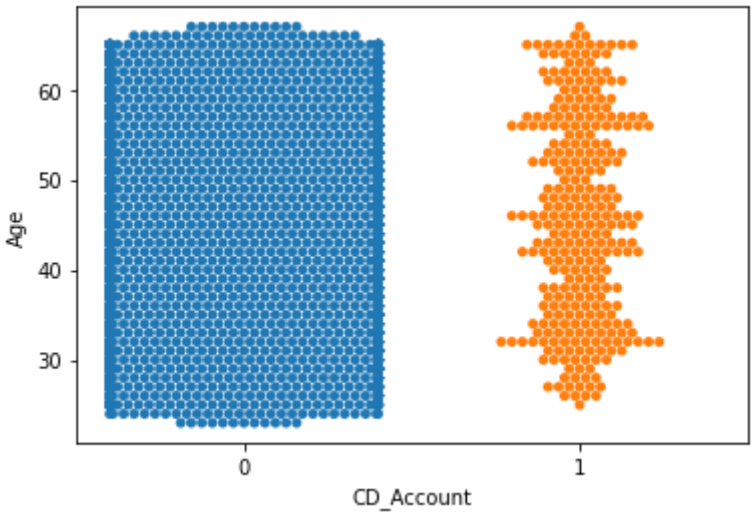
In [37]:
```
sns.swarmplot(df["Personal_Loan"], df["Age"])
```

Out[37]: <AxesSubplot:xlabel='Personal_Loan', ylabel='Age'>



In [38]:
```
sns.swarmplot(df["CD_Account"], df["Age"])
```

Out[38]: <AxesSubplot:xlabel='CD_Account', ylabel='Age'>

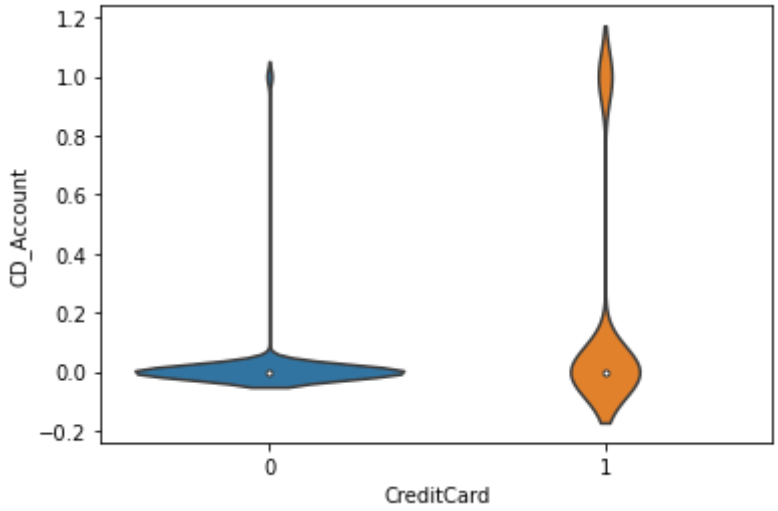


In [39]:
```
sns.swarmplot(df["CreditCard"], df["Income"])
```

Out[39]: <AxesSubplot:xlabel='CreditCard', ylabel='Income'>

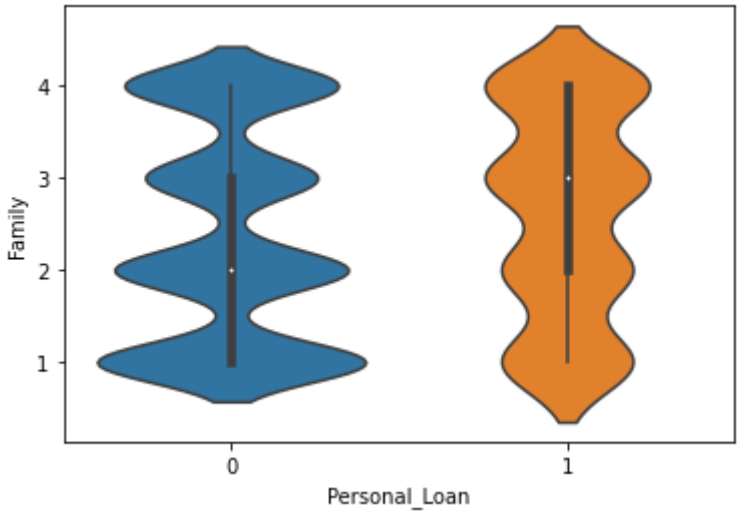
In [40]:
```
sns.violinplot(df["CreditCard"], df["CD_Account"])
```

Out[40]: <AxesSubplot:xlabel='CreditCard', ylabel='CD_Account'>



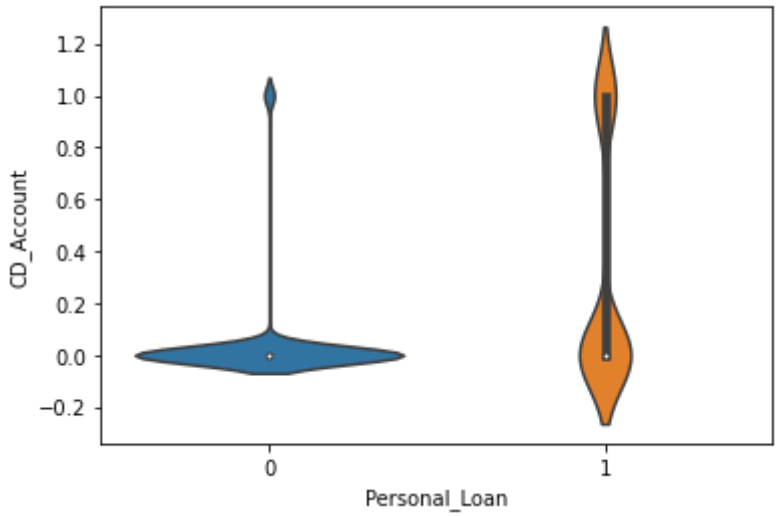
In [41]:
```
sns.violinplot(df["Personal_Loan"], df["Family"])
```

Out[41]: <AxesSubplot:xlabel='Personal_Loan', ylabel='Family'>



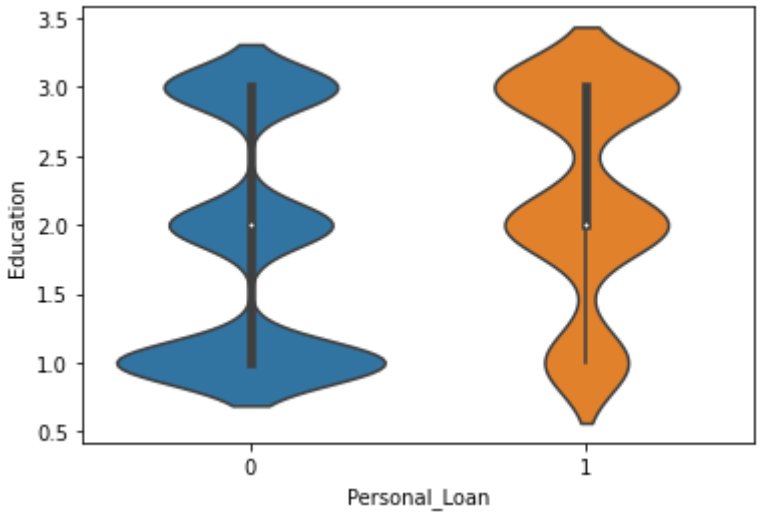
In [42]:
```
sns.violinplot(df["Personal_Loan"], df["CD_Account"])
```

Out[42]: <AxesSubplot:xlabel='Personal_Loan', ylabel='CD_Account'>



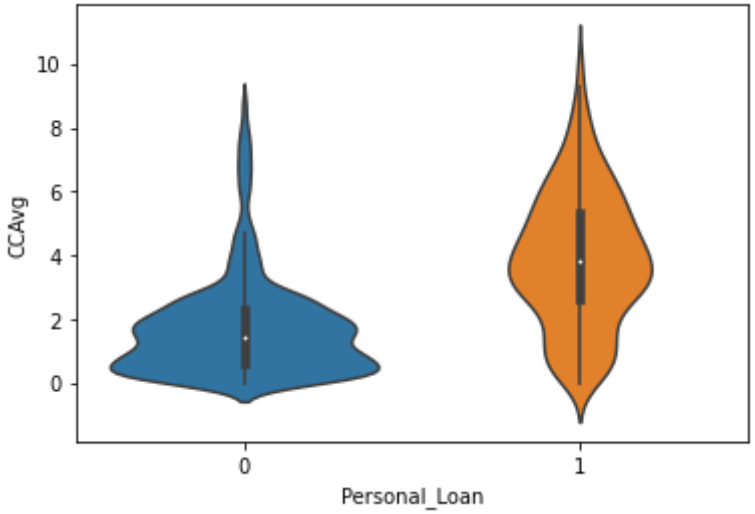
In [43]:
```
sns.violinplot(df["Personal_Loan"], df["Education"])
```

Out[43]:
```
<AxesSubplot:xlabel='Personal_Loan', ylabel='Education'>
```



In [44]:
```
sns.violinplot(df["Personal_Loan"], df["CCAvg"])
```

Out[44]:
```
<AxesSubplot:xlabel='Personal_Loan', ylabel='CCAvg'>
```

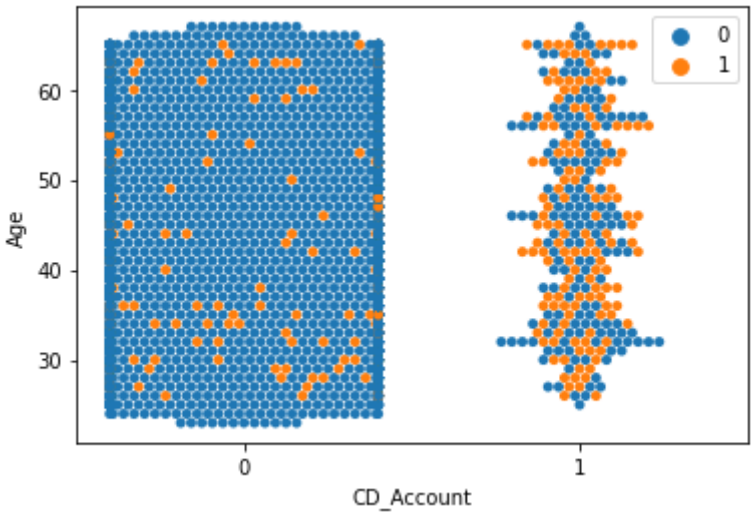


In [45]:
```
sns.swarmplot(df["CreditCard"], df["Income"], hue=df["Personal_Loan"])
plt.legend(bbox_to_anchor=(1, 1))
```

Out[45]: <matplotlib.legend.Legend at 0x1d5d31a4c40>

In [46]:
```
sns.swarmplot(df["CD_Account"], df["Age"], hue=df["Personal_Loan"])
plt.legend(bbox_to_anchor=(1, 1))
```

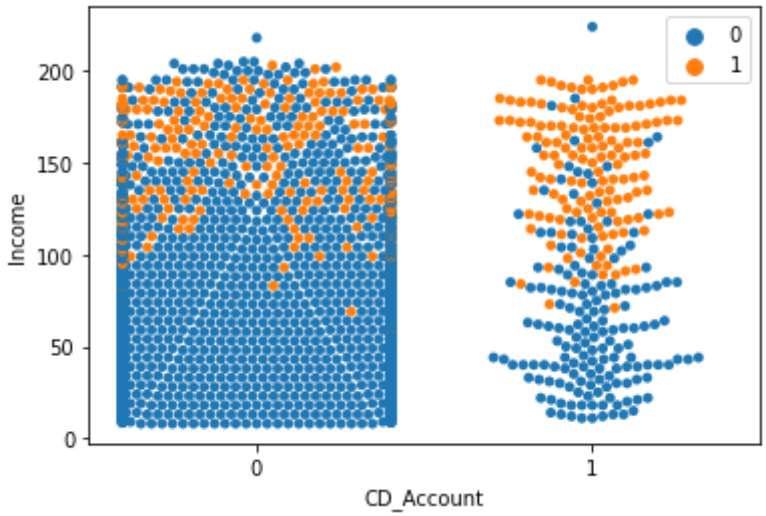
Out[46]: <matplotlib.legend.Legend at 0x1d5d3572940>



- The goal is to get those with a CD Account (1) to also be a Personal Loan customer (orange).

In [47]:
```
sns.swarmplot(df["CD_Account"], df["Income"], hue=df["Personal_Loan"])
plt.legend(bbox_to_anchor=(1, 1))
```

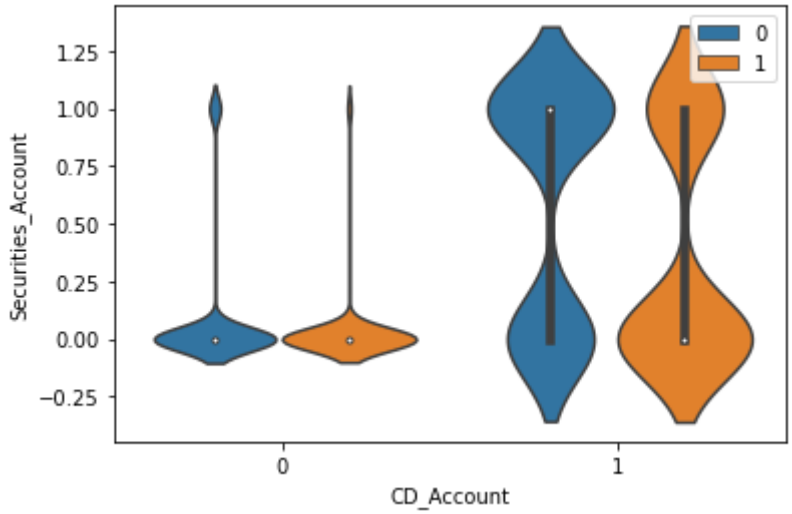
Out[47]: <matplotlib.legend.Legend at 0x1d5d2b42640>



- Here we see that those with higher incomes accept a loan offer. Makes sense as they are more likely to pay off the loan.

In [48]:
```
sns.violinplot(df["CD_Account"], df["Securities_Account"], hue=df["Personal_Loan
plt.legend(bbox_to_anchor=(1, 1))
```

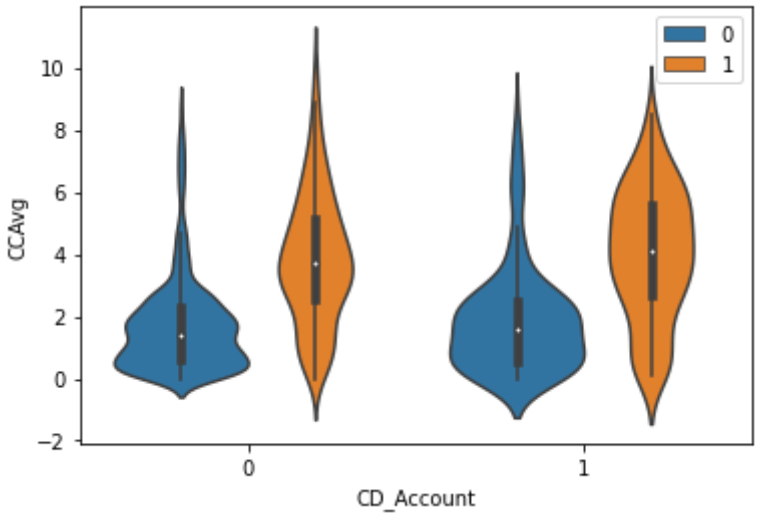
Out[48]:
```
<matplotlib.legend.Legend at 0x1d5d38be460>
```



- Having a Securities Account looks to have no real impact on if someone with accept a loan.
- However, having a Securities Account has some correlation to having a CD Account.

In [49]:
```
sns.violinplot(df["CD_Account"], df["CCAvg"], hue=df["Personal_Loan"])
plt.legend(bbox_to_anchor=(1, 1))
```

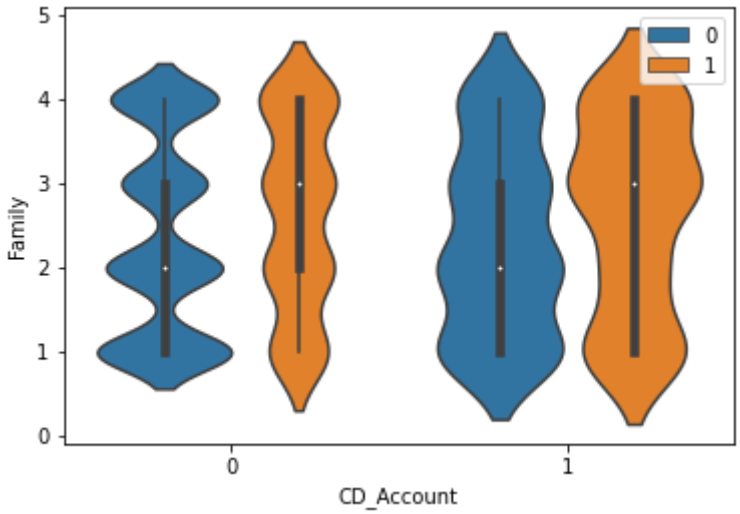
Out[49]:
```
<matplotlib.legend.Legend at 0x1d5d24c3d00>
```



- Those with a higher CCAvg have a higher chance of accepting a loan.
- Right now it seems the best way to have more loan customers is to target those with higher incomes and CCAvg.

In [50]:
```
sns.violinplot(df["CD_Account"], df["Family"], hue=df["Personal_Loan"])
plt.legend(bbox_to_anchor=(1, 1))
```

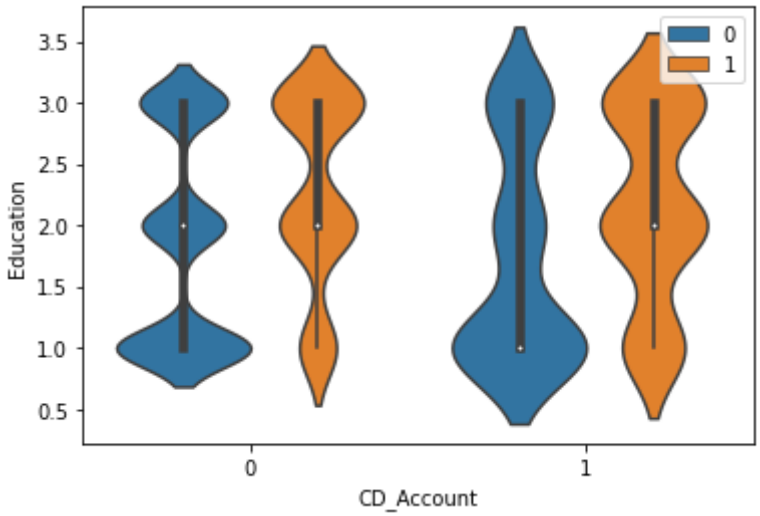
Out[50]:
```
<matplotlib.legend.Legend at 0x1d5dd1b4d60>
```



- Those with a CD_Account and with a family size of 3-4 have an increase chance of accepting a loan.

In [51]:
```
sns.violinplot(df["CD_Account"], df["Education"], hue=df["Personal_Loan"])
plt.legend(bbox_to_anchor=(1, 1))
```

Out[51]:
```
<matplotlib.legend.Legend at 0x1d5dd240280>
```



- Those with a CD_Account and with an education level of 2 and above have an increase chance of accepting a loan.

I would probably target people with higher incomes who do not have a credit card with AllLife. Possibly offer some type of credit card deal as those with higher credit card average have a greater chance of accepting a loan.

Those who open up a credit card account with AllLife could have a points system to incentivize credit card use. This in turn increases the chance of people building up their credit card average. As mentioned before, those with a credit card average of around 3k and above have a greater chance of accepting a loan.

People to target:

- Family size of 3-4
- Have a CD_Account
- Education level of 2 or greater
- Credit card average of 3k or more

Preprocessing

I noticed that there are a few negative numbers for years of experience, so I'll make them absolute values.

In [52]:
```
np.sum((df["Experience"] < 0).values.ravel())
```

Out[52]: 52

In [53]:
```
df["Experience"] = df["Experience"].abs()
```

In [54]:
```
np.sum((df["Experience"] < 0).values.ravel())
```

Out[54]: 0

In [55]:
```
my_tab = pd.crosstab(index=df["Experience"], columns="count")
my_tab
```

Out[55]:

col_0	count
Experience	
0	66
1	107
2	100
3	133
4	113
5	146
6	119
7	121
8	119
9	147
10	118
11	116
12	102
13	117
14	127
15	119
16	127
17	125
18	137
19	135
20	148
21	113
22	124
23	144
24	131
25	142
26	134
27	125
28	138

Experience	col_0 count
29	124
30	126
31	104
32	154
33	117
34	125
35	143
36	114
37	116
38	88
39	85
40	57
41	43
42	8
43	3

ZIP Codes

In [56]:
```
df1 = df.copy()
```

In [57]:
```
import uszipcode
from uszipcode import SearchEngine

search = SearchEngine()
```

In [58]:
```
county = []
for i in np.arange(0, len(df1["ZIPCode"])):
    zipcode = search.by_zipcode(df1["ZIPCode"][i])
    county.append(zipcode.county)
```

In [59]:
```
df1["County"] = county
df1.head()
```

Out[59]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	S
0	1	25	1	49	91107	4	1.6	1	0	0	
1	2	45	19	34	90089	3	1.5	1	0	0	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	S
2	3	39	15	11	94720	1	1.0	1	0	0	
3	4	35	9	100	94112	1	2.7	2	0	0	
4	5	35	8	45	91330	4	1.0	2	0	0	

In [60]:
```
df1["County"].unique()
```

Out[60]:
```
array(['Los Angeles County', 'Alameda County', 'San Francisco County',
       'San Diego County', 'Monterey County', 'Ventura County',
       'Santa Barbara County', 'Marin County', 'Santa Clara County',
       'Santa Cruz County', 'San Mateo County', 'Humboldt County',
       'Contra Costa County', 'Orange County', 'Sacramento County',
       'Yolo County', 'Placer County', 'San Bernardino County',
       'San Luis Obispo County', 'Riverside County', 'Kern County', None,
       'Fresno County', 'Sonoma County', 'El Dorado County',
       'San Benito County', 'Butte County', 'Solano County',
       'Mendocino County', 'San Joaquin County', 'Imperial County',
       'Siskiyou County', 'Merced County', 'Trinity County',
       'Stanislaus County', 'Shasta County', 'Tuolumne County',
       'Napa County', 'Lake County'], dtype=object)
```

In [61]:
```
my_tab = pd.crosstab(index=df1["County"], columns="count")
my_tab
```

Out[61]:

col_0	count
County	
Alameda County	500
Butte County	19
Contra Costa County	85
El Dorado County	17
Fresno County	26
Humboldt County	32
Imperial County	3
Kern County	54
Lake County	4
Los Angeles County	1095
Marin County	54
Mendocino County	8
Merced County	4
Monterey County	128

col_0	count
County	
Napa County	3
Orange County	339
Placer County	24
Riverside County	56
Sacramento County	184
San Benito County	14
San Bernardino County	101
San Diego County	568
San Francisco County	257
San Joaquin County	13
San Luis Obispo County	33
San Mateo County	204
Santa Barbara County	154
Santa Clara County	563
Santa Cruz County	68
Shasta County	18
Siskiyou County	7
Solano County	33
Sonoma County	28
Stanislaus County	15
Trinity County	4
Tuolumne County	7
Ventura County	114
Yolo County	130

In [62]:
```
df1["County"].isnull().sum()
```

Out[62]: 34

In [63]:
```
df1[df1["County"].isnull()]
```

Out[63]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_l
106	107	43	17	69	92717	4	2.90	1	0	
172	173	38	13	171	92717	2	7.80	1	0	
184	185	52	26	63	92717	2	1.50	2	0	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
321	322	44	20	101	92717	3	4.40	2	82	
366	367	50	24	35	92717	1	0.30	3	0	
384	385	51	25	21	93077	4	0.60	3	0	
468	469	34	10	21	92634	1	0.50	3	0	
476	477	60	34	53	92717	1	0.80	2	0	
630	631	32	7	35	96651	3	1.30	1	108	
672	673	51	27	23	96651	1	0.20	1	0	
695	696	29	4	115	92717	1	1.90	1	0	
721	722	49	24	39	92717	1	1.40	3	0	
780	781	32	7	42	92634	4	0.80	1	0	
1099	1100	30	6	52	92717	3	0.70	2	0	
1189	1190	42	17	115	92717	2	0.40	1	0	
1426	1427	37	11	60	96651	3	0.50	3	0	
1483	1484	58	32	63	92717	1	1.60	1	0	
1653	1654	26	1	24	96651	2	0.90	3	123	
1752	1753	33	8	155	92717	1	7.40	3	0	
1844	1845	65	40	21	92717	3	0.10	3	0	
2049	2050	43	18	94	92717	4	1.10	2	0	
2211	2212	39	14	31	92717	2	1.40	2	94	
2218	2219	38	13	9	92634	2	0.30	2	0	
2428	2429	39	12	108	92717	4	3.67	2	301	
2486	2487	61	36	130	92717	1	1.30	1	257	
2731	2732	29	5	28	96651	1	0.20	3	0	
2957	2958	61	36	53	92717	3	0.50	2	0	
3525	3526	59	34	13	96651	4	0.90	2	0	
3887	3888	24	2	118	92634	2	7.20	1	0	
4090	4091	42	18	49	92717	3	2.10	3	0	
4276	4277	50	24	155	92717	1	7.30	1	0	
4321	4322	27	0	34	92717	1	2.00	2	112	
4384	4385	45	20	61	92717	3	2.70	2	0	
4392	4393	52	27	81	92634	4	3.80	2	0	

In [64]:
```
df1.iloc[
    [
        106,
        172,
        184,
```

```
        321,
        366,
        476,
        695,
        721,
        1099,
        1189,
        1483,
        1752,
        1844,
        2049,
        2211,
        2428,
        2486,
        2957,
        4090,
        4276,
        4321,
        4384,
    ],
    [14],
] = "Orange County"
```

In [65]:
```
gk = df1.groupby("County")
gk.get_group("Orange County")
```

Out[65]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_l
35	36	48	24	81	92647	3	0.7	1	0	
40	41	57	32	84	92672	3	1.6	3	0	
94	95	65	39	121	92612	1	2.0	1	0	
106	107	43	17	69	92717	4	2.9	1	0	
113	114	58	34	92	92703	2	2.8	1	103	
...	...	...	...	...	...	...	...	...	...	...
4967	4968	41	16	69	92697	1	0.1	2	0	
4974	4975	59	33	64	92867	4	1.7	2	0	
4979	4980	50	26	92	90740	1	2.6	2	213	
4995	4996	29	3	40	92697	1	1.9	3	0	
4999	5000	28	4	83	92612	3	0.8	1	0	

361 rows × 15 columns

In [66]:
```
df1[df1["County"].isnull()]
```

Out[66]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
384	385	51	25	21	93077	4	0.6	3	0	
468	469	34	10	21	92634	1	0.5	3	0	
630	631	32	7	35	96651	3	1.3	1	108	
672	673	51	27	23	96651	1	0.2	1	0	
780	781	32	7	42	92634	4	0.8	1	0	
1426	1427	37	11	60	96651	3	0.5	3	0	
1653	1654	26	1	24	96651	2	0.9	3	123	
2218	2219	38	13	9	92634	2	0.3	2	0	
2731	2732	29	5	28	96651	1	0.2	3	0	
3525	3526	59	34	13	96651	4	0.9	2	0	
3887	3888	24	2	118	92634	2	7.2	1	0	
4392	4393	52	27	81	92634	4	3.8	2	0	

In [67]:
```
df1.iloc[[384], [14]] = "Ventura County"
```

In [68]:
```
gk = df1.groupby("County")
gk.get_group("Ventura County")
```

Out[68]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
9	10	34	9	180	93023	1	8.90	3	0	
23	24	44	18	43	91320	2	0.70	1	163	
59	60	31	5	188	91320	2	4.50	1	455	
65	66	59	35	131	91360	1	3.80	1	0	
76	77	58	32	12	91320	3	0.30	3	0	
135	136	58	33	45	93010	4	2.10	1	0	
168	169	50	26	13	91320	4	1.00	1	0	
185	186	39	14	115	91320	1	1.00	3	0	
188	189	64	40	169	91320	2	2.10	1	122	
233	234	62	37	58	91320	4	1.70	1	0	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
384	385	51	25	21	93077	4	0.60	3	0	
396	397	50	24	29	93023	4	0.10	1	0	
523	524	56	31	39	93023	4	0.90	1	0	
601	602	58	32	38	91320	1	1.40	1	0	
618	619	63	37	42	91320	2	0.70	3	115	
659	660	63	39	79	93009	4	1.70	2	0	
798	799	29	2	38	93063	1	2.00	2	0	
908	909	66	36	55	93023	4	1.67	3	0	
934	935	58	33	81	91320	2	0.00	3	0	
942	943	55	29	30	91320	4	0.70	2	0	
1170	1171	35	10	104	91320	3	0.60	2	0	
1205	1206	32	7	94	91361	2	3.10	1	0	
1240	1241	52	27	15	91320	4	0.80	1	101	
1377	1378	27	3	109	93023	2	2.50	1	0	
1458	1459	51	25	33	93033	1	1.40	3	0	
1461	1462	54	28	48	93022	1	0.20	1	0	
1477	1478	40	14	64	91320	4	0.20	3	0	
1486	1487	35	9	141	93022	2	4.50	2	0	
1540	1541	34	8	11	91320	4	0.30	1	0	
1555	1556	59	33	49	93009	4	1.70	2	104	
1558	1559	35	10	72	91320	3	2.30	1	285	
1593	1594	63	38	83	91320	3	1.80	2	0	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
1609	1610	66	41	105	93023	1	0.80	3	0	
1729	1730	50	20	25	91320	4	0.40	3	0	
1770	1771	62	37	9	91320	1	0.10	1	94	
1775	1776	46	22	73	91360	1	0.80	3	117	
1792	1793	46	20	118	93009	1	5.70	1	0	
1808	1809	55	31	50	93010	4	1.50	1	0	
1823	1824	33	8	125	91320	1	0.00	1	0	
1845	1846	43	18	65	93065	2	2.20	3	0	
1897	1898	54	29	98	93065	1	0.10	3	0	
1922	1923	39	15	25	93023	1	1.40	3	0	
2035	2036	36	10	29	93065	4	1.00	1	0	
2091	2092	31	4	41	91360	1	2.00	2	0	
2118	2119	31	5	125	91320	2	1.30	1	0	
2150	2151	62	38	54	91320	1	0.80	1	0	
2168	2169	55	29	64	93063	4	2.60	3	0	
2176	2177	41	14	51	91320	3	2.33	2	0	
2203	2204	50	25	130	91320	1	0.60	1	311	
2258	2259	59	33	93	91320	2	0.70	2	0	
2353	2354	61	36	12	93023	4	0.60	2	0	
2446	2447	25	1	70	93010	4	2.60	1	218	
2533	2534	54	29	111	93023	1	1.10	2	0	
2585	2586	51	26	70	91320	1	2.80	2	0	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
2588	2589	61	36	29	93065	1	1.30	1	0	
2660	2661	39	14	74	93063	1	0.10	2	144	
2678	2679	63	38	148	93023	2	4.30	3	0	
2712	2713	31	7	32	91320	1	1.70	1	0	
2764	2765	31	5	84	91320	1	2.90	3	105	
2769	2770	33	9	183	91320	2	8.80	3	582	
2900	2901	52	28	55	91320	2	3.20	3	151	
2911	2912	30	4	54	93033	4	1.80	3	235	
2935	2936	53	23	80	93023	1	3.00	3	0	
2943	2944	56	32	83	91320	4	1.60	2	0	
2966	2967	32	7	84	91320	3	0.60	2	0	
3036	3037	33	9	14	91320	4	0.70	2	105	
3066	3067	63	33	40	91320	4	1.67	3	0	
3069	3070	47	20	68	91320	1	2.67	2	0	
3071	3072	32	8	74	93023	4	0.10	2	257	
3119	3120	61	36	54	91320	3	0.90	3	179	
3217	3218	65	39	94	93022	4	4.10	1	120	
3363	3364	58	34	54	93003	4	1.30	2	0	
3374	3375	57	31	61	91360	1	2.20	3	0	
3388	3389	45	21	115	91320	2	3.30	1	85	
3423	3424	61	35	38	93009	2	1.00	3	0	
3429	3430	39	14	28	91320	2	1.40	2	108	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
3455	3456	43	19	28	93010	3	0.50	1	0	
3481	3482	52	26	34	93023	1	0.30	3	0	
3489	3490	36	12	154	91320	3	6.40	1	0	
3672	3673	38	13	65	91320	3	0.50	3	0	
3747	3748	26	0	83	91360	3	3.90	2	0	
3752	3753	55	30	82	93003	4	1.30	3	219	
3839	3840	31	5	42	93010	2	2.00	3	0	
3943	3944	61	36	188	91360	1	9.30	2	0	
3944	3945	56	26	62	91320	3	1.40	3	0	
3979	3980	38	14	90	93010	2	0.00	1	258	
4000	4001	62	37	93	93003	3	3.00	3	0	
4012	4013	30	6	124	91320	2	0.60	1	0	
4031	4032	42	18	29	91320	1	0.30	3	0	
4056	4057	51	25	113	91320	2	6.30	1	0	
4060	4061	31	6	174	93023	2	6.70	1	0	
4078	4079	36	12	58	91320	1	3.60	2	0	
4129	4130	29	3	10	91320	4	0.40	1	87	
4152	4153	44	18	91	91361	2	0.80	3	0	
4283	4284	58	32	62	91320	3	2.20	3	217	
4295	4296	65	41	91	91360	2	0.00	3	146	
4319	4320	63	38	85	91320	4	0.10	2	0	
4322	4323	38	14	44	91320	2	1.70	1	0	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_I
4337	4338	26	2	182	93010	2	3.20	2	0	
4379	4380	42	17	53	93023	4	1.90	3	0	
4405	4406	61	35	83	91320	2	1.70	3	245	
4419	4420	42	17	85	93065	1	3.70	3	272	
4504	4505	27	1	41	93023	4	1.80	3	147	
4576	4577	55	30	41	93003	2	0.60	3	0	
4594	4595	53	27	31	91320	3	0.90	3	78	
4613	4614	63	38	52	91361	4	1.70	1	218	
4675	4676	35	11	32	91360	1	1.33	1	137	
4684	4685	59	34	103	91360	1	2.60	1	0	
4699	4700	61	36	61	91320	2	2.80	1	153	
4704	4705	54	28	102	91360	3	1.70	2	0	
4713	4714	25	1	122	93022	2	0.20	1	0	
4761	4762	61	35	74	91320	2	0.70	2	0	
4966	4967	41	17	34	91361	1	0.70	1	143	
4968	4969	58	32	41	93022	4	2.50	1	0	
4997	4998	63	39	24	93023	2	0.30	3	0	

In [69]:
```
df1[df1["County"].isnull()]
```

Out[69]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_I
468	469	34	10	21	92634	1	0.5	3	0	
630	631	32	7	35	96651	3	1.3	1	108	
672	673	51	27	23	96651	1	0.2	1	0	
780	781	32	7	42	92634	4	0.8	1	0	
1426	1427	37	11	60	96651	3	0.5	3	0	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
1653	1654	26	1	24	96651	2	0.9	3	123	
2218	2219	38	13	9	92634	2	0.3	2	0	
2731	2732	29	5	28	96651	1	0.2	3	0	
3525	3526	59	34	13	96651	4	0.9	2	0	
3887	3888	24	2	118	92634	2	7.2	1	0	
4392	4393	52	27	81	92634	4	3.8	2	0	

In [70]:
```
df1.iloc[[468, 780, 2218, 3887, 4392], [14]] = "Orange County"
```

In [71]:
```
gk = df1.groupby("County")
gk.get_group("Orange County")
```

Out[71]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
35	36	48	24	81	92647	3	0.7	1	0	
40	41	57	32	84	92672	3	1.6	3	0	
94	95	65	39	121	92612	1	2.0	1	0	
106	107	43	17	69	92717	4	2.9	1	0	
113	114	58	34	92	92703	2	2.8	1	103	
...	...	...	...	...	...	...	...	...	...	...
4967	4968	41	16	69	92697	1	0.1	2	0	
4974	4975	59	33	64	92867	4	1.7	2	0	
4979	4980	50	26	92	90740	1	2.6	2	213	
4995	4996	29	3	40	92697	1	1.9	3	0	
4999	5000	28	4	83	92612	3	0.8	1	0	

366 rows × 15 columns

In [72]:
```
df1[df1["County"].isnull()]
```

Out[72]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
630	631	32	7	35	96651	3	1.3	1	108	

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
672	673	51	27	23	96651	1	0.2	1	0	
1426	1427	37	11	60	96651	3	0.5	3	0	
1653	1654	26	1	24	96651	2	0.9	3	123	
2731	2732	29	5	28	96651	1	0.2	3	0	
3525	3526	59	34	13	96651	4	0.9	2	0	

In [73]:
```
df1.iloc[[630, 672, 1426, 1653, 2731, 3525], [14]] = "San Francisco County"
```

In [74]:
```
gk = df1.groupby("County")
gk.get_group("San Francisco County")
```

Out[74]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_L
3	4	35	9	100	94112	1	2.7	2	0	
29	30	38	13	119	94104	1	3.3	2	0	
31	32	40	16	29	94117	1	2.0	2	0	
38	39	42	18	141	94114	3	5.0	3	0	
39	40	38	13	80	94115	4	0.7	3	285	
...	...	...	...	...	...	...	...	...	...	
4892	4893	43	19	35	94112	1	0.3	3	120	
4898	4899	52	26	19	94143	1	1.4	3	96	
4949	4950	29	5	64	94114	4	0.0	1	249	
4956	4957	39	13	59	94109	4	0.2	3	0	
4969	4970	45	19	60	94143	2	0.4	3	250	

263 rows × 15 columns

In [75]:
```
df1["County"].isnull().sum()
```

Out[75]: 0

In [76]:
```
df1["County"].replace("Alameda County", "Bay Area Counties", inplace=True)
```

In [77]:
```
df1["County"].replace("Butte County", "Superior Counties", inplace=True)
```

In [78]:
```
df1["County"].replace("Contra Costa County", "Bay Area Counties", inplace=True)
```

In [79]:
```
df1["County"].replace("El Dorado County", "Central Counties", inplace=True)
```

In [80]:
```
df1["County"].replace("Fresno County", "Central Counties", inplace=True)
```

In [81]:
```
df1["County"].replace("Humboldt County", "Superior Counties", inplace=True)
```

In [82]:
```
df1["County"].replace("Imperial County", "Southern Counties", inplace=True)
```

In [83]:
```
df1["County"].replace("Kern County", "Southern Counties", inplace=True)
```

In [84]:
```
df1["County"].replace("Lake County", "Superior Counties", inplace=True)
```

In [85]:
```
df1["County"].replace("Los Angeles County", "Los Angeles Region", inplace=True)
```

In [86]:
```
df1["County"].replace("Marin County", "Bay Area Counties", inplace=True)
```

In [87]:
```
df1["County"].replace("Mendocino County", "Superior Counties", inplace=True)
```

In [88]:
```
df1["County"].replace("Merced County", "Central Counties", inplace=True)
```

In [89]:
```
df1["County"].replace("Monterey County", "Bay Area Counties", inplace=True)
```

In [90]:
```
df1["County"].replace("Napa County", "Bay Area Counties", inplace=True)
```

In [91]:
```
df1["County"].replace("Orange County", "Southern Counties", inplace=True)
```

In [92]:
```
df1["County"].replace("Placer County", "Central Counties", inplace=True)
```

In [93]:
```
df1["County"].replace("Riverside County", "Southern Counties", inplace=True)
```

In [94]:
```
df1["County"].replace("Sacramento County", "Central Counties", inplace=True)
```

In [95]:
```
df1["County"].replace("San Benito County", "Bay Area Counties", inplace=True)
```

In [96]:
```
df1["County"].replace("San Bernardino County", "Southern Counties", inplace=True
```

In [97]:
```
df1["County"].replace("San Diego County", "Southern Counties", inplace=True)
```

In [98]:
```
df1["County"].replace("San Francisco County", "Bay Area Counties", inplace=True)
```

In [99]:
```
df1["County"].replace("San Joaquin County", "Central Counties", inplace=True)
```

In [100…
```
df1["County"].replace("San Luis Obispo County", "Southern Counties", inplace=Tru
```

In [101…
```
df1["County"].replace("San Mateo County", "Bay Area Counties", inplace=True)
```

In [102…
```
df1["County"].replace("Santa Barbara County", "Southern Counties", inplace=True)
```

In [103…
```
df1["County"].replace("Santa Clara County", "Bay Area Counties", inplace=True)
```

In [104…
```
df1["County"].replace("Santa Cruz County", "Bay Area Counties", inplace=True)
```

In [105…
```
df1["County"].replace("Shasta County", "Superior Counties", inplace=True)
```

In [106…
```
df1["County"].replace("Siskiyou County", "Superior Counties", inplace=True)
```

In [107…
```
df1["County"].replace("Solano County", "Bay Area Counties", inplace=True)
```

In [108…
```
df1["County"].replace("Sonoma County", "Bay Area Counties", inplace=True)
```

In [109…
```
df1["County"].replace("Stanislaus County", "Central Counties", inplace=True)
```

In [110…
```
df1["County"].replace("Trinity County", "Superior Counties", inplace=True)
```

In [111…
```
df1["County"].replace("Tuolumne County", "Central Counties", inplace=True)
```

In [112…
```
df1["County"].replace("Ventura County", "Southern Counties", inplace=True)
```

In [113…
```
df1["County"].replace("Yolo County", "Central Counties", inplace=True)
```

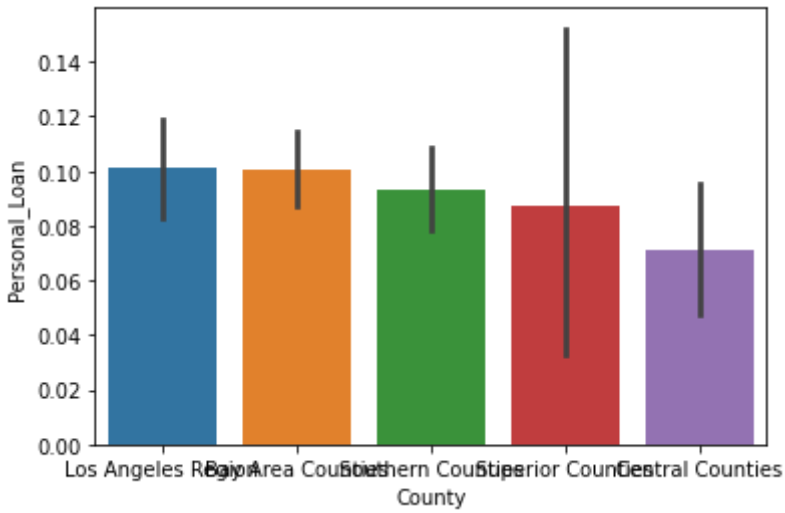
In [114…
```
my_tab = pd.crosstab(index=df1["County"], columns="count")
my_tab
```

Out[114…

col_0	count
County	
Bay Area Counties	1943
Central Counties	420
Los Angeles Region	1095
Southern Counties	1450
Superior Counties	92

In [115…
```
sns.barplot(df1["County"], df1["Personal_Loan"])
```

Out[115…
```
<AxesSubplot:xlabel='County', ylabel='Personal_Loan'>
```



In [116…
```
df1.drop(["ID"], axis=1, inplace=True)
```

- Dropped 'ID' as it is useless.

In [117…
```
df1.info()
```

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

In [118…
```
df2 = df1.copy()
```

In [119…
```
df2.head()
```

Out[119…

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

In [120…
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Age                 5000 non-null   int64  
 1   Experience          5000 non-null   int64  
 2   Income              5000 non-null   int64  
```

```
 3   ZIPCode             5000 non-null   int64
 4   Family              5000 non-null   int64
 5   CCAvg               5000 non-null   float64
 6   Education           5000 non-null   int64
 7   Mortgage            5000 non-null   int64
 8   Personal_Loan       5000 non-null   int64
 9   Securities_Account  5000 non-null   int64
 10  CD_Account          5000 non-null   int64
 11  Online              5000 non-null   int64
 12  CreditCard          5000 non-null   int64
 13  County              5000 non-null   object
dtypes: float64(1), int64(12), object(1)
memory usage: 547.0+ KB
```

Creating dummies for County.

In [121…
```
cty_dummies = pd.get_dummies(df2["County"], drop_first=True)
```

In [122…
```
df2 = pd.concat([df2, cty_dummies], axis=1)
df2.drop(["County"], inplace=True, axis=1)
```

In [123…
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 5000 non-null   int64
 1   Experience          5000 non-null   int64
 2   Income              5000 non-null   int64
 3   ZIPCode             5000 non-null   int64
 4   Family              5000 non-null   int64
 5   CCAvg               5000 non-null   float64
 6   Education           5000 non-null   int64
 7   Mortgage            5000 non-null   int64
 8   Personal_Loan       5000 non-null   int64
 9   Securities_Account  5000 non-null   int64
 10  CD_Account          5000 non-null   int64
 11  Online              5000 non-null   int64
 12  CreditCard          5000 non-null   int64
 13  Central Counties    5000 non-null   uint8
 14  Los Angeles Region  5000 non-null   uint8
 15  Southern Counties   5000 non-null   uint8
 16  Superior Counties   5000 non-null   uint8
dtypes: float64(1), int64(12), uint8(4)
memory usage: 527.5 KB
```

Spliting the data

In [124…
```
X = df2.drop("Personal_Loan", axis=1)
y = df2["Personal_Loan"]
```

In [125…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
print(X_train.shape, X_test.shape)
```

```
(3500, 16) (1500, 16)
```

In [126…
```
print("Number of rows in train data =", X_train.shape[0])
print("Number of rows in test data =", X_test.shape[0])
```

```
Number of rows in train data = 3500
Number of rows in test data = 1500
```

In [127…
```
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))
```

```
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
```

Model Evaluation Functions

In [128…
```
def get_recall_score(model, predictors, target):
    """
    model: classifier
    predictors: independent variables
    target: dependent variable

    """
    prediction = model.predict(predictors)
    return recall_score(target, prediction)
```

In [129…
```
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().su
            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")
```

In [130…

```
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics

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

    # predicting using the independent variables
    pred_prob = model.predict_proba(predictors)[:, 1]
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)
    recall = recall_score(target, pred)
    precision = precision_score(target, pred)
    f1 = f1_score(target, pred)

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

    return df_perf
```

In [131…

```
def confusion_matrix_sklearn_with_threshold(model, predictors, target, threshold
    """
    To plot the confusion_matrix, based on the threshold specified, with percent

    model: classifier
    predictors: independent variables
    target: dependent variable
    threshold: threshold for classifying the observation as class 1
    """
    pred_prob = model.predict_proba(predictors)[:, 1]
    pred_thres = pred_prob > threshold
    y_pred = np.round(pred_thres)

    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().su
            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")
```

In [132…
```
def model_performance_classification_sklearn_with_threshold(
    model, predictors, target, threshold=0.5
):
    """
    Function to compute different metrics, based on the threshold specified, to

    model: classifier
    predictors: independent variables
    target: dependent variable
    threshold: threshold for classifying the observation as class 1
    """

    # predicting using the independent variables
    pred_prob = model.predict_proba(predictors)[:, 1]
    pred_thres = pred_prob > threshold
    pred = np.round(pred_thres)

    acc = accuracy_score(target, pred)
    recall = recall_score(target, pred)
    precision = precision_score(target, pred)
    f1 = f1_score(target, pred)

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

    return df_perf
```

Building Logistic Regression Model

In [133…
```
lg = LogisticRegression(solver="newton-cg", random_state=1)
model = lg.fit(X_train, y_train)
```

Checking Coefficients

In [134…
```
coef_df = pd.DataFrame(
    np.append(lg.coef_, lg.intercept_),
    index=X_train.columns.tolist() + ["Intercept"],
    columns=["Coefficients"],
)
coef_df
```

Out[134…

	Coefficients
Age	-0.025348

	Coefficients
Experience	0.030482
Income	0.052317
ZIPCode	-0.000133
Family	0.711639
CCAvg	0.167175
Education	1.660955
Mortgage	0.000763
Securities_Account	-0.834054
CD_Account	3.250066
Online	-0.522184
CreditCard	-0.928874
Central Counties	-0.060427
Los Angeles Region	-0.566271
Southern Counties	-0.222722
Superior Counties	-0.411202
Intercept	-0.125868

- Coefficient of Family, CCAvg, Education, and CD_Account are positive, so an increase in these will lead to an increase in chance of someone having a personal loan.
- Coefficient of Securities_Account, Online, CreditCard, Los Angeles Region, and Superior Counties are negative, so an increase in these will lead to a decrease in chance of someone having a personal loan.

Converting coefficients to odds.

- odds = (exp(b) - 1) * 100

In [135…
```
odds = np.exp(lg.coef_[0])

perc_change_odds = (np.exp(lg.coef_[0]) - 1) * 100

pd.set_option("display.max_columns", None)

pd.DataFrame({"Odds": odds, "Change_odd%": perc_change_odds}, index=X_train.colu
```

Out[135…

	Odds	Change_odd%
Age	0.974971	-2.502940
Experience	1.030951	3.095120
Income	1.053710	5.370997
ZIPCode	0.999867	-0.013305

	Odds	Change_odd%
Family	2.037328	103.732800
CCAvg	1.181961	18.196121
Education	5.264337	426.433747
Mortgage	1.000763	0.076308
Securities_Account	0.434285	-56.571473
CD_Account	25.792043	2479.204251
Online	0.593224	-40.677644
CreditCard	0.394998	-60.500168
Central Counties	0.941363	-5.863741
Los Angeles Region	0.567638	-43.236200
Southern Counties	0.800338	-19.966231
Superior Counties	0.662853	-33.714701

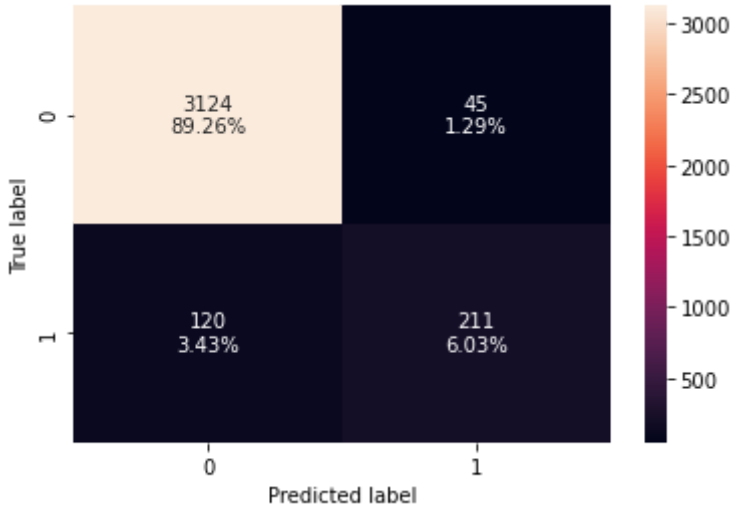
- A 1 unit change in CD_Account will increase the odds of someone having a personal loan by 2479%.
- A 1 unit change in Education will increase the odds of someone having a personal loan by 426%.
- A 1 unit change in Family will increase the odds of someone having a personal loan by 103.7%.
- A 1 unit change in CCAvg will increase the odds of someone having a personal loan by 18%.

- A 1 unit change in CreditCard will decrease the odds of someone having a personal loan by 60.5%.
- A 1 unit change in Securities_Account will decrease the odds of someone having a personal loan by 56.57%.
- A 1 unit change in Los Angeles Region will decrease the odds of someone having a personal loan by 43%.
- A 1 unit change in Online will decrease the odds of someone having a personal loan by 40.6%.

Checking model performance

In [136…

```
confusion_matrix_sklearn_with_threshold(lg, X_train, y_train)
```



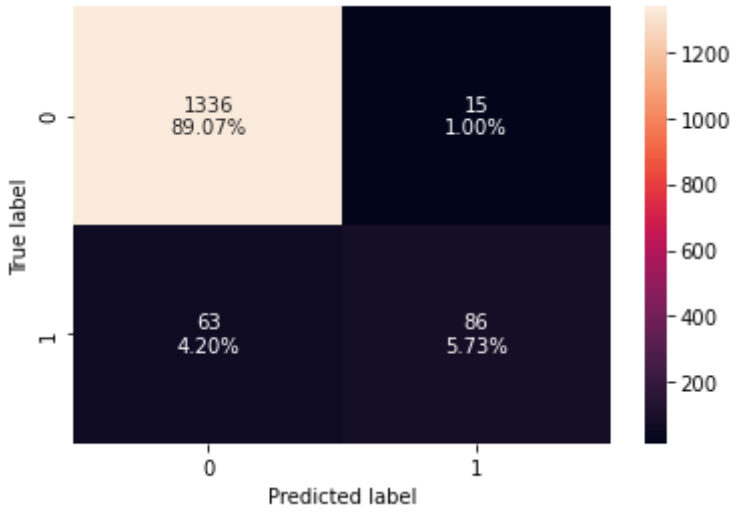
In [137…
```
log_reg_model_train_perf = model_performance_classification_sklearn_with_thresho
    lg, X_train, y_train
)

print("Training performance:")
log_reg_model_train_perf
```

```
Training performance:
```

Out[137…

	Accuracy	Recall	Precision	F1
0	0.952857	0.637462	0.824219	0.71891

In [138…
```
confusion_matrix_sklearn_with_threshold(lg, X_test, y_test)
```



In [139…
```
log_reg_model_test_perf = model_performance_classification_sklearn_with_threshol
    lg, X_test, y_test
)

print("Test set performance:")
log_reg_model_test_perf
```

```
Test set performance:
```

Out[139…

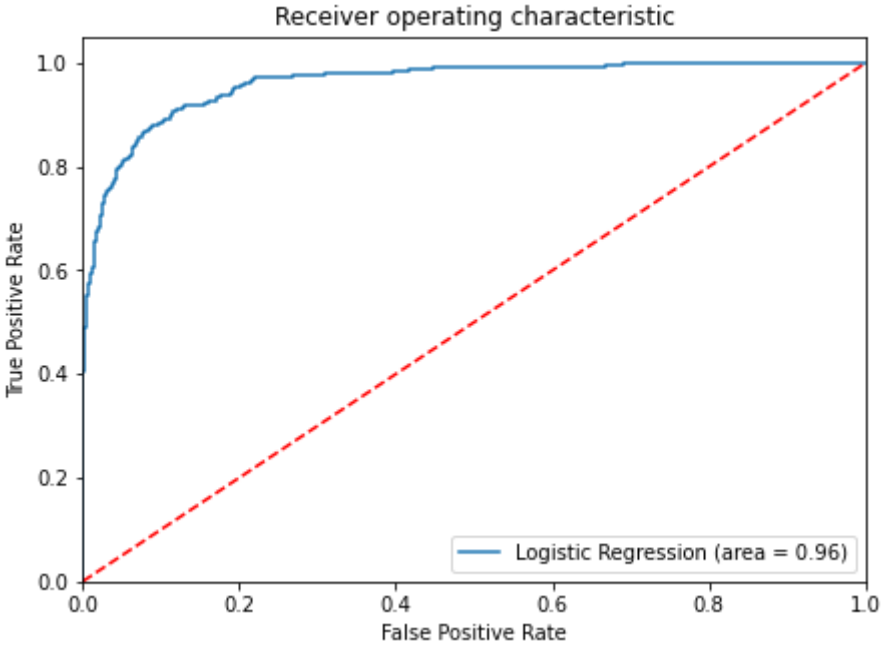
	Accuracy	Recall	Precision	F1
0	0.948	0.577181	0.851485	0.688

ROC-AUC Chart

- Train Data

In [140…

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

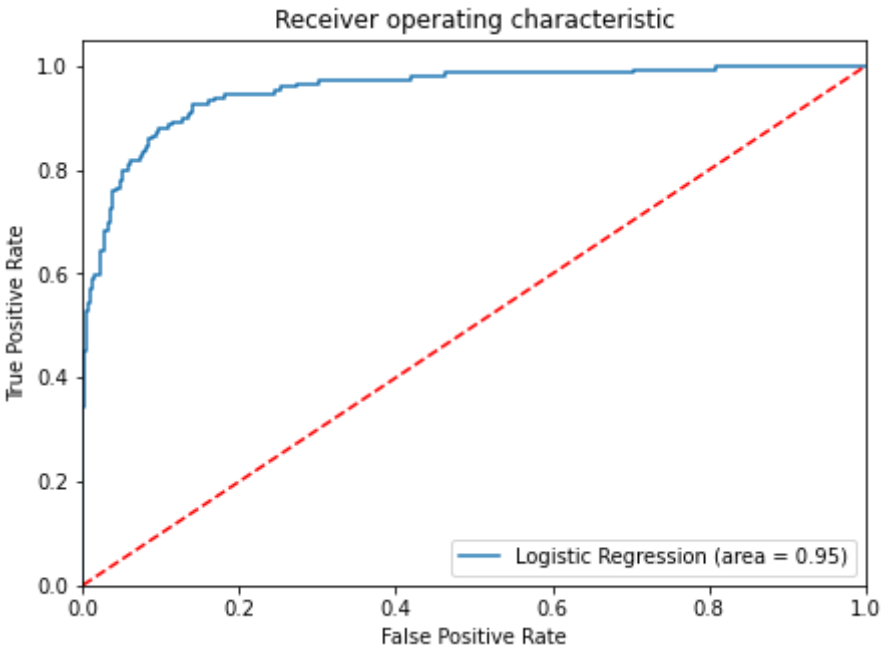


- Test Data

In [141…

```
logit_roc_auc_test = roc_auc_score(y_test, lg.predict_proba(X_test)[:, 1])
fpr, tpr, thresholds = roc_curve(y_test, lg.predict_proba(X_test)[:, 1])
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, label="Logistic Regression (area = %0.2f)" % logit_roc_auc_te
plt.plot([0, 1], [0, 1], "r--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
```

```
plt.legend(loc="lower right")
plt.show()
```



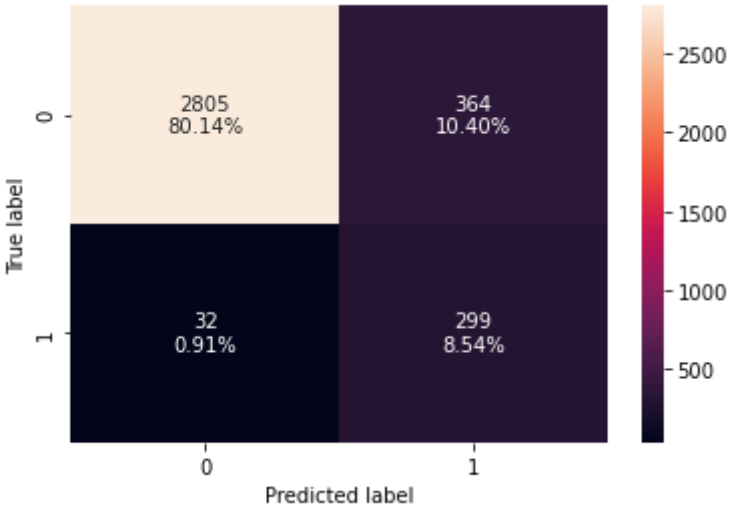
Changing threshold to potentially get a better score

- Finding the optimal threshold.

In [142…
```
fpr, tpr, thresholds = roc_curve(y_train, lg.predict_proba(X_train)[:, 1])

optimal_idx = np.argmax(tpr - fpr)
optimal_threshold_auc_roc = thresholds[optimal_idx]
print(optimal_threshold_auc_roc)
```

```
0.09016579592253927
```

In [143…
```
confusion_matrix_sklearn_with_threshold(
    lg, X_train, y_train, threshold=optimal_threshold_auc_roc
)
```



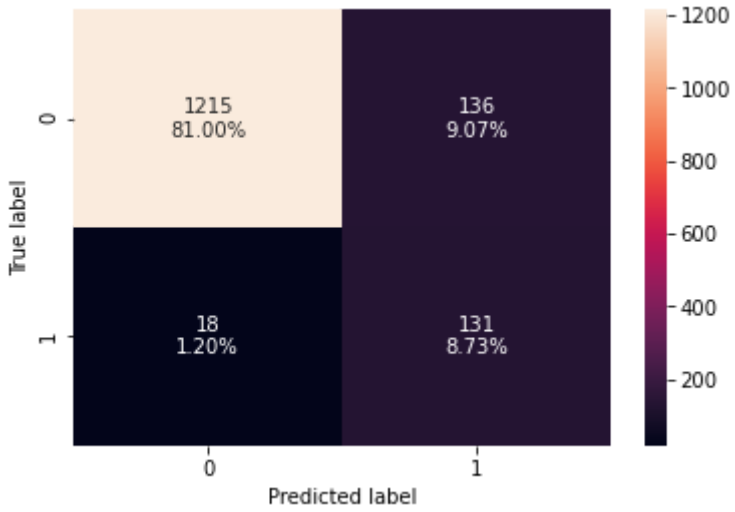
In [144…
```
log_reg_model_train_perf_threshold_auc_roc = (
    model_performance_classification_sklearn_with_threshold(
        lg, X_train, y_train, threshold=optimal_threshold_auc_roc
    )
)
print("Training performance:")
log_reg_model_train_perf_threshold_auc_roc
```

```
Training performance:
```

Out[144…

	Accuracy	Recall	Precision	F1
0	0.886857	0.903323	0.45098	0.60161

In [145…
```
confusion_matrix_sklearn_with_threshold(
    lg, X_test, y_test, threshold=optimal_threshold_auc_roc
)
```



In [146…
```
log_reg_model_test_perf_threshold_auc_roc = (
    model_performance_classification_sklearn_with_threshold(
        lg, X_test, y_test, threshold=optimal_threshold_auc_roc
    )
)
print("Test set performance:")
log_reg_model_test_perf_threshold_auc_roc
```

```
Test set performance:
```

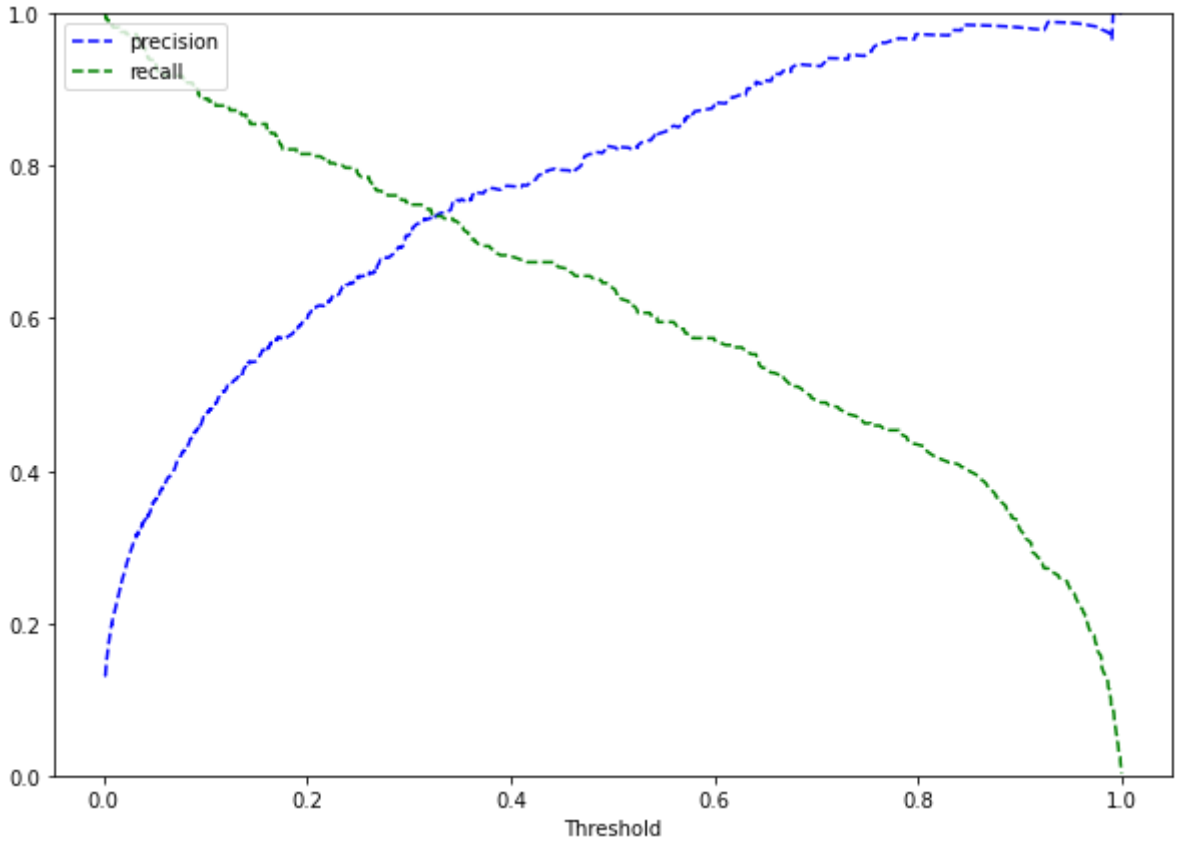
Out[146…

	Accuracy	Recall	Precision	F1
0	0.897333	0.879195	0.490637	0.629808

- The precision went down. I want a good balance between Recal, Precision, and F1.

In [147…
```
y_scores = lg.predict_proba(X_train)[:, 1]
prec, rec, tre = precision_recall_curve(
    y_train,
    y_scores,
)
```

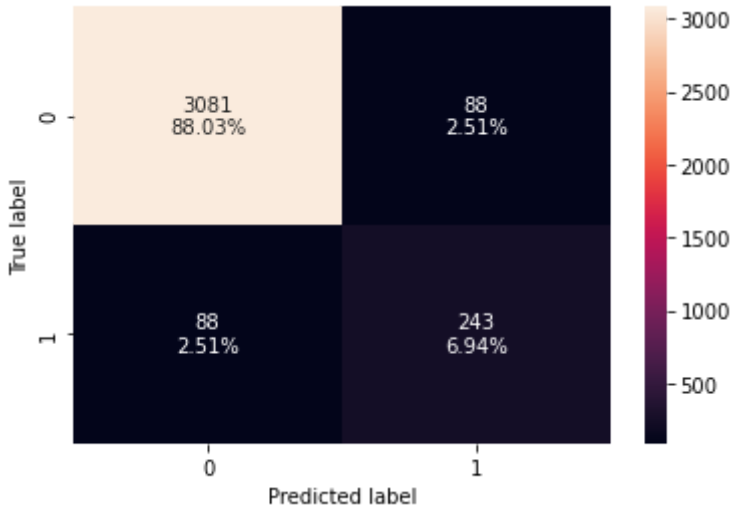
```
def plot_prec_recall_vs_tresh(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper left")
    plt.ylim([0, 1])


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



- The balance between prec and recall is ~ 0.325

In [148…
```
optimal_threshold_curve = 0.325
```

In [149…
```
confusion_matrix_sklearn_with_threshold(
    lg, X_train, y_train, threshold=optimal_threshold_curve
)
```



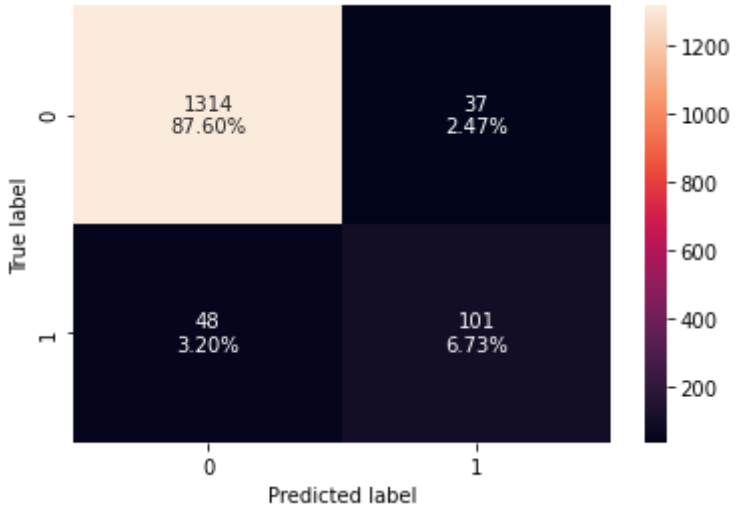
In [150…
```
log_reg_model_train_perf_threshold_curve = (
    model_performance_classification_sklearn_with_threshold(
        lg, X_train, y_train, threshold=optimal_threshold_curve
    )
)
print("Training performance:")
log_reg_model_train_perf_threshold_curve
```

```
Training performance:
```

Out[150…

	Accuracy	Recall	Precision	F1
0	0.949714	0.734139	0.734139	0.734139

In [151…
```
confusion_matrix_sklearn_with_threshold(
    lg, X_test, y_test, threshold=optimal_threshold_curve
)
```



In [152…
```
log_reg_model_test_perf_threshold_curve = (
    model_performance_classification_sklearn_with_threshold(
        lg, X_test, y_test, threshold=optimal_threshold_curve
    )
)
```

```
print("Test set performance:")
log_reg_model_test_perf_threshold_curve
```

```
Test set performance:
```

Out[152…

	Accuracy	Recall	Precision	F1
0	0.943333	0.677852	0.731884	0.703833

- I want to use this model with a threshold of 0.325

A good F1 score is what I want because that means there is a good balance between recall and precision (or FN and FP). If I have a precision score too high, I won't be able to identify customer in the past accurately. If I have a recall that is too high, it will give me a false sense of revenue from loans.

Decision Tree

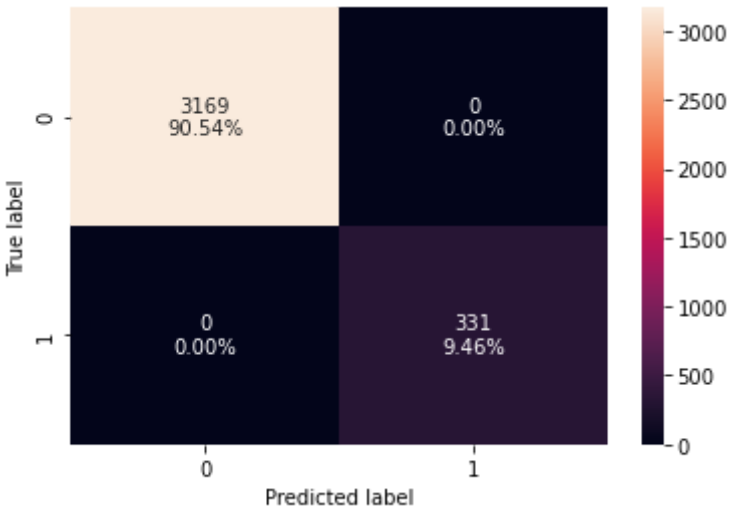
- The balance of data is ~0:0.90, ~1:0.10, so I will set the class weight accordingly.

In [153…

```
dt = DecisionTreeClassifier(
    criterion="gini", class_weight={0: 0.10, 1: 0.90}, random_state=1
)
model2 = dt.fit(X_train, y_train)
```

In [154…

```
confusion_matrix_sklearn(dt, X_train, y_train)
```



In [155…

```
decision_tree_perf_train = get_recall_score(dt, X_train, y_train)

print("Recall Score:", decision_tree_perf_train)
```

```
Recall Score: 1.0
```

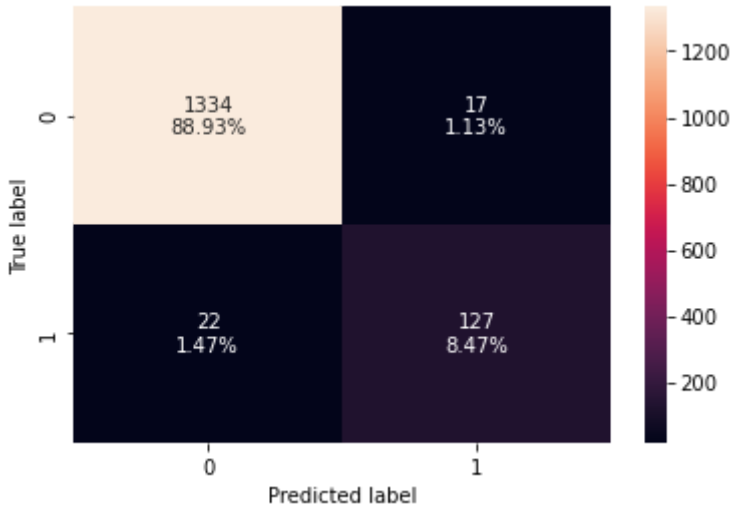
In [156…
```
dec_tree_model_train_perf = model_performance_classification_sklearn(
    dt, X_train, y_train
)

print("Training performance:")
dec_tree_model_train_perf
```

```
Training performance:
```

Out[156…

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

In [157…
```
confusion_matrix_sklearn(dt, X_test, y_test)
```



In [158…
```
decision_tree_perf_test = get_recall_score(dt, X_test, y_test)
print("Recall Score:", decision_tree_perf_test)
```

```
Recall Score: 0.8523489932885906
```

In [159…
```
dec_tree_model_test_perf = model_performance_classification_sklearn(dt, X_test,

print("Test set performance:")
dec_tree_model_test_perf
```

```
Test set performance:
```

Out[159…

	Accuracy	Recall	Precision	F1
0	0.974	0.852349	0.881944	0.866894

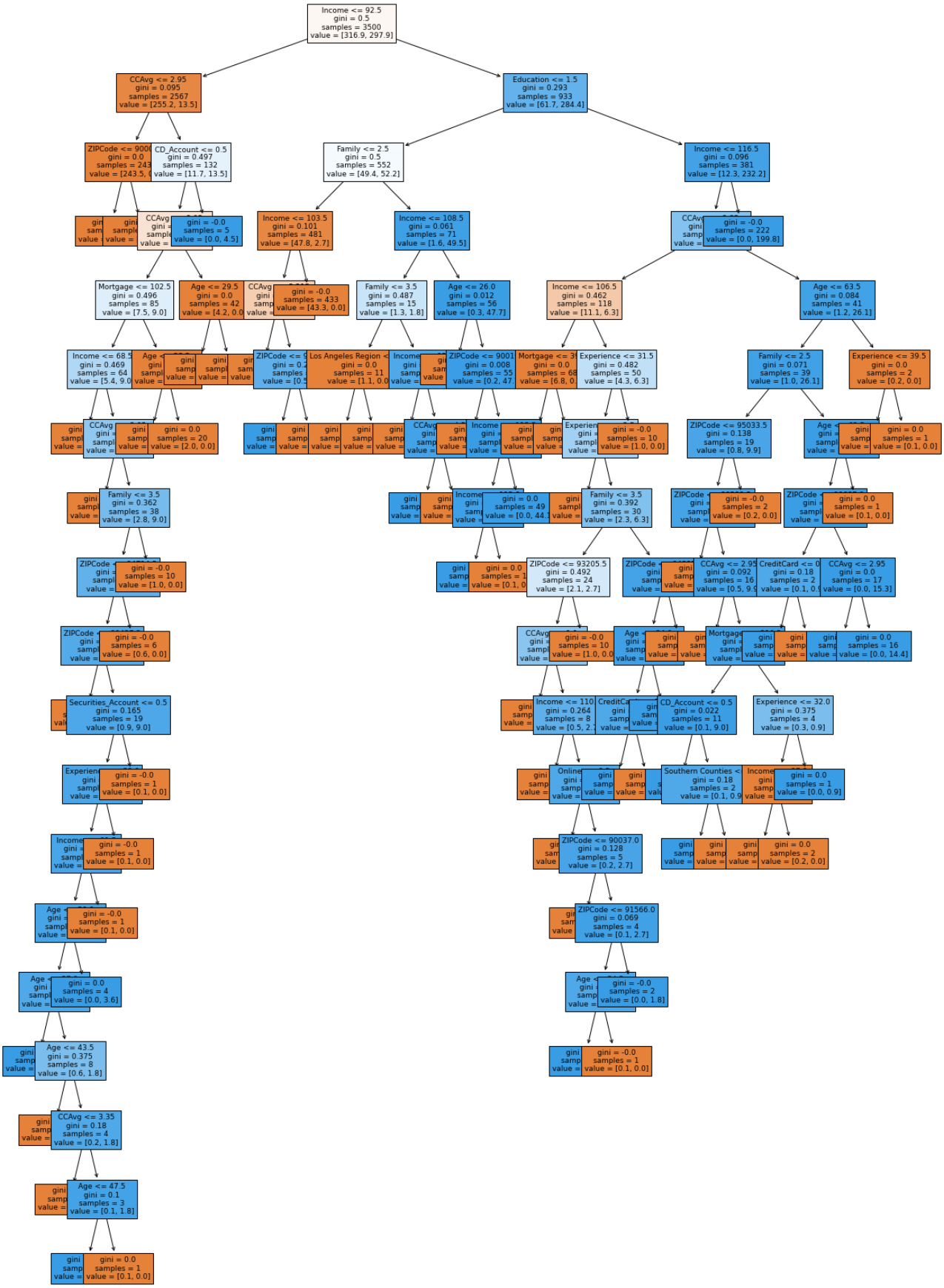
- There is quite the difference between the train and test scores, so this means the model is overfitting.

Tree Visualization

In [160…
```
feature_names = X_train.columns.to_list()
```

In [161…

```
plt.figure(figsize=(20, 30))
out = tree.plot_tree(
    model2,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)

for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



In [162…

```
print(tree.export_text(model2, feature_names=feature_names, show_weights=True))
```

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- ZIPCode <= 90006.00
|   |   |   |--- weights: [0.40, 0.00] class: 0
|   |   |--- ZIPCode >  90006.00
|   |   |   |--- weights: [243.10, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- CD_Account <= 0.50
|   |   |   |--- CCAvg <= 3.95
|   |   |   |   |--- Mortgage <= 102.50
|   |   |   |   |   |--- Income <= 68.50
|   |   |   |   |   |   |--- weights: [1.50, 0.00] class: 0
|   |   |   |   |   |--- Income >  68.50
|   |   |   |   |   |   |--- CCAvg <= 3.05
|   |   |   |   |   |   |   |--- weights: [1.10, 0.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  3.05
|   |   |   |   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |   |   |   |--- ZIPCode <= 94714.50
|   |   |   |   |   |   |   |   |   |--- ZIPCode <= 90437.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.30, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- ZIPCode >  90437.50
|   |   |   |   |   |   |   |   |   |   |--- Securities_Account <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 8
|   |   |   |   |   |   |   |   |   |   |--- Securities_Account >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- ZIPCode >  94714.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.60, 0.00] class: 0
|   |   |   |   |   |   |   |--- Family >  3.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- Mortgage >  102.50
|   |   |   |   |   |--- Age <= 28.50
|   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |--- Age >  28.50
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |--- CCAvg >  3.95
|   |   |   |   |--- Age <= 29.50
|   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |--- Age >  29.50
|   |   |   |   |   |--- weights: [4.10, 0.00] class: 0
|   |   |--- CD_Account >  0.50
|   |   |   |--- weights: [0.00, 4.50] class: 1
|--- Income >  92.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- Income <= 103.50
|   |   |   |   |--- CCAvg <= 3.21
|   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |--- CCAvg >  3.21
|   |   |   |   |   |--- ZIPCode <= 91485.50
|   |   |   |   |   |   |--- weights: [0.00, 2.70] class: 1
|   |   |   |   |   |--- ZIPCode >  91485.50
|   |   |   |   |   |   |--- weights: [0.50, 0.00] class: 0
|   |   |   |--- Income >  103.50
|   |   |   |   |--- weights: [43.30, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- Income <= 108.50
|   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |--- Los Angeles Region <= 0.50
|   |   |   |   |   |   |--- weights: [0.90, 0.00] class: 0
|   |   |   |   |   |--- Los Angeles Region >  0.50
|   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |--- Family >  3.50
|   |   |   |   |   |--- Income <= 93.50
|   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |--- Income >  93.50
```

```
|   |   |   |   |   |   |--- CCAvg <= 4.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.80] class: 1
|   |   |   |   |   |   |--- CCAvg >  4.50
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |--- Income >  108.50
|   |   |   |   |   |--- Age <= 26.00
|   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |--- Age >  26.00
|   |   |   |   |   |   |--- ZIPCode <= 90019.50
|   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |--- ZIPCode >  90019.50
|   |   |   |   |   |   |   |--- Income <= 113.50
|   |   |   |   |   |   |   |   |--- Income <= 112.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.60] class: 1
|   |   |   |   |   |   |   |   |--- Income >  112.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- Income >  113.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 44.10] class: 1
|   |   |--- Education >  1.50
|   |   |   |--- Income <= 116.50
|   |   |   |   |--- CCAvg <= 2.85
|   |   |   |   |   |--- Income <= 106.50
|   |   |   |   |   |   |--- Mortgage <= 39.50
|   |   |   |   |   |   |   |--- weights: [4.80, 0.00] class: 0
|   |   |   |   |   |   |--- Mortgage >  39.50
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |--- Income >  106.50
|   |   |   |   |   |   |--- Experience <= 31.50
|   |   |   |   |   |   |   |--- Experience <= 3.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Experience >  3.50
|   |   |   |   |   |   |   |   |--- Family <= 3.50
|   |   |   |   |   |   |   |   |   |--- ZIPCode <= 93205.50
|   |   |   |   |   |   |   |   |   |   |--- CCAvg <= 1.10
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.60, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- CCAvg >  1.10
|   |   |   |   |   |   |   |   |   |   |   |--- Income <= 110.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- Income >  110.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 5
|   |   |   |   |   |   |   |   |   |--- ZIPCode >  93205.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- Family >  3.50
|   |   |   |   |   |   |   |   |   |--- ZIPCode <= 94887.00
|   |   |   |   |   |   |   |   |   |   |--- Age <= 34.00
|   |   |   |   |   |   |   |   |   |   |   |--- CreditCard <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |   |   |   |   |   |   |   |--- CreditCard >  0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- Age >  34.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.70] class: 1
|   |   |   |   |   |   |   |   |   |--- ZIPCode >  94887.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |--- Experience >  31.50
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- CCAvg >  2.85
|   |   |   |   |--- Age <= 63.50
|   |   |   |   |   |--- Family <= 2.50
|   |   |   |   |   |   |--- ZIPCode <= 95033.50
|   |   |   |   |   |   |   |--- ZIPCode <= 90389.50
|   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |--- ZIPCode >  90389.50
|   |   |   |   |   |   |   |   |--- CCAvg <= 2.95
|   |   |   |   |   |   |   |   |   |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- CCAvg >  2.95
```

```
|   |   |   |   |   |   |   |   |   |--- Mortgage <= 210.50
|   |   |   |   |   |   |   |   |   |    |--- CD_Account <= 0.50
|   |   |   |   |   |   |   |   |   |    |    |--- weights: [0.00, 8.10] class: 1
|   |   |   |   |   |   |   |   |   |    |--- CD_Account >  0.50
|   |   |   |   |   |   |   |   |   |    |    |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |--- Mortgage >  210.50
|   |   |   |   |   |   |   |   |   |    |--- Experience <= 32.00
|   |   |   |   |   |   |   |   |   |    |    |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |    |--- Experience >  32.00
|   |   |   |   |   |   |   |   |   |    |    |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |   |   |   |--- ZIPCode >  95033.50
|   |   |   |   |   |   |   |    |--- weights: [0.20, 0.00] class: 0
|   |   |   |   |   |   |--- Family >  2.50
|   |   |   |   |   |   |    |--- Age <= 62.50
|   |   |   |   |   |   |    |    |--- ZIPCode <= 90687.00
|   |   |   |   |   |   |    |    |    |--- CreditCard <= 0.50
|   |   |   |   |   |   |    |    |    |    |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |   |   |    |    |    |--- CreditCard >  0.50
|   |   |   |   |   |   |    |    |    |    |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |   |    |    |--- ZIPCode >  90687.00
|   |   |   |   |   |   |    |    |    |--- CCAvg <= 2.95
|   |   |   |   |   |   |    |    |    |    |--- weights: [0.00, 0.90] class: 1
|   |   |   |   |   |   |    |    |    |--- CCAvg >  2.95
|   |   |   |   |   |   |    |    |    |    |--- weights: [0.00, 14.40] class: 1
|   |   |   |   |   |   |    |--- Age >  62.50
|   |   |   |   |   |   |    |    |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |--- Age >  63.50
|   |   |   |   |   |    |--- Experience <= 39.50
|   |   |   |   |   |    |    |--- weights: [0.10, 0.00] class: 0
|   |   |   |   |   |    |--- Experience >  39.50
|   |   |   |   |   |    |    |--- weights: [0.10, 0.00] class: 0
|   |   |   |--- Income >  116.50
|   |   |   |    |--- weights: [0.00, 199.80] class: 1
```

- The tree above is a bit too complicated.

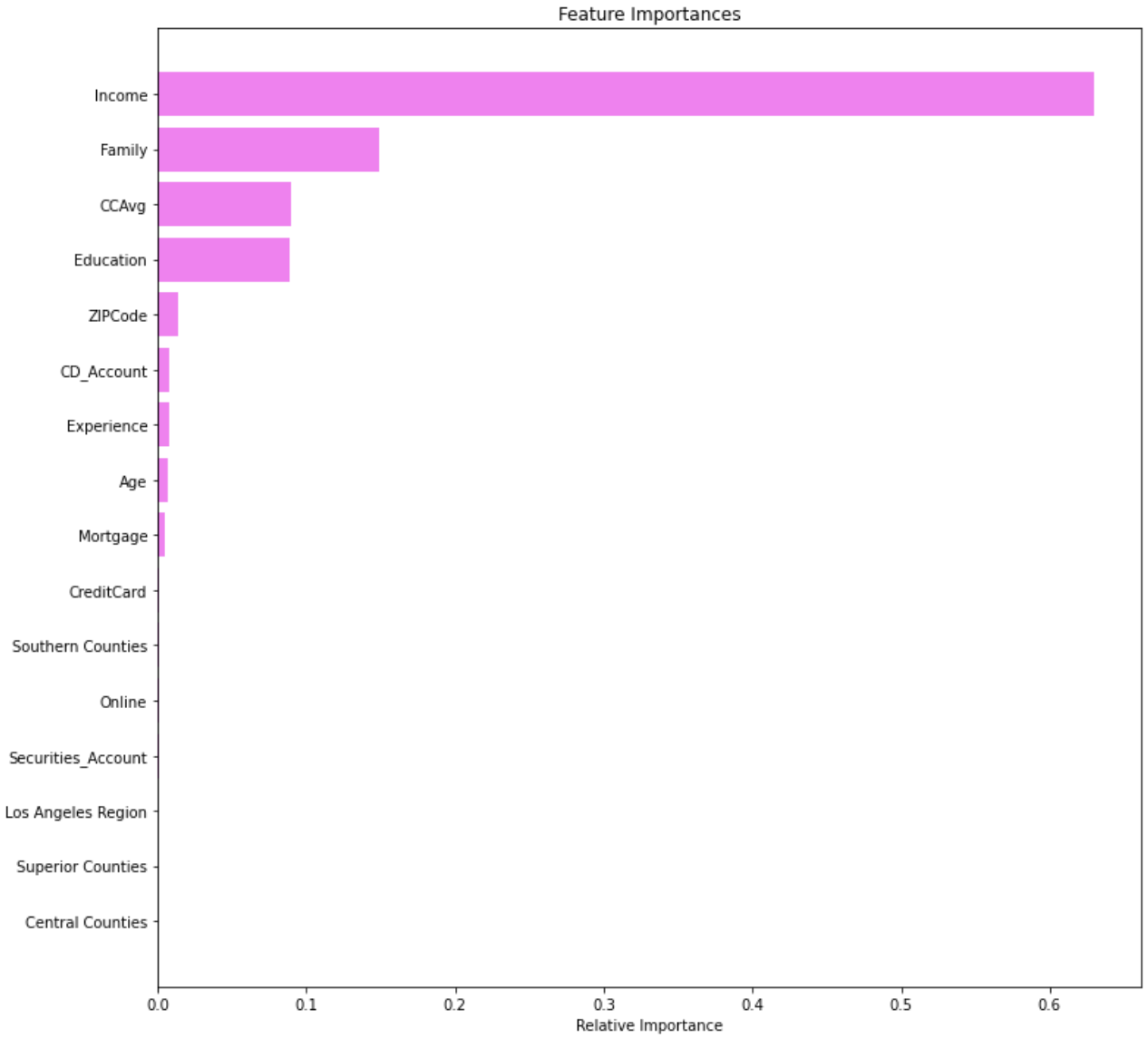
Going to find which features are most important

In [163…

```
print(
    pd.DataFrame(
        model2.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                         Imp
Income              6.297688e-01
Family              1.486462e-01
CCAvg               8.974937e-02
Education           8.882049e-02
ZIPCode             1.344637e-02
CD_Account          7.748466e-03
Experience          7.691054e-03
Age                 6.224695e-03
Mortgage            5.056710e-03
CreditCard          1.172232e-03
Southern Counties   5.861161e-04
Online              5.456943e-04
Securities_Account  5.437069e-04
Los Angeles Region  7.157915e-18
Central Counties    0.000000e+00
Superior Counties   0.000000e+00
```

In [164…
```
importances = model2.feature_importances_
indices = np.argsort(importances)

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



- The main gini importance features are Income, Family, CCAvg, and Education.

Using GridSearch for Hyperparameter tuning of the tree model

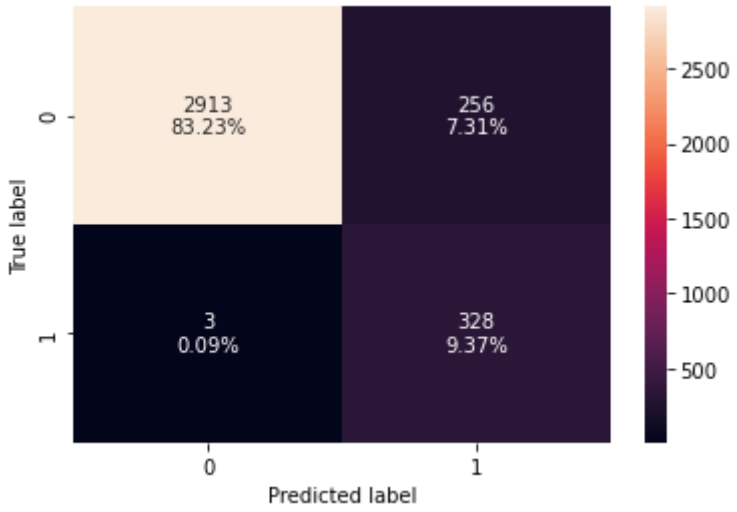
In [165…
```
estimator = DecisionTreeClassifier(random_state=1, class_weight={0: 0.10, 1: 0.9

parameters = {
    "max_depth": [5, 10, 15, None],
```

```
    "criterion": ["entropy", "gini"],
    "splitter": ["best", "random"],
    "min_impurity_decrease": [0.00001, 0.0001, 0.01, 0.1],
}

scorer = make_scorer(recall_score)

grid_obj = GridSearchCV(estimator, parameters, scoring=scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

estimator = grid_obj.best_estimator_

estimator.fit(X_train, y_train)
```

Out[165…
```
DecisionTreeClassifier(class_weight={0: 0.1, 1: 0.9}, max_depth=5,
                       min_impurity_decrease=0.01, random_state=1)
```

Checking performance on train and test data

In [166…
```
confusion_matrix_sklearn(estimator, X_train, y_train)
```



In [167…
```
decision_tree_tune_perf_train = get_recall_score(estimator, X_train, y_train)
print("Recall Score:", decision_tree_tune_perf_train)
```

```
Recall Score: 0.9909365558912386
```

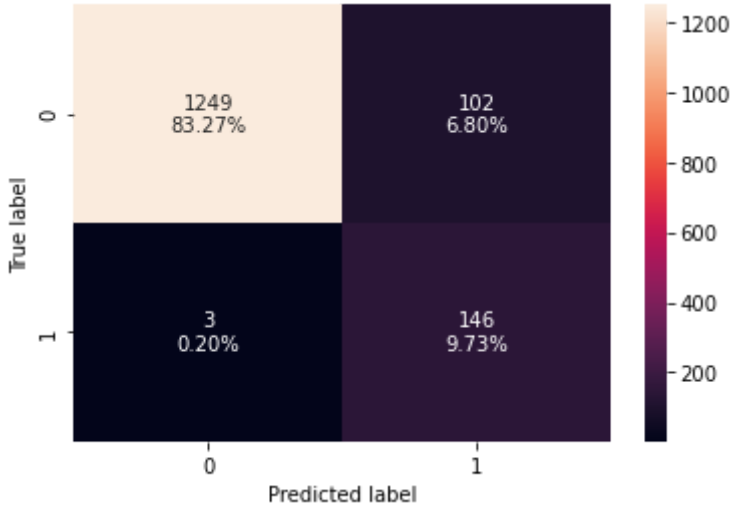
In [168…
```
dec_tree_model_train_perf = model_performance_classification_sklearn(
    estimator, X_train, y_train
)

print("Training performance:")
dec_tree_model_train_perf
```

```
Training performance:
```

Out[168…

	Accuracy	Recall	Precision	F1
0	0.926	0.990937	0.561644	0.71694

In [169…
```
confusion_matrix_sklearn(estimator, X_test, y_test)
```



In [170…
```
decision_tree_tune_perf_test = get_recall_score(estimator, X_test, y_test)
print("Recall Score:", decision_tree_tune_perf_test)
```

```
Recall Score: 0.9798657718120806
```

In [171…
```
dec_tree_model_test_perf = model_performance_classification_sklearn(
    estimator, X_test, y_test
)

print("Test set performance:")
dec_tree_model_test_perf
```

```
Test set performance:
```

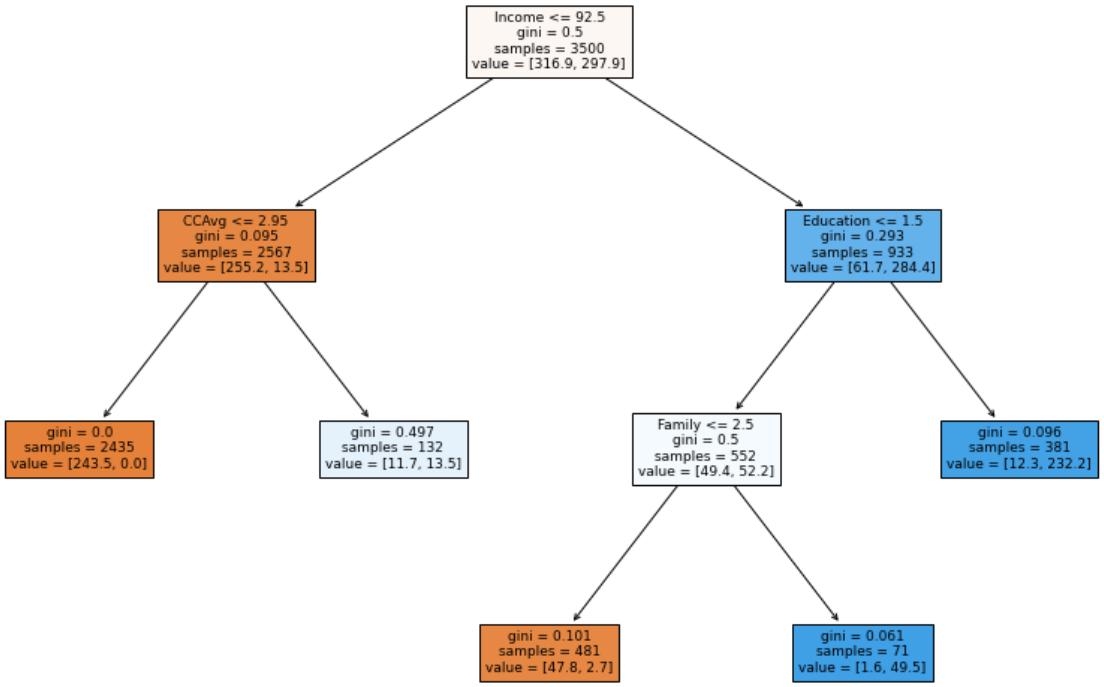
Out[171…

	Accuracy	Recall	Precision	F1
0	0.93	0.979866	0.58871	0.735516

- The train and test scores aren't as different now.

Visualization of estimator model

In [172…
```
plt.figure(figsize=(15, 10))
out = tree.plot_tree(
    estimator,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



In [173…
```
print(tree.export_text(estimator, feature_names=feature_names, show_weights=True
```

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [243.50, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- weights: [11.70, 13.50] class: 1
|--- Income >  92.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [47.80, 2.70] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [1.60, 49.50] class: 1
|   |--- Education >  1.50
|   |   |--- weights: [12.30, 232.20] class: 1
```

This is easier to read and interpret

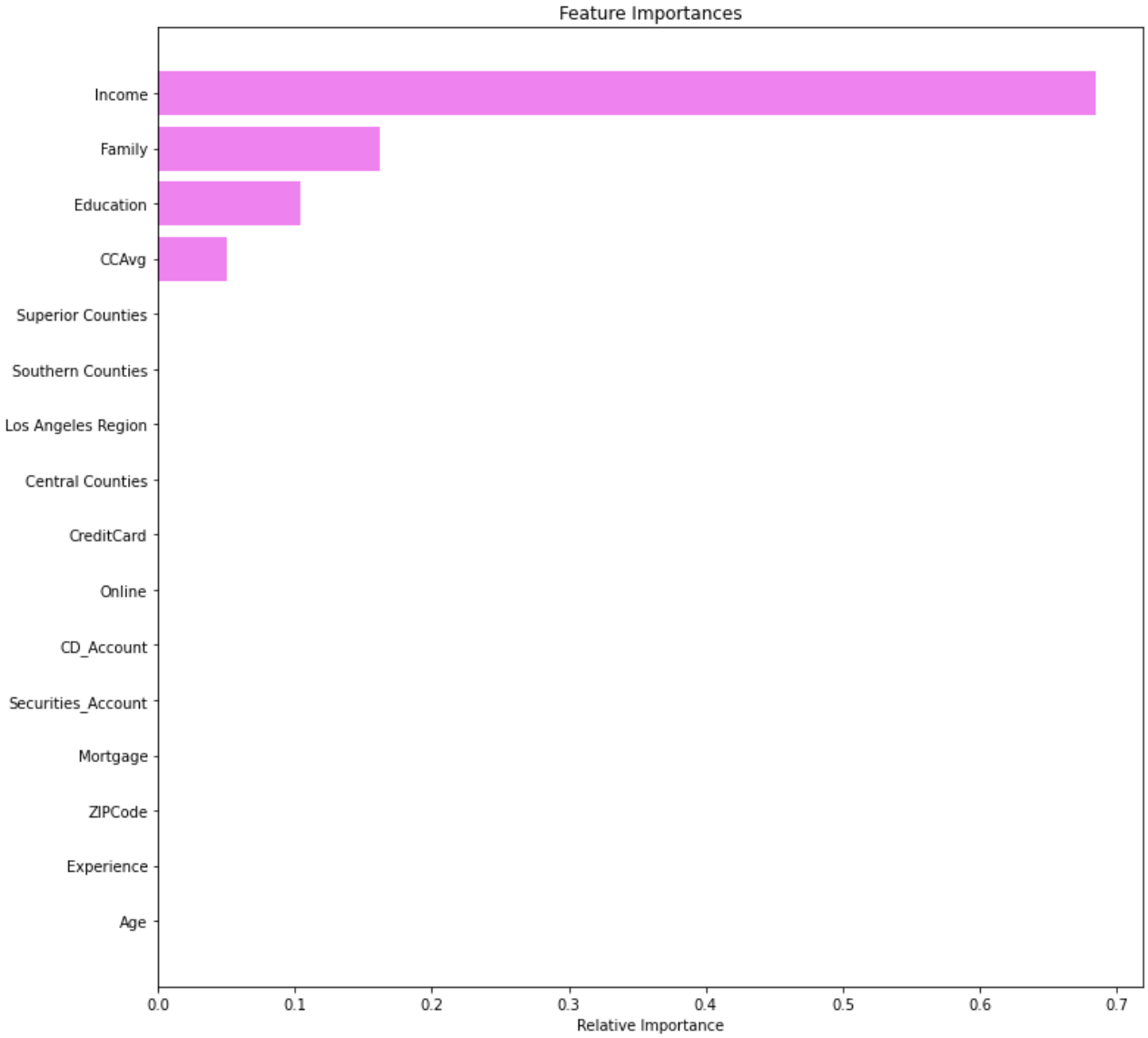
- If someone's income is less than or equal to 92.50 and has a CCAvg of less than or equal to 2.95, they will not recieve a personal loan. If they have a CCAvg of greater than 2.95, they are more likely to recieve a personal loan.
- If someone has an income greater than 92.50 with an education of 1.50 or lower with a family size greater than 2.5, they are likely to recieve a personal loan. If they have an education level greater than 1.5, they will recieve a personal loan.

Gini Importance of estimator model

In [174…
```
print(
    pd.DataFrame(
        estimator.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                         Imp
Income              0.684653
Family              0.161790
Education           0.103717
CCAvg               0.049840
Age                 0.000000
Experience          0.000000
ZIPCode             0.000000
Mortgage            0.000000
Securities_Account  0.000000
CD_Account          0.000000
Online              0.000000
CreditCard          0.000000
Central Counties    0.000000
Los Angeles Region  0.000000
Southern Counties   0.000000
Superior Counties   0.000000
```

In [175…
```
importances = estimator.feature_importances_
indices = np.argsort(importances)

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



- The feature importances for this model are Income, Family, Education, and CCAvg.

Cost Complexity Pruning

In [176…
```
clf = DecisionTreeClassifier(random_state=1, class_weight={0: 0.10, 1: 0.90})
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

In [177…
```
pd.DataFrame(path)
```

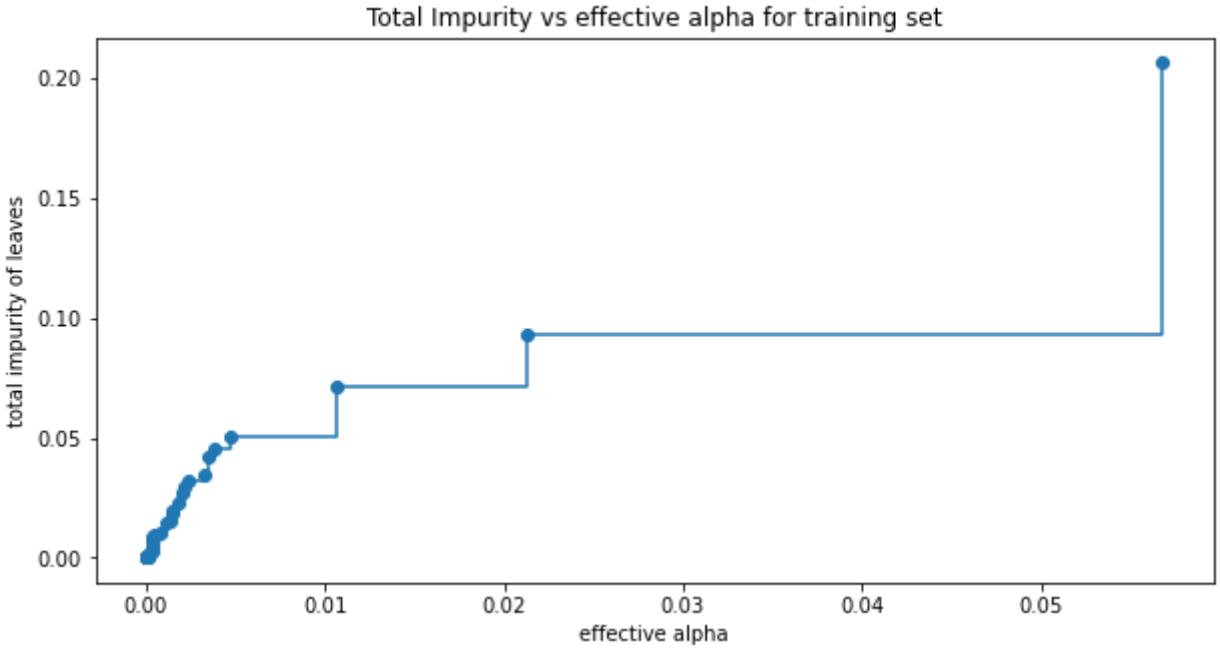
Out[177…

	ccp_alphas	impurities
0	0.000000e+00	-2.551617e-15
1	3.792239e-19	-2.551238e-15
2	2.762917e-18	-2.548475e-15
3	3.575539e-18	-2.544899e-15
4	9.029140e-18	-2.535870e-15

	ccp_alphas	impurities
5	2.275343e-17	-2.513117e-15
6	4.543463e-17	-2.467682e-15
7	2.419448e-16	-2.225737e-15
8	5.936208e-15	3.710471e-15
9	1.568454e-04	3.136909e-04
10	1.582585e-04	6.302078e-04
11	1.608671e-04	9.519420e-04
12	1.616566e-04	1.275255e-03
13	1.623142e-04	1.599884e-03
14	2.823218e-04	2.164527e-03
15	2.927781e-04	2.750084e-03
16	2.927781e-04	3.335640e-03
17	2.998581e-04	3.635498e-03
18	3.193704e-04	3.954868e-03
19	3.225996e-04	4.600068e-03
20	3.460105e-04	6.676131e-03
21	3.614387e-04	8.844763e-03
22	4.940631e-04	9.338826e-03
23	5.990667e-04	9.937893e-03
24	7.828292e-04	1.072072e-02
25	1.164102e-03	1.421303e-02
26	1.372398e-03	1.558543e-02
27	1.416358e-03	1.841814e-02
28	1.435187e-03	1.985333e-02
29	1.743294e-03	2.333992e-02
30	1.985898e-03	2.731171e-02
31	2.127748e-03	2.943946e-02
32	2.328917e-03	3.176838e-02
33	3.240232e-03	3.500861e-02
34	3.470671e-03	4.194995e-02
35	3.841577e-03	4.579153e-02
36	4.760264e-03	5.055179e-02
37	1.059696e-02	7.174571e-02
38	2.132036e-02	9.306606e-02
39	5.678893e-02	2.066439e-01

	ccp_alphas	impurities
40	2.928785e-01	4.995225e-01

In [178…
```
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()
```



Going to train a decision tree using the effective alphas.

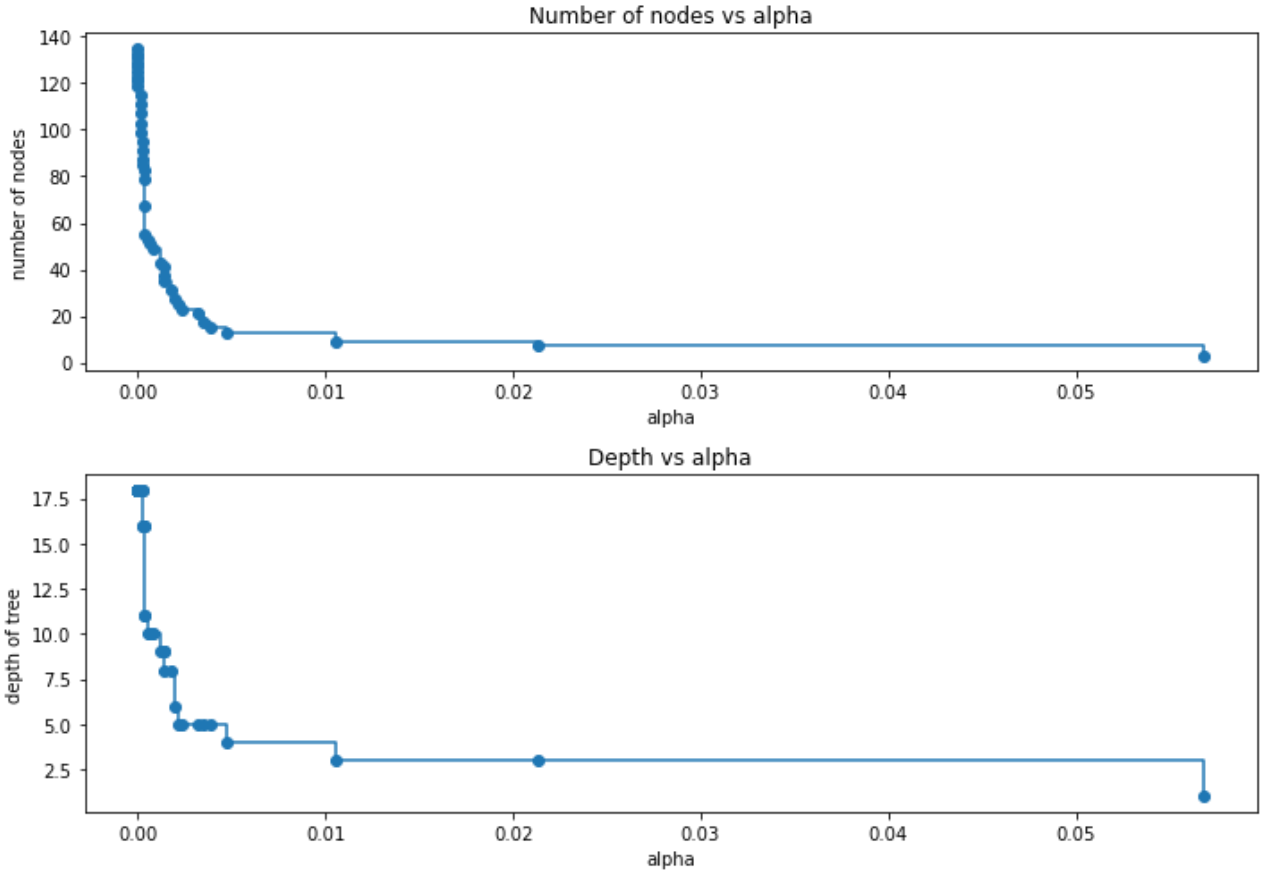
In [179…
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(
        random_state=1, ccp_alpha=ccp_alpha, class_weight={0: 0.10, 1: 0.90}
    )
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.2928785401980033
```

Going to remove the last element in clfs and ccp_alphas as it is the trivial tree with only one node.

In [180…
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
```

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



- As alpha increases, number of nodes and depth of tree decreases.

In [181…

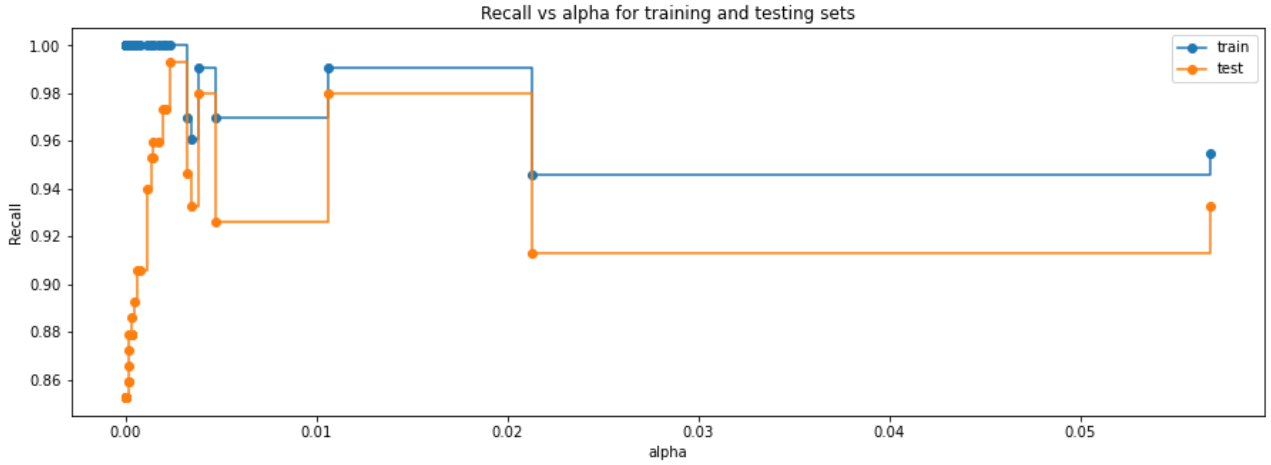
```
recall_train = []
for clf in clfs:
    pred_train = clf.predict(X_train)
    values_train = recall_score(y_train, pred_train)
    recall_train.append(values_train)
```

In [182…

```
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)
```

In [183…
```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
```

In [184…
```
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()
```



Creating a model that will give me the highest train and test recall.

In [185…
```
index_best_model = np.argmax(recall_test)
best_model = clfs[index_best_model]
print(best_model)
```

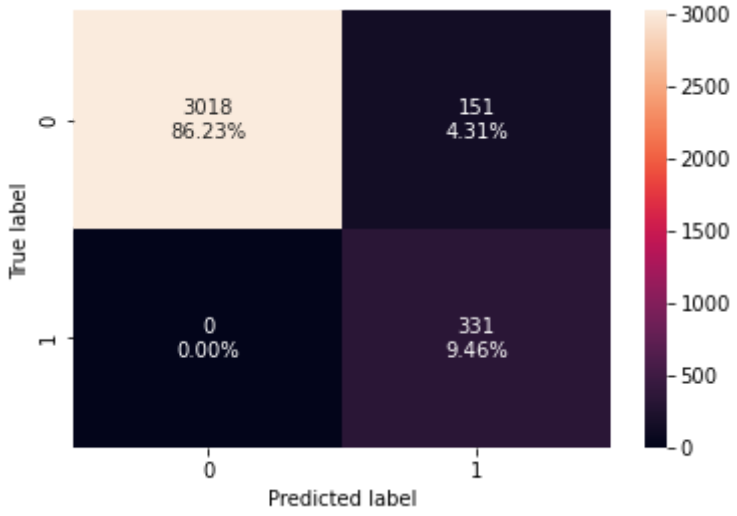
```
DecisionTreeClassifier(ccp_alpha=0.0023289170166201478,
                       class_weight={0: 0.1, 1: 0.9}, random_state=1)
```

In [186…
```
best_model.fit(X_train, y_train)
```

Out[186…
```
DecisionTreeClassifier(ccp_alpha=0.0023289170166201478,
                       class_weight={0: 0.1, 1: 0.9}, random_state=1)
```

Checking performance on training and test data

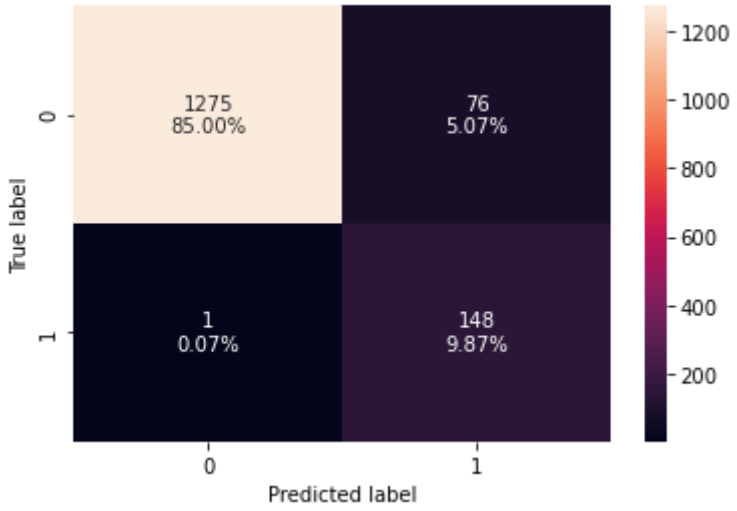
In [187…
```
confusion_matrix_sklearn(best_model, X_train, y_train)
```



In [188…
```
decision_tree_postpruned_perf_train = get_recall_score(best_model, X_train, y_tr
print("Recall Score:", decision_tree_postpruned_perf_train)
```

```
Recall Score: 1.0
```

In [189…
```
confusion_matrix_sklearn(best_model, X_test, y_test)
```



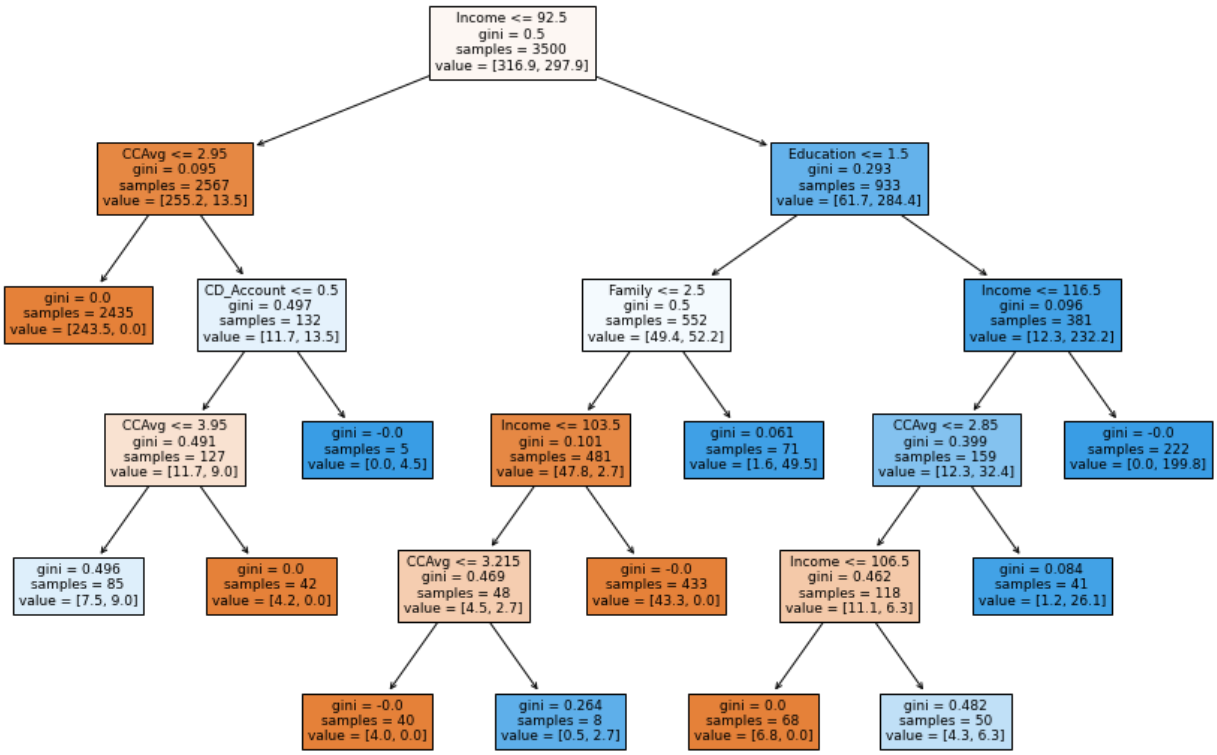
In [190…
```
decision_tree_postpruned_perf_test = get_recall_score(best_model, X_test, y_test
print("Recall Score:", decision_tree_postpruned_perf_test)
```

```
Recall Score: 0.9932885906040269
```

Visualization of best_model

In [191…
```
plt.figure(figsize=(15, 10))

out = tree.plot_tree(
    best_model,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
```

```
    class_names=None,
)
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



In [192…

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

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [243.50, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- CD_Account <= 0.50
|   |   |   |--- CCAvg <= 3.95
|   |   |   |   |--- weights: [7.50, 9.00] class: 1
|   |   |   |--- CCAvg >  3.95
|   |   |   |   |--- weights: [4.20, 0.00] class: 0
|   |   |--- CD_Account >  0.50
|   |   |   |--- weights: [0.00, 4.50] class: 1
|--- Income >  92.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- Income <= 103.50
|   |   |   |   |--- CCAvg <= 3.21
|   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |--- CCAvg >  3.21
|   |   |   |   |   |--- weights: [0.50, 2.70] class: 1
|   |   |   |--- Income >  103.50
|   |   |   |   |--- weights: [43.30, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [1.60, 49.50] class: 1
|   |--- Education >  1.50
```

```
|   |   |--- Income <= 116.50
|   |   |   |--- CCAvg <= 2.85
|   |   |   |   |--- Income <= 106.50
|   |   |   |   |   |--- weights: [6.80, 0.00] class: 0
|   |   |   |   |--- Income >  106.50
|   |   |   |   |   |--- weights: [4.30, 6.30] class: 1
|   |   |   |--- CCAvg >  2.85
|   |   |   |   |--- weights: [1.20, 26.10] class: 1
|   |   |--- Income >  116.50
|   |   |   |--- weights: [0.00, 199.80] class: 1
```

Importances

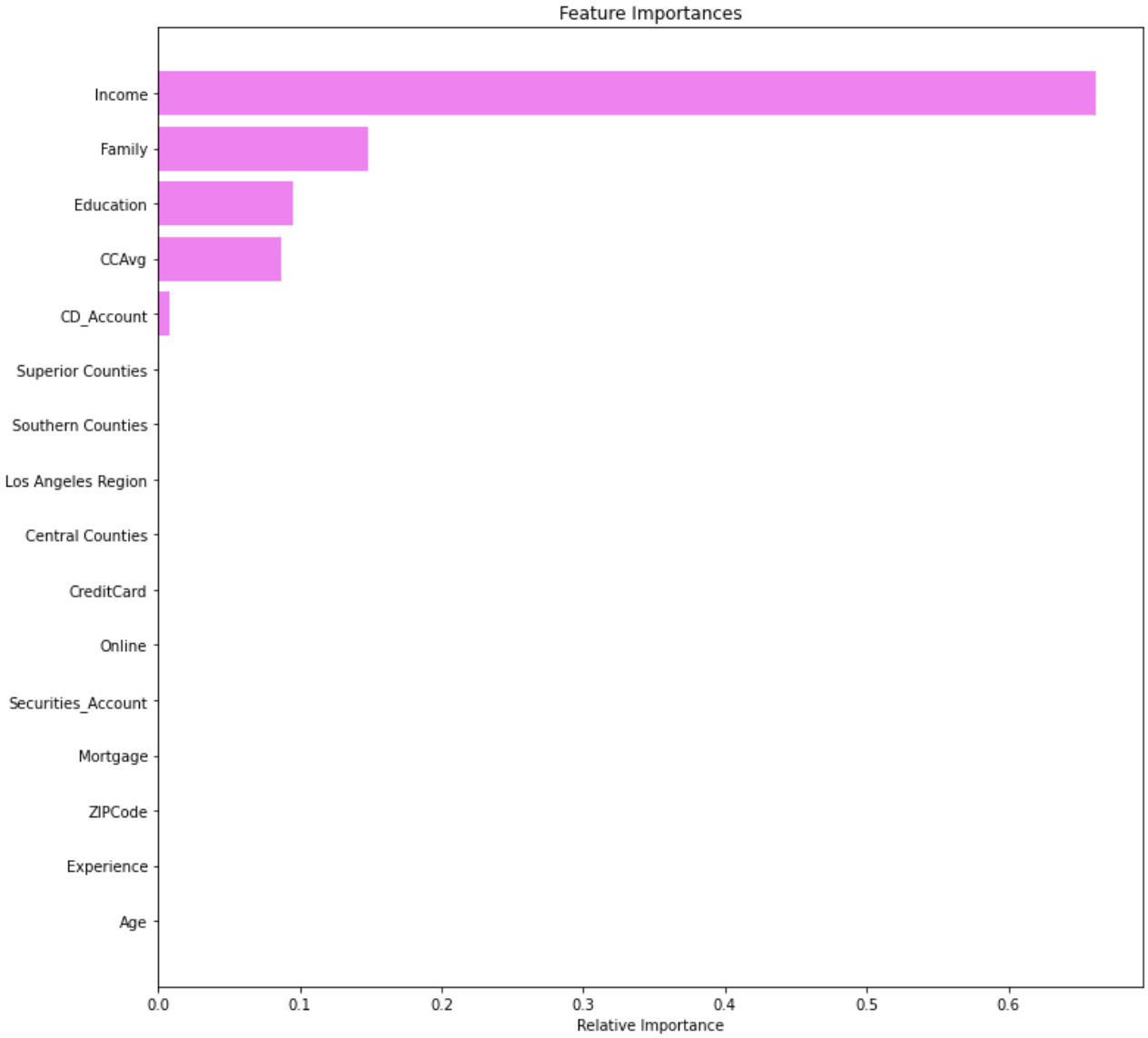
In [193…

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

```
                       Imp
Income            0.661588
Family            0.147962
Education         0.094853
CCAvg             0.087384
CD_Account        0.008213
Age               0.000000
Experience        0.000000
ZIPCode           0.000000
Mortgage          0.000000
Securities_Account  0.000000
Online            0.000000
CreditCard        0.000000
Central Counties  0.000000
Los Angeles Region  0.000000
Southern Counties  0.000000
Superior Counties  0.000000
```

In [194…

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

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



- Importances are Income, Family, Education, CCAvg, and CD_Account.

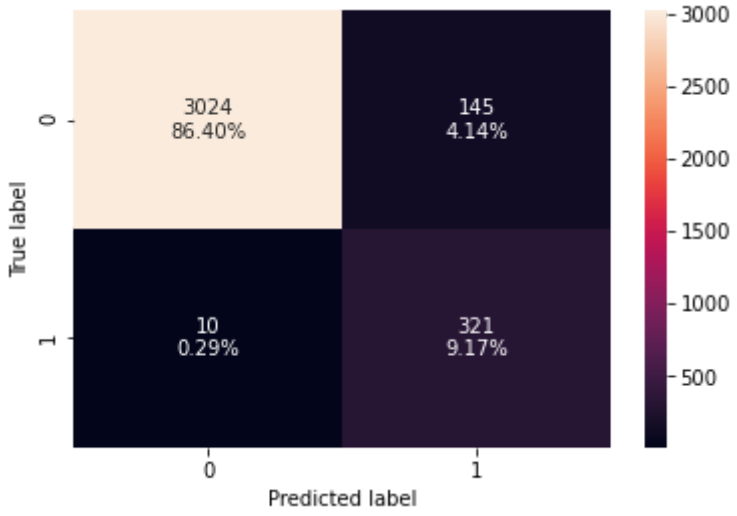
Going to create a model with an alpha of 0.01

In [195…
```
best_model2 = DecisionTreeClassifier(
    ccp_alpha=0.01, class_weight={0: 0.10, 1: 0.90}, random_state=1
)
best_model2.fit(X_train, y_train)
```

Out[195…
```
DecisionTreeClassifier(ccp_alpha=0.01, class_weight={0: 0.1, 1: 0.9},
                       random_state=1)
```

Checking performance on train and test data

In [196…
```
confusion_matrix_sklearn(best_model2, X_train, y_train)
```



In [197…
```
decision_tree_postpruned_perf_train2 = get_recall_score(best_model2, X_train, y_
print("Recall Score:", decision_tree_postpruned_perf_train2)
```

```
Recall Score: 0.9697885196374623
```

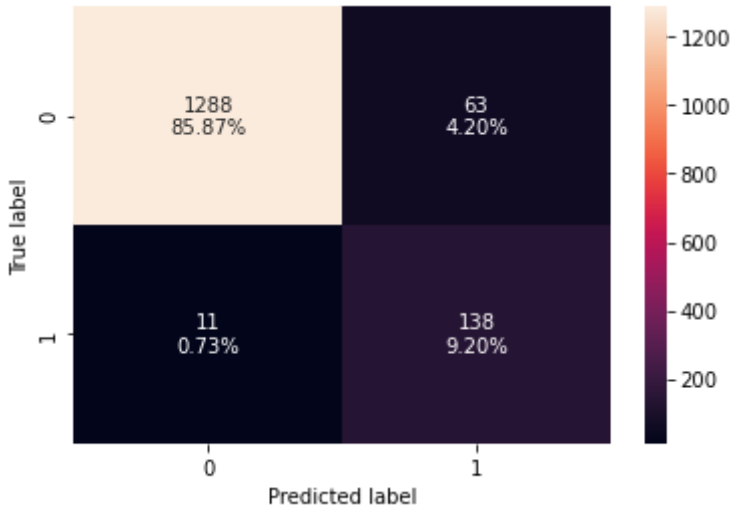
In [198…
```
dec_tree_model_train_perf = model_performance_classification_sklearn(
    best_model2, X_train, y_train
)

print("Training performance:")
dec_tree_model_train_perf
```

```
Training performance:
```

Out[198…

	Accuracy	Recall	Precision	F1
0	0.955714	0.969789	0.688841	0.805521

In [199…
```
confusion_matrix_sklearn(best_model2, X_test, y_test)
```



In [200…
```
decision_tree_postpruned_perf_test2 = get_recall_score(best_model2, X_test, y_te
print("Recall Score:", decision_tree_postpruned_perf_test2)
```

```
Recall Score: 0.9261744966442953
```

In [201…

```
dec_tree_model_test_perf = model_performance_classification_sklearn(
    best_model2, X_test, y_test
)

print("Test set performance:")
dec_tree_model_test_perf
```

```
Test set performance:
```

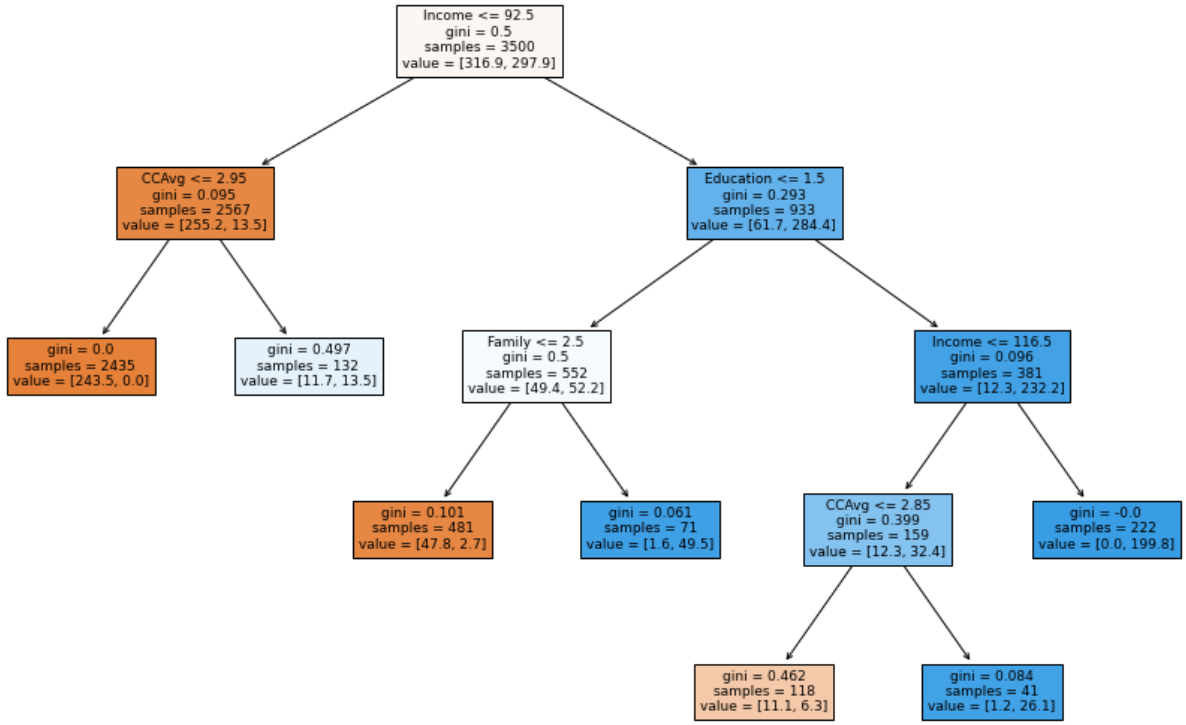
Out[201…

	Accuracy	Recall	Precision	F1
0	0.950667	0.926174	0.686567	0.788571

Visualization of best_model2

In [202…

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

out = tree.plot_tree(
    best_model2,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



In [203…
```
print(tree.export_text(best_model2, feature_names=feature_names, show_weights=Tr
```

```
|--- Income <= 92.50
|   |--- CCAvg <= 2.95
|   |   |--- weights: [243.50, 0.00] class: 0
|   |--- CCAvg >  2.95
|   |   |--- weights: [11.70, 13.50] class: 1
|--- Income >  92.50
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [47.80, 2.70] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [1.60, 49.50] class: 1
|   |--- Education >  1.50
|   |   |--- Income <= 116.50
|   |   |   |--- CCAvg <= 2.85
|   |   |   |   |--- weights: [11.10, 6.30] class: 0
|   |   |   |--- CCAvg >  2.85
|   |   |   |   |--- weights: [1.20, 26.10] class: 1
|   |   |--- Income >  116.50
|   |   |   |--- weights: [0.00, 199.80] class: 1
```

Importances

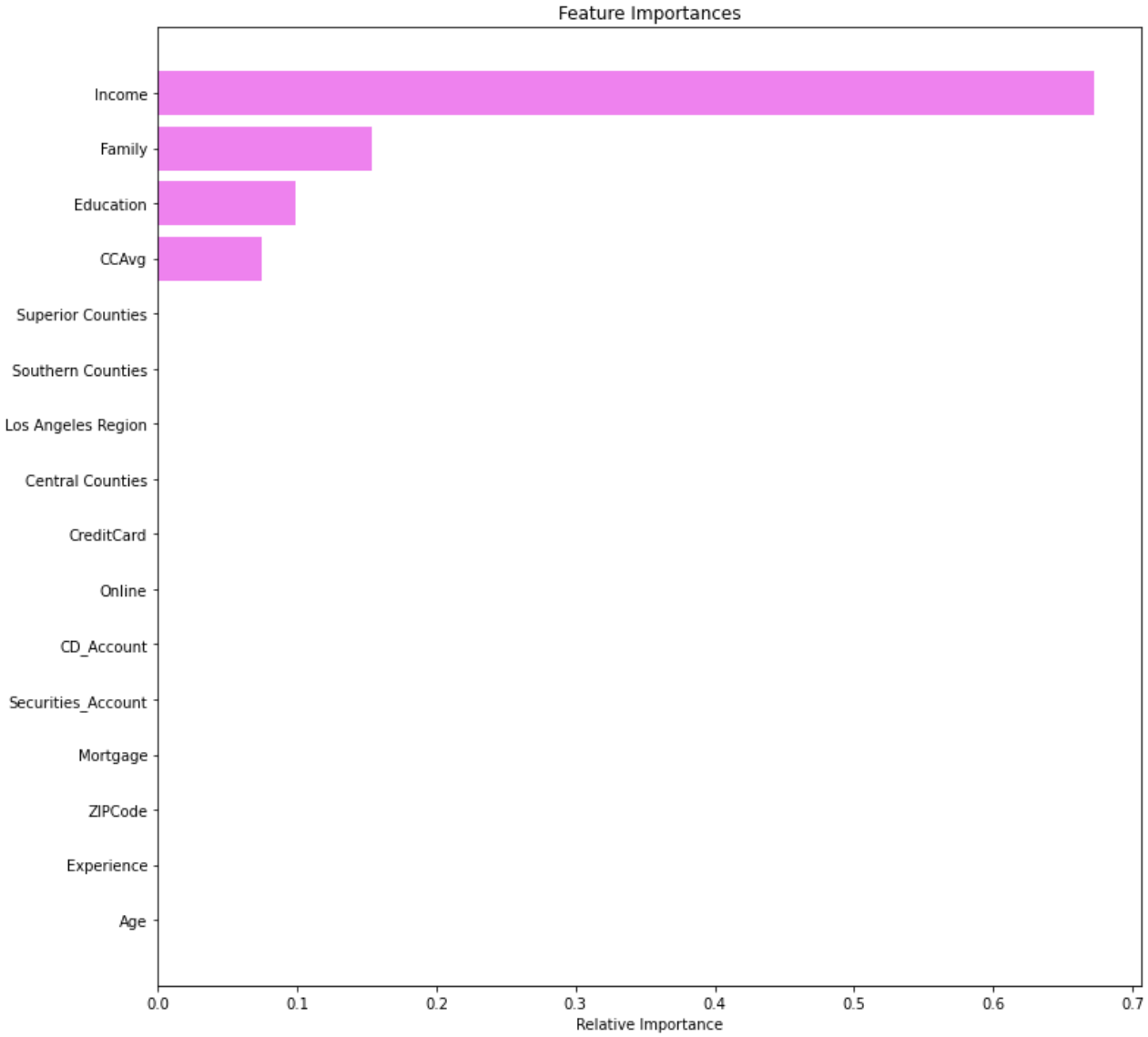
In [204…
```
print(
    pd.DataFrame(
        best_model2.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                    Imp
Income         0.672373
```

```
Family              0.154153
Education           0.098821
CCAvg               0.074653
Age                 0.000000
Experience          0.000000
ZIPCode             0.000000
Mortgage            0.000000
Securities_Account  0.000000
CD_Account          0.000000
Online              0.000000
CreditCard          0.000000
Central Counties    0.000000
Los Angeles Region  0.000000
Southern Counties   0.000000
Superior Counties   0.000000
```

In [205…

```
importances = best_model2.feature_importances_
indices = np.argsort(importances)

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



- Importances are Income, Family, Education, and CCAvg.

Comparing the different decision tree models

In [206…

```
models_train_comp_df = pd.DataFrame(
    [
        decision_tree_perf_train,
        decision_tree_tune_perf_train,
        decision_tree_postpruned_perf_train,
        decision_tree_postpruned_perf_train2,
    ],
    columns=["Recall on training set"],
)

print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[206…

	Recall on training set
0	1.000000

	Recall on training set
1	0.99933
2	1.000000
3	0.969789

In [207…
```
models_test_comp_df = pd.DataFrame(
    [
        decision_tree_perf_test,
        decision_tree_tune_perf_test,
        decision_tree_postpruned_perf_test,
        decision_tree_postpruned_perf_test2,
    ],
    columns=["Recall on testing set"],
)

print("Test performance comparison:")
models_test_comp_df
```

```
Test performance comparison:
```

Out[207…

	Recall on testing set
0	0.852349
1	0.979866
2	0.993289
3	0.926174

Conclusion

- Would not use the first model (0) as it is too complex and overfits the train data.
- The second or pre-pruned model (1) is a lot better than model (0) and is definitely usable.
- The third or post-pruned model (2) has the highest recall, but seems a little too complex and it will be harder to identify potential customers.
- The fourth or second post-pruned model (3) is similar to model (1) in that it is giving a generalized performance, but has a bit more information in terms of income. It also has a better precision and f1 score than model (1). This is the model I would use.

- The Logistic Regression model I would use has a threshold of 0.325. It has a good f1 score, so there is a good recall/precision balance.

Actionable Insights & Recommendations

Target people with an income higher than 92.50k, a family size greater than 2.5, an education level of 1.5 or higher, and a CCAvg of 2.85 or higher.

These are pretty much all uncontrollable factors, but the CCAvg variable can be influenced. By having a system that promotes credit card use could potentially increase the probability of someone accepting a personal loan.