Personal Loan Campaign

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

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

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

Objective

- To predict whether a liability customer will buy a personal loan or not.
- Which variables are most significant.
- Which segment of customers should be targeted more.

Data Dictionary

- ID: Customer ID
- Age: Customer’s age in completed years
- Experience: #years of professional experience
- Income: Annual income of the customer (in thousand dollars)
- ZIP Code: Home Address ZIP code.
- Family: the Family size of the customer
- CCAvg: Average spending on credit cards per month (in thousand dollars)
- Education: Education Level. 1: Undergrad; 2: Graduate;3: Advanced/Professional
- Mortgage: Value of house mortgage if any. (in thousand dollars)
- Personal_Loan: Did this customer accept the personal loan offered in the last campaign?
- Securities_Account: Does the customer have securities account with the bank?
- CD_Account: Does the customer have a certificate of deposit (CD) account with the bank?
- Online: Do customers use internet banking facilities?
- CreditCard: Does the customer use a credit card issued by any other Bank (excluding All life Bank)?

Import the necessary packages

In [208…

```
# Library to suppress warnings or deprecation notes
import warnings

warnings.filterwarnings("ignore")

# Libraries to help with reading and manipulating data

import pandas as pd
import numpy as np

# Library to split data
from sklearn.model_selection import train_test_split

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

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)

# Libraries to build decision tree classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import PowerTransformer
from sklearn.linear_model import LogisticRegression
from sklearn import tree

# To tune different models
from sklearn.model_selection import GridSearchCV

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

Read the dataset

In [209…

```
data = pd.read_csv("Loan_Modelling.csv")
```

In [210…

```
# copying data to another varaible to avoid any changes to original data
loan = data.copy()
```

Understanding the structure of the data

View the first and last 5 rows of the dataset.

In [5]:
```
loan.head()
```

Out[5]:

	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 [6]:
```
loan.tail()
```

Out[6]:

	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	

Understand the shape of the dataset.

In [7]:
```
loan.shape
```

Out[7]:
```
(5000, 14)
```

Observation: The dataset has 5000 rows and 14 columns of data

Check the data types of the columns for the dataset.

In [8]:
```
loan.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
```

Observation: All the variables are int data type except CCAvg which is float

Summary of the dataset.

In [9]:
```
loan.describe(include="all")
```

Out[9]:

	ID	Age	Experience	Income	ZIPCode	Family	
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.(
mean	2500.500000	45.338400	20.104600	73.774200	93169.257000	2.396400	1.!
std	1443.520003	11.463166	11.467954	46.033729	1759.455086	1.147663	1.
min	1.000000	23.000000	-3.000000	8.000000	90005.000000	1.000000	0.(
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.7
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.!
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.!
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.(

In [18]:
```
# To check number of unique elements in each columns
loan.nunique()
```

Out[18]:
```
ID                    5000
Age                     45
Experience              47
Income                 162
ZIPCode                467
Family                   4
CCAvg                  108
Education                3
Mortgage               347
Personal_Loan            2
Securities_Account       2
CD_Account               2
Online                   2
CreditCard               2
dtype: int64
```

Observation:

- Since all the values in ID column are unique we can drop it
- Zip Code has 467 distinct value.

Since these fields will not affect our predictions we can drop it

Data Preprocessing

In [211…
```
loan.drop(["ID"], axis=1, inplace=True)
```

In [212…

```
loan.drop(["ZIPCode"], axis=1, inplace=True)
```

Check for missing values

In [213…

```
loan.isnull().sum()
```

Out[213…

```
Age                   0
Experience            0
Income                0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal_Loan         0
Securities_Account    0
CD_Account            0
Online                0
CreditCard            0
dtype: int64
```

Observation: There are no missing vaues in the dataset

Data Visualization - Univariate analysis

- Univariate analysis refer to the analysis of a single variable. The main purpose of univariate analysis is to summarize and find patterns in the data. The key point is that there is only one variable involved in the analysis.

  Let us take the loan dataset and work on that for the univariate analysis.

In [214…

```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

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

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 2, 6))
    else:
        plt.figure(figsize=(n + 2, 6))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
```

```
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

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

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

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

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

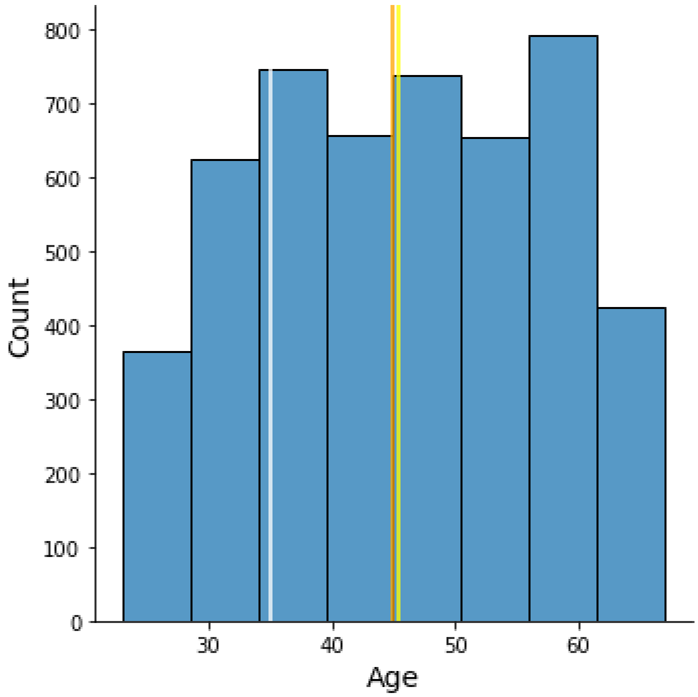
Observation on Age

In [215…

```
# plots a histogram plt using the seaborn package for Age column.
# Using displot since distplot going to be decommissioned in the future

sns.displot(loan,
            x = "Age",
            bins=8,
            height=5)
plt.xlabel("Age", size=14)
plt.ylabel("Count", size=14)
plt.axvline(x=loan.Age.mean(),
            color='yellow')
plt.axvline(x=loan.Age.median(),
            color='orange')
plt.axvline(x=loan.Age.mode()[0],
            color='white')
```

Out[215…

```
<matplotlib.lines.Line2D at 0x7feb397de160>
```



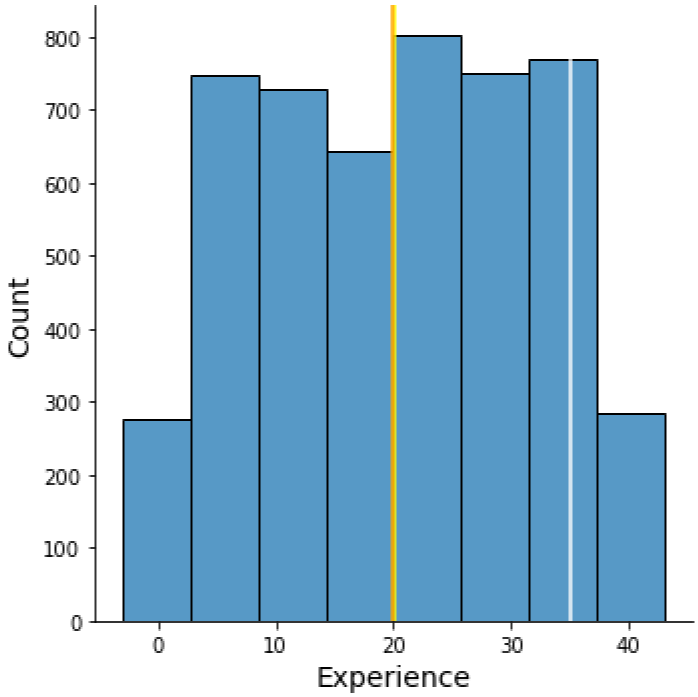
Observation on Experience

In [216…

```
# plots a histogram plt using the seaborn package for Experience column.
# Using displot since distplot going to be decommissioned in the future

sns.displot(loan,
            x = "Experience",
            bins=8,
            height=5)
plt.xlabel("Experience", size=14)
plt.ylabel("Count", size=14)
plt.axvline(x=loan.Experience.mean(),
            color='yellow')
plt.axvline(x=loan.Experience.median(),
            color='orange')
plt.axvline(x=loan.Age.mode()[0],
            color='white')
```

Out[216…

```
<matplotlib.lines.Line2D at 0x7feb398ebf70>
```



Observation: There are some negative experience found in the Experience column

In [217…

```
#To check the count of negative values
loan[loan['Experience'] < 0]['Experience'].value_counts()
```

Out[217…

```
-1    33
-2    15
-3     4
Name: Experience, dtype: int64
```

In [39]:

```
# Dropping Experience column since its highly correlated with Age column

loan.drop(["Experience"], axis=1, inplace=True)
```

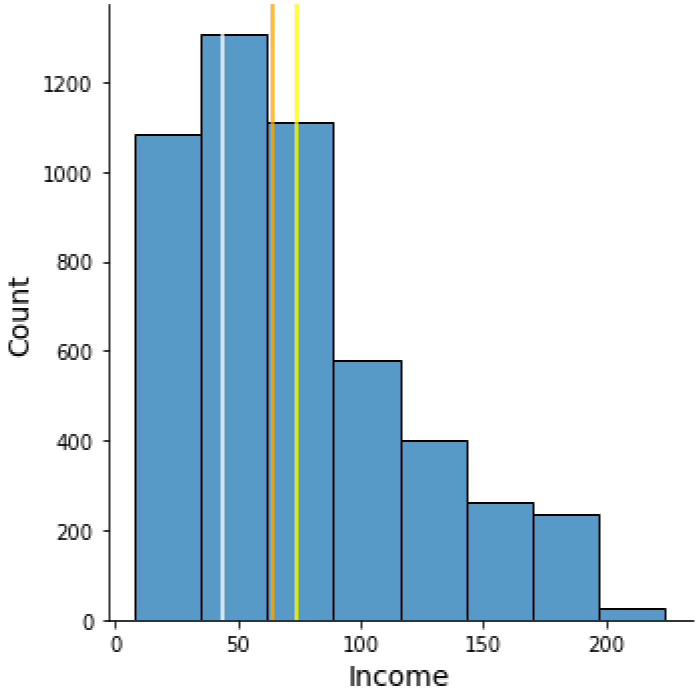
Observation on Income

In [30]:

```
# plots a histogram plt using the seaborn package for Income column.
# Using displot since distplot going to be decommissioned in the future

sns.displot(loan,
            x = "Income",
            bins=8,
            height=5)
plt.xlabel("Income", size=14)
plt.ylabel("Count", size=14)
plt.axvline(x=loan.Income.mean(),
            color='yellow')
plt.axvline(x=loan.Income.median(),
            color='orange')
plt.axvline(x=loan.Income.mode()[0],
            color='white')
```

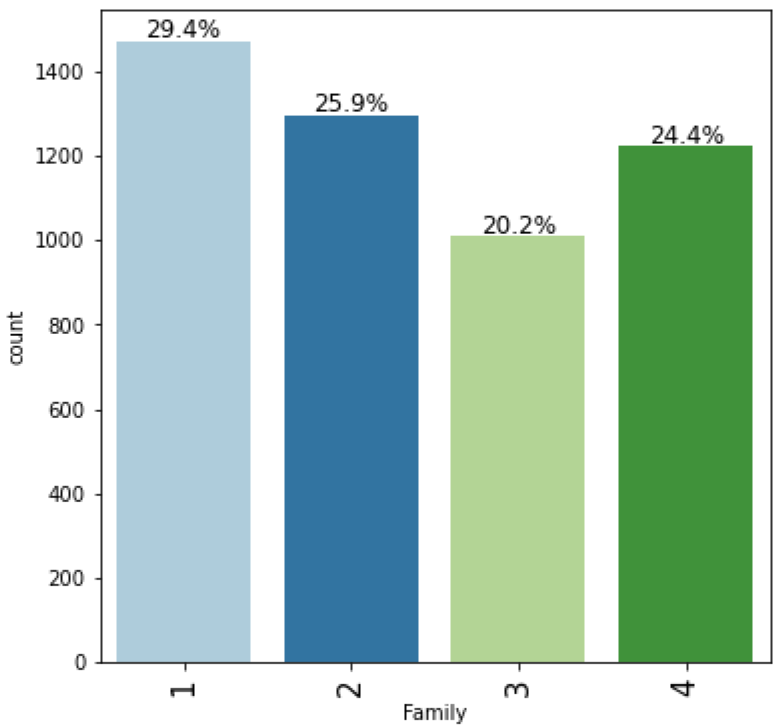
Out[30]:

```
<matplotlib.lines.Line2D at 0x7feb6a6c65b0>
```



Observations on Family

In [48]:
```
labeled_barplot(loan, "Family", perc=True)
```

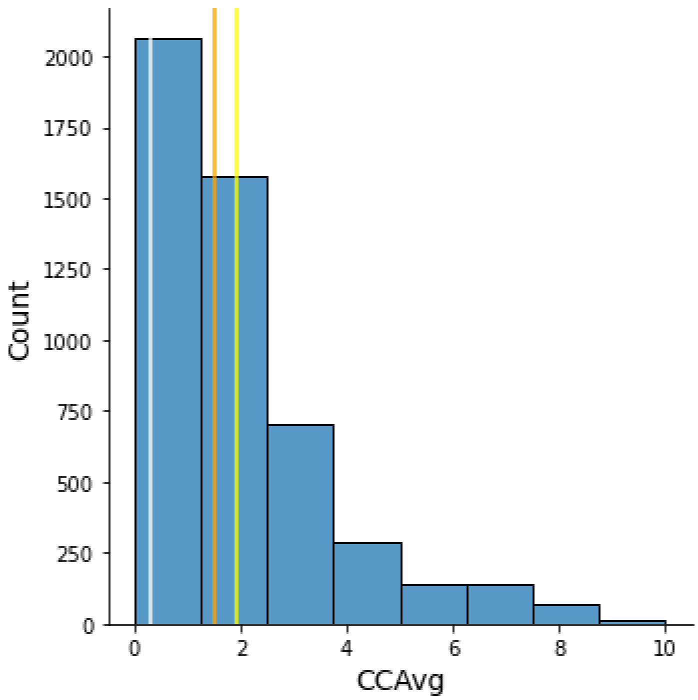


Observations on CCAvg

In [51]:
```
# plots a histogram plt using the seaborn package for CCAVG column.
# Using displot since distplot going to be decommissioned in the future

sns.displot(loan,
            x = "CCAvg",
```

```
             bins=8,
             height=5)
plt.xlabel("CCAvg", size=14)
plt.ylabel("Count", size=14)
plt.axvline(x=loan.CCAvg.mean(),
             color='yellow')
plt.axvline(x=loan.CCAvg.median(),
             color='orange')
plt.axvline(x=loan.CCAvg.mode()[0],
             color='white')
```

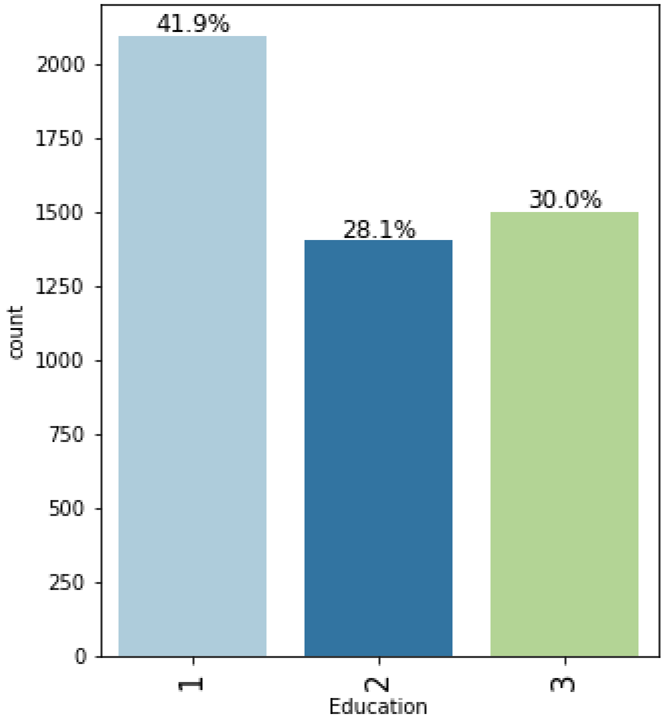
Out[51]: <matplotlib.lines.Line2D at 0x7feb6156c0a0>



Observations on Education

In [49]:

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

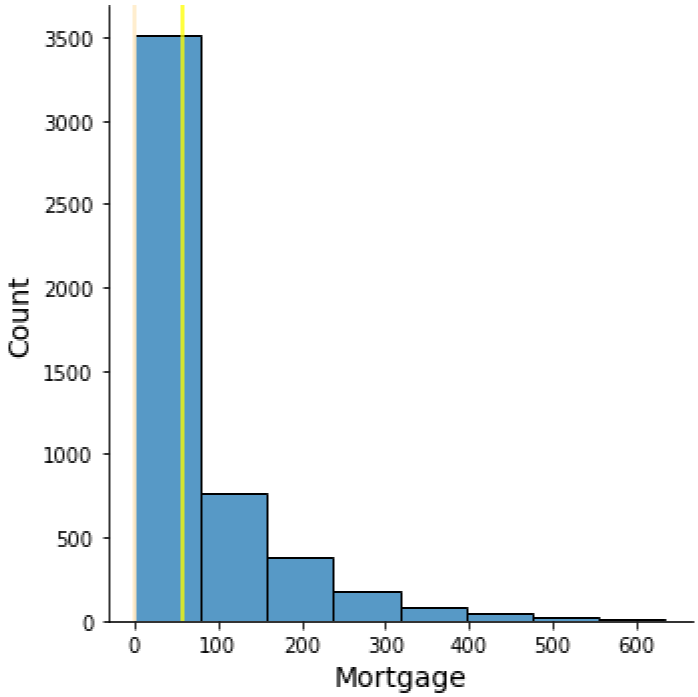


Observations on Mortgage

In [45]:

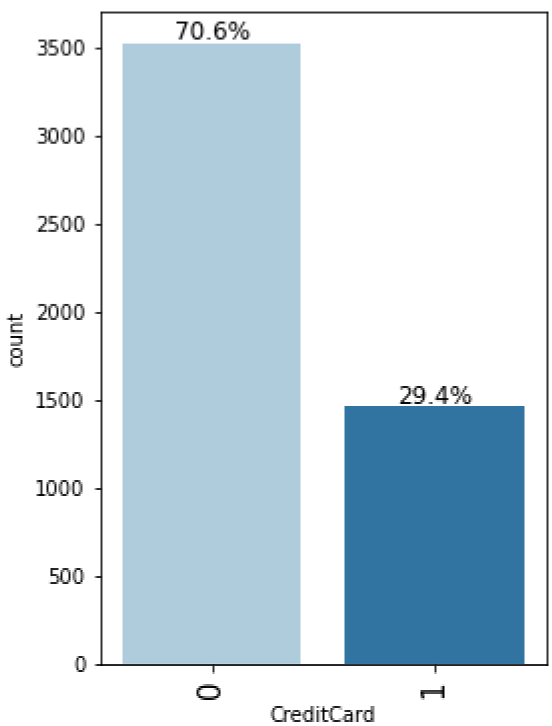
```
# plots a histogram plt using the seaborn package for Mortgage column.
# Using displot since distplot going to be decommissioned in the future

sns.displot(loan,
            x = "Mortgage",
            bins=8,
            height=5)
plt.xlabel("Mortgage", size=14)
plt.ylabel("Count", size=14)
plt.axvline(x=loan.Mortgage.mean(),
            color='yellow')
plt.axvline(x=loan.Mortgage.median(),
            color='orange')
plt.axvline(x=loan.Mortgage.mode()[0],
            color='white')
```

Out[45]: <matplotlib.lines.Line2D at 0x7feb791f96d0>



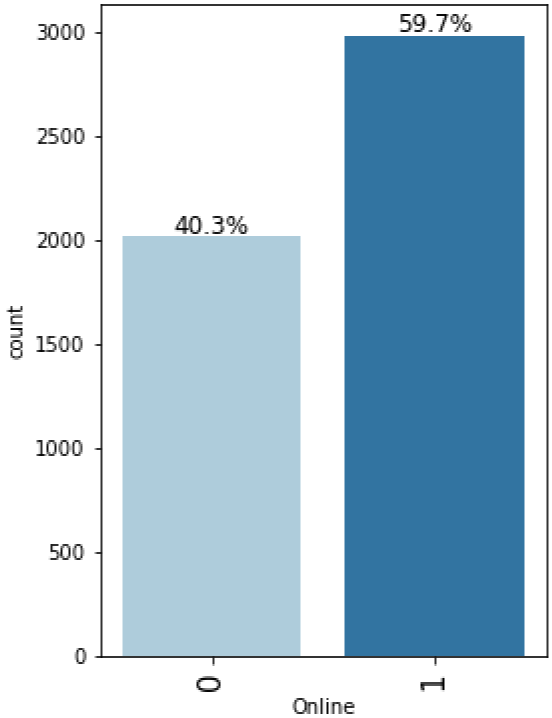
Observations on CreditCard

In [52]:
```
labeled_barplot(loan, "CreditCard", perc=True)
```



Observations on Online

In [53]:
```
labeled_barplot(loan, "Online", perc=True)
```

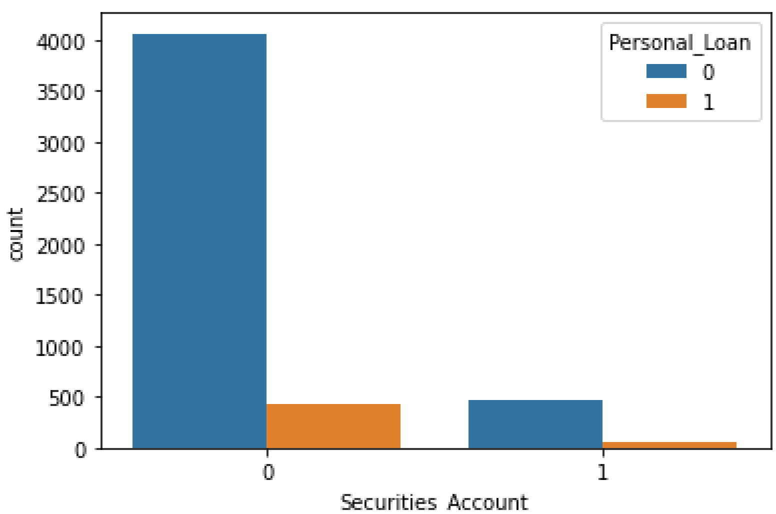


Observations: Income, CCAVg and Mortgage variables are right skewed so we have to take care of these

Bivariate Analysis

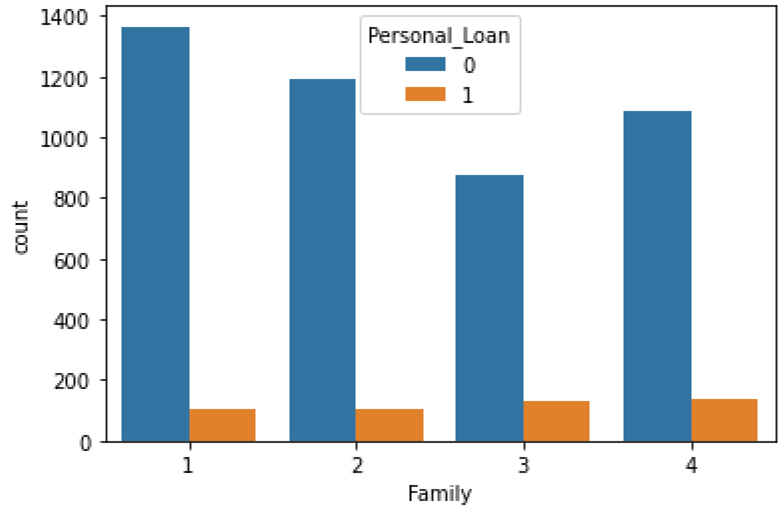
In [67]:
```
sns.countplot(x="Securities_Account",hue="Personal_Loan",data=loan)
```

Out[67]: <AxesSubplot:xlabel='Securities_Account', ylabel='count'>



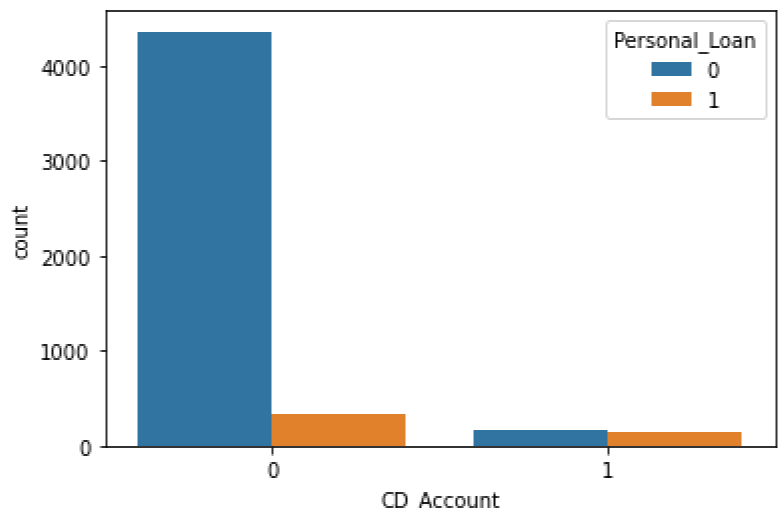
In [66]:
```
sns.countplot(x='Family',hue='Personal_Loan',data=loan)
```

Out[66]: <AxesSubplot:xlabel='Family', ylabel='count'>



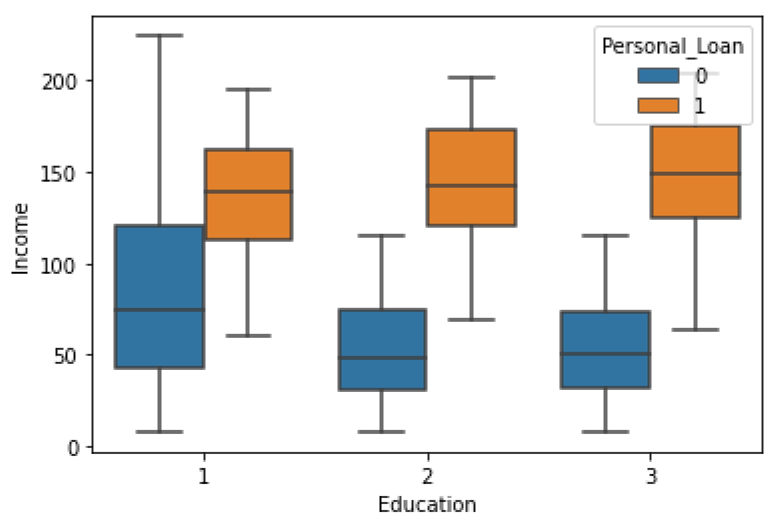
In [68]:
```
sns.countplot(x='CD_Account',hue='Personal_Loan',data=loan)
```

Out[68]: <AxesSubplot:xlabel='CD_Account', ylabel='count'>



In [76]:
```
sns.boxplot(x='Education',y='Income',hue='Personal_Loan',data=loan)
```

Out[76]: <AxesSubplot:xlabel='Education', ylabel='Income'>

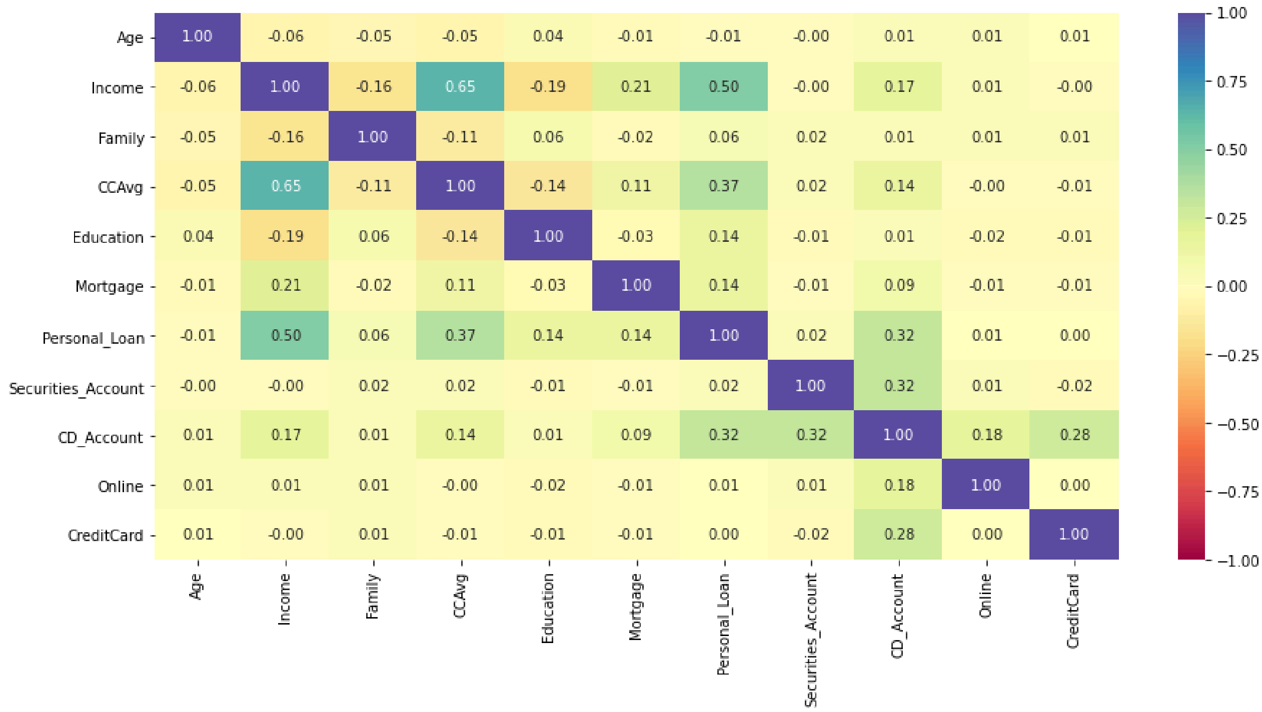


Observations:

- It seems the customers whose education level is 1 is having more income. However customers who has taken the personal loan have the same income levels
- Majority of customers who does not have loan have securities account
- Family size does not have any impact in personal loan. But it seems families with size of 3 are more likely to take loan.
- Customers who does not have CD account , does not have loan as well. This seems to be majority. But almost all customers who has CD account has loan as well

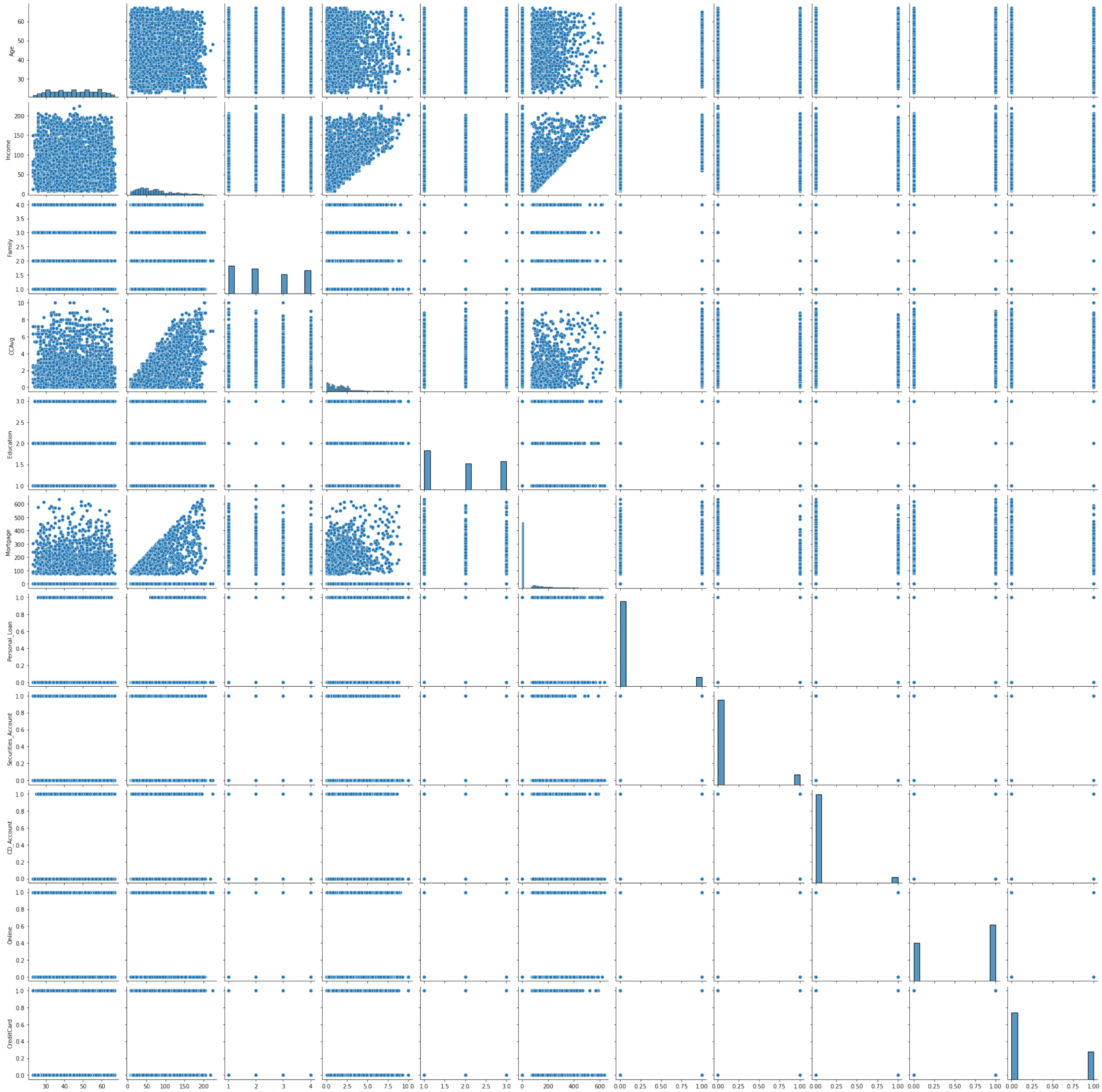
In [78]:

```
# Heatmap

plt.figure(figsize=(15, 7))
sns.heatmap(
    loan.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



In [80]:

```
# Pairplot

sns.pairplot(data=loan)
plt.show()
```



Data Pre-processing

In [218…

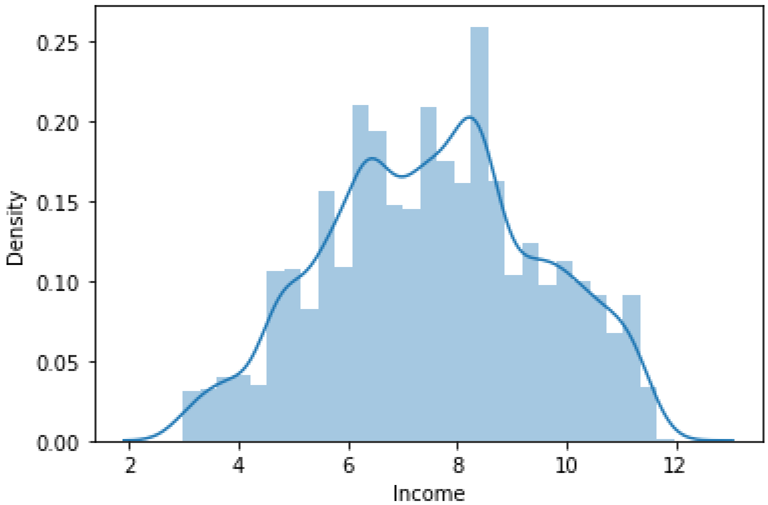
```
X = loan.drop(["Personal_Loan"], axis=1)
y = loan["Personal_Loan"]
```

In [219…

```
# Applying the Yeo Johnson method of Transformation on the Income variable.
pt = PowerTransformer(method='yeo-johnson',standardize=False)
pt.fit(X['Income'].values.reshape(-1,1))
temp = pt.transform(X['Income'].values.reshape(-1,1))
X['Income'] = pd.Series(temp.flatten())
```

In [133…

```
# Distplot to show transformed Income variable
sns.distplot(X['Income'])
plt.show()
```

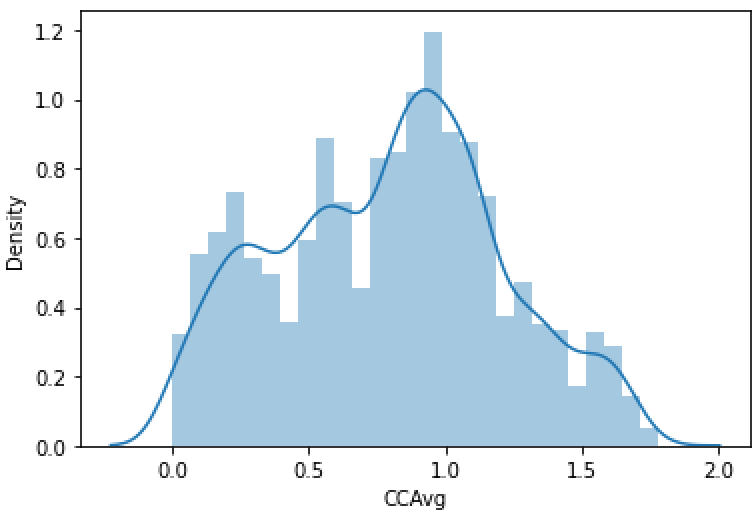


In [220…

```
# Applying the Yeo Johnson method of Transformation on the CCAvg variable.
pt = PowerTransformer(method='yeo-johnson',standardize=False)
pt.fit(X['CCAvg'].values.reshape(-1,1))
temp = pt.transform(X['CCAvg'].values.reshape(-1,1))
X['CCAvg'] = pd.Series(temp.flatten())
```

In [135…

```
# Distplot to show transformed CCAvg variable
sns.distplot(X['CCAvg'])
plt.show()
```



In [221…

```
# Binning on Mortgage variable.
X['Mortgage_val'] = pd.cut(X['Mortgage'],
                              bins=[0,100,200,300,400,500,600,700],
                              labels= [0,1,2,3,4,5,6],
                              include_lowest =True)
X.drop('Mortgage', axis = 1, inplace= True)
```

In [222…

```
# To display top 5 rows
X.head()
```

Out[222…

	Age	Experience	Income	Family	CCAvg	Education	Securities_Account	CD_Account	Onl
0	25	1	6.827583	4	0.845160	1	1	0	
1	45	19	5.876952	3	0.814478	1	1	0	
2	39	15	3.504287	1	0.633777	1	0	0	
3	35	9	8.983393	1	1.107427	2	0	0	
4	35	8	6.597314	4	0.633777	2	0	0	

Model Building - Approach

Split Data

In [266…

```
# Split Data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.4, random_sta
```

In [260…

```
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 = 3000
Number of rows in test data = 2000
```

In [224…

```
# To display top 5 rows
X_train.head()
```

Out[224…

	Age	Experience	Income	Family	CCAvg	Education	Securities_Account	CD_Account
4522	31	5	5.492854	1	0.253539	1	0	0
2851	61	36	8.302424	3	0.902279	2	0	0
2313	58	32	7.097040	3	0.253539	2	0	0
982	58	33	6.991517	3	0.384645	2	0	0
1164	41	17	8.779396	3	1.285926	2	1	1

In [267…

```
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    0.904333
1    0.095667
Name: Personal_Loan, dtype: float64
Percentage of classes in test set:
0    0.9035
1    0.0965
Name: Personal_Loan, dtype: float64
```

Building Logistic Regression Model

In [268…

```
model = LogisticRegression(random_state = 0)
```

In [269…

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

Out[269…

```
LogisticRegression(random_state=0)
```

In [270…

```
# defining a function to compute different metrics to check performance of a cla
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performa

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

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

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

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

    return df_perf
```

In [271…

```
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 [272…

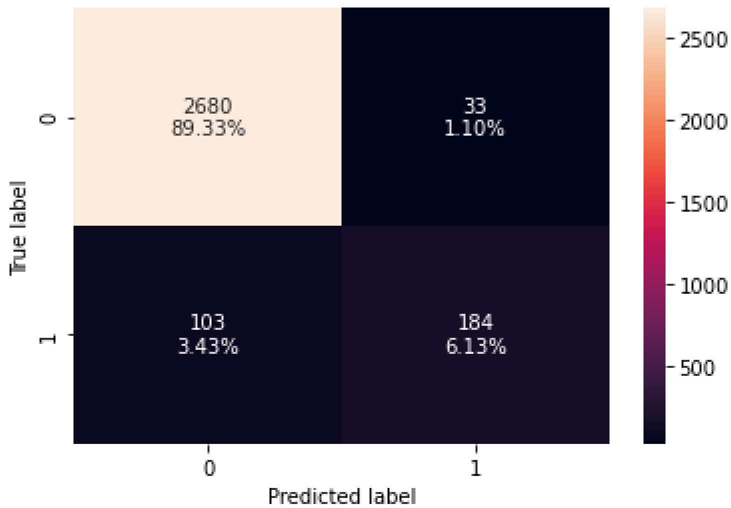
```
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, y_train
)
decision_tree_perf_train
```

Out[272…

	Accuracy	Recall	Precision	F1
0	0.954667	0.641115	0.847926	0.730159

In [273…

```
confusion_matrix_sklearn(model, X_train, y_train)
```



In [274…

```
decision_tree_perf_test = model_performance_classification_sklearn(
    model, X_test, y_test
)
decision_tree_perf_test
```

Out[274…

	Accuracy	Recall	Precision	F1
0	0.9475	0.601036	0.805556	0.688427

Observation: For Logistic Regression we got 94% accuracy for test data. The F1 score is 0.68. Now lets compare that values with other models.

Building Decision Tree Model

- Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

In [235…

```
model = DecisionTreeClassifier(random_state=0, max_depth=8)
model.fit(X_train, y_train)
```

Out[235…

```
DecisionTreeClassifier(max_depth=8, random_state=0)
```

Checking model performance on training set

In [236…

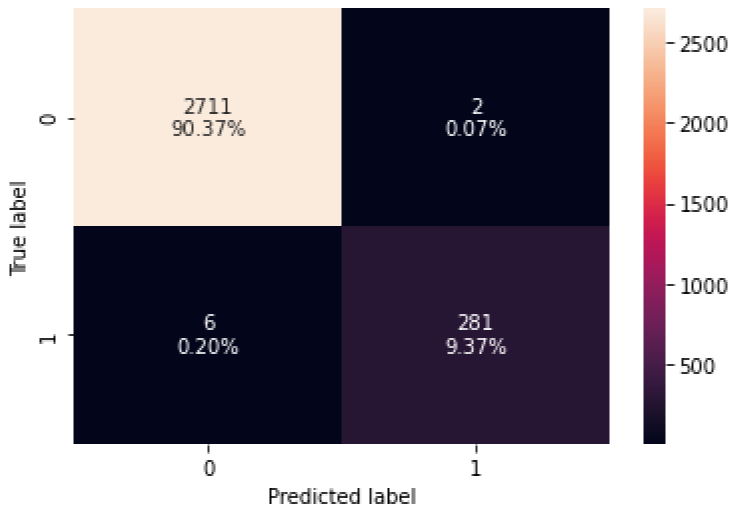
```
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, y_train
)
decision_tree_perf_train
```

Out[236…

	Accuracy	Recall	Precision	F1
0	0.997667	0.979094	0.996454	0.987698

In [159…

```
confusion_matrix_sklearn(model, X_train, y_train)
```



Checking model performance on test set

In [237…

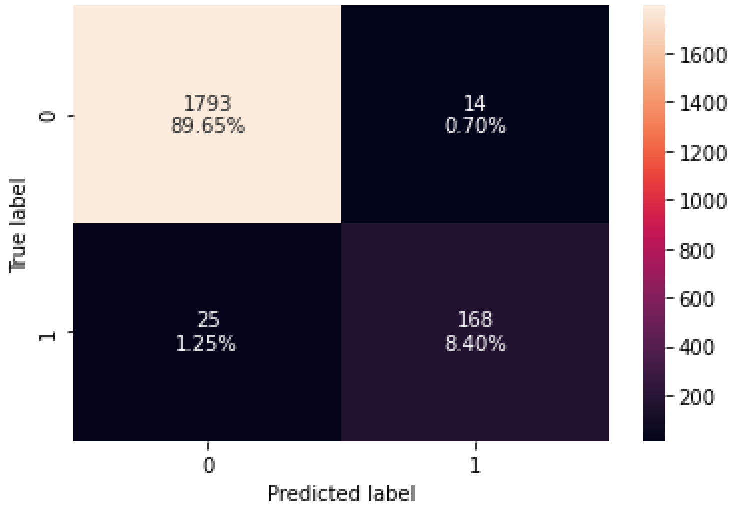
```
decision_tree_perf_test = model_performance_classification_sklearn(
    model, X_test, y_test
)
decision_tree_perf_test
```

Out[237…

	Accuracy	Recall	Precision	F1
0	0.98	0.870466	0.918033	0.893617

In [163…

```
confusion_matrix_sklearn(model, X_test, y_test)
```



Observation: Model is giving good and generalized results on training and test set.

Visualizing the Decision Tree

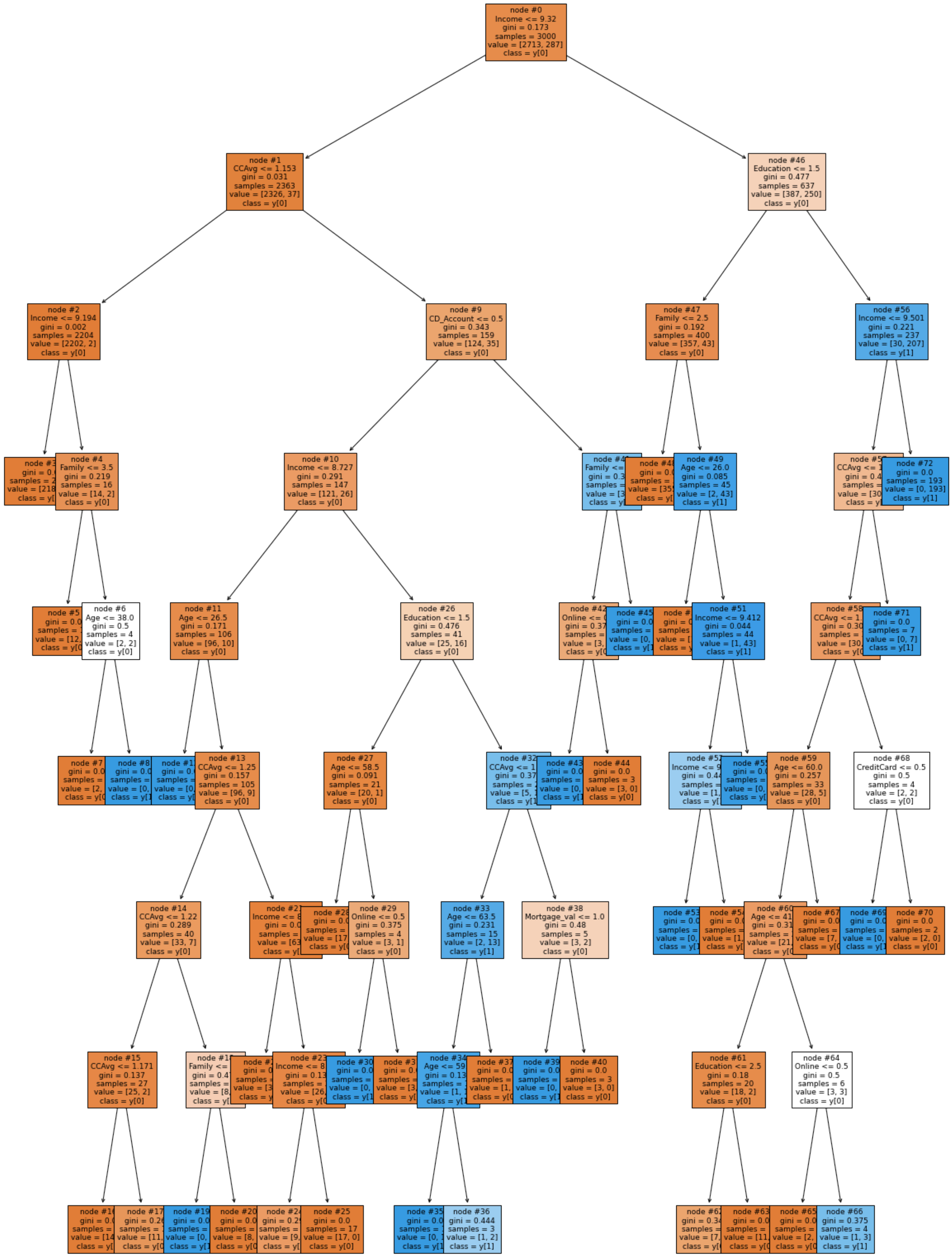
In [238…

```
column_names = list(X.columns)
feature_names = column_names
print(feature_names)
```

```
['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Education', 'Securities_Acco
unt', 'CD_Account', 'Online', 'CreditCard', 'Mortgage_val']
```

In [166…

```
plt.figure(figsize=(20, 30))

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



In [167…

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

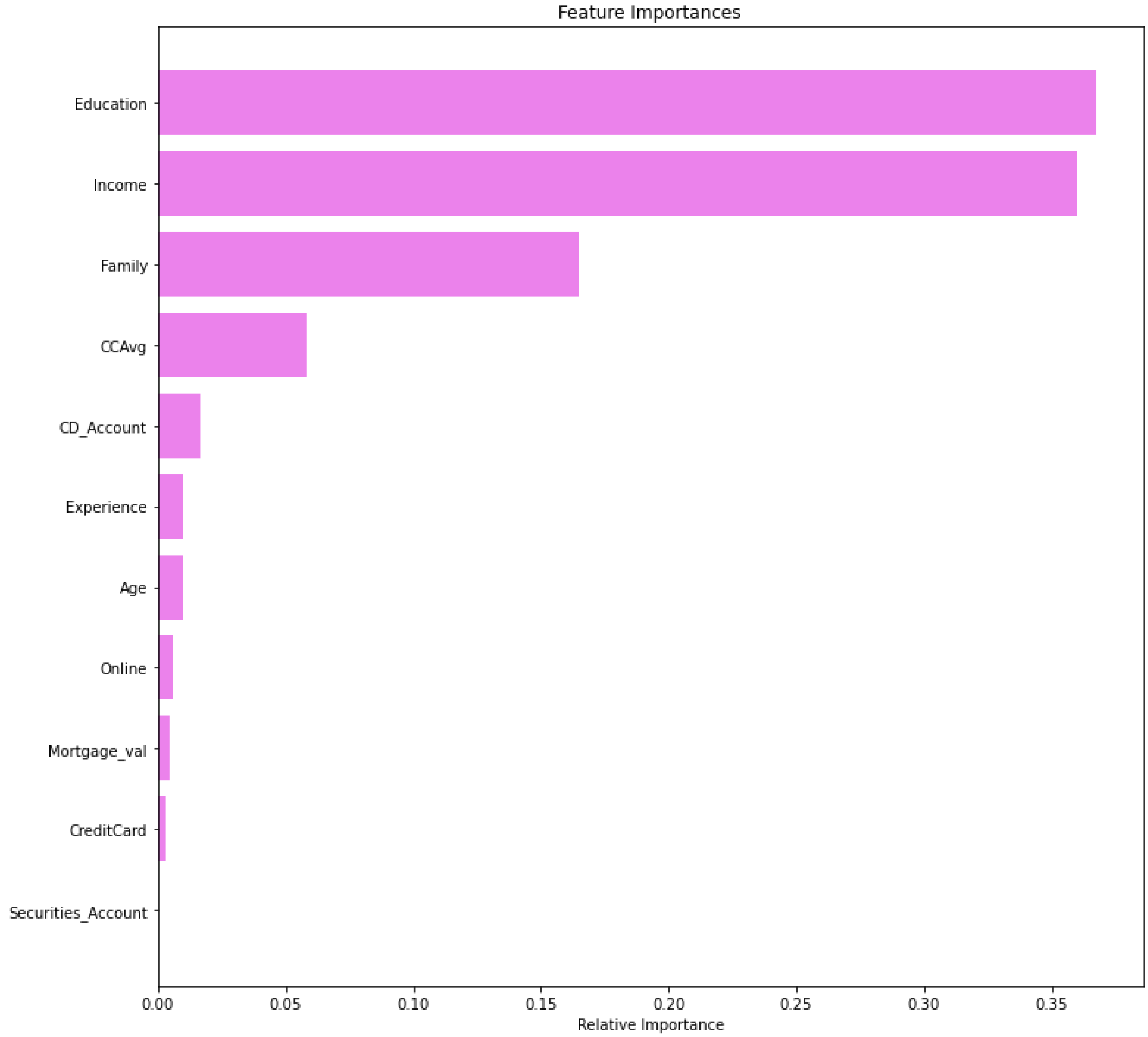
```
print(tree.export_text(model, feature_names=feature_names, show_weights=True))
```

```
|--- Income <= 9.32
|   |--- CCAvg <= 1.15
|   |   |--- Income <= 9.19
|   |   |   |--- weights: [2188.00, 0.00] class: 0
|   |   |--- Income >  9.19
|   |   |   |--- Family <= 3.50
|   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |--- Family >  3.50
|   |   |   |   |--- Age <= 38.00
|   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |--- Age >  38.00
|   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |--- CCAvg >  1.15
|   |   |--- CD_Account <= 0.50
|   |   |   |--- Income <= 8.73
|   |   |   |   |--- Age <= 26.50
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- Age >  26.50
|   |   |   |   |   |--- CCAvg <= 1.25
|   |   |   |   |   |   |--- CCAvg <= 1.22
|   |   |   |   |   |   |   |--- CCAvg <= 1.17
|   |   |   |   |   |   |   |   |--- weights: [14.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- CCAvg >  1.17
|   |   |   |   |   |   |   |   |--- weights: [11.00, 2.00] class: 0
|   |   |   |   |   |   |--- CCAvg >  1.22
|   |   |   |   |   |   |   |--- Family <= 3.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |--- Family >  3.00
|   |   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  1.25
|   |   |   |   |   |   |--- Income <= 8.32
|   |   |   |   |   |   |   |--- weights: [37.00, 0.00] class: 0
|   |   |   |   |   |   |--- Income >  8.32
|   |   |   |   |   |   |   |--- Income <= 8.40
|   |   |   |   |   |   |   |   |--- weights: [9.00, 2.00] class: 0
|   |   |   |   |   |   |   |--- Income >  8.40
|   |   |   |   |   |   |   |   |--- weights: [17.00, 0.00] class: 0
|   |   |   |--- Income >  8.73
|   |   |   |   |--- Education <= 1.50
|   |   |   |   |   |--- Age <= 58.50
|   |   |   |   |   |   |--- weights: [17.00, 0.00] class: 0
|   |   |   |   |   |--- Age >  58.50
|   |   |   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- Online >  0.50
|   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |--- Education >  1.50
|   |   |   |   |   |--- CCAvg <= 1.35
|   |   |   |   |   |   |--- Age <= 63.50
|   |   |   |   |   |   |   |--- Age <= 59.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 11.00] class: 1
|   |   |   |   |   |   |   |--- Age >  59.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 2.00] class: 1
|   |   |   |   |   |   |--- Age >  63.50
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- CCAvg >  1.35
|   |   |   |   |   |   |--- Mortgage_val <= 1.00
|   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- Mortgage_val >  1.00
|   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |--- CD_Account >  0.50
|   |   |   |--- Family <= 1.50
|   |   |   |   |--- Online <= 0.50
```

```
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- Online >  0.50
|   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |--- Family >  1.50
|   |   |   |   |--- weights: [0.00, 8.00] class: 1
|--- Income >  9.32
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [355.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- Age <= 26.00
|   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- Age >  26.00
|   |   |   |   |--- Income <= 9.41
|   |   |   |   |   |--- Income <= 9.37
|   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |--- Income >  9.37
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- Income >  9.41
|   |   |   |   |   |--- weights: [0.00, 41.00] class: 1
|   |--- Education >  1.50
|   |   |--- Income <= 9.50
|   |   |   |--- CCAvg <= 1.24
|   |   |   |   |--- CCAvg <= 1.13
|   |   |   |   |   |--- Age <= 60.00
|   |   |   |   |   |   |--- Age <= 41.50
|   |   |   |   |   |   |   |--- Education <= 2.50
|   |   |   |   |   |   |   |   |--- weights: [7.00, 2.00] class: 0
|   |   |   |   |   |   |   |--- Education >  2.50
|   |   |   |   |   |   |   |   |--- weights: [11.00, 0.00] class: 0
|   |   |   |   |   |   |--- Age >  41.50
|   |   |   |   |   |   |   |--- Online <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- Online >  0.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 3.00] class: 1
|   |   |   |   |   |--- Age >  60.00
|   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |--- CCAvg >  1.13
|   |   |   |   |   |--- CreditCard <= 0.50
|   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |--- CreditCard >  0.50
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |--- CCAvg >  1.24
|   |   |   |   |--- weights: [0.00, 7.00] class: 1
|   |   |--- Income >  9.50
|   |   |   |--- weights: [0.00, 193.00] class: 1
```

In [239…

```
importances = 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()
```



Obseravtion: Education, Income and Family are the top 3 important features.

Using GridSearch for Hyperparameter tuning of our tree model

- Let's see if we can improve our model performance even more.

In [240…

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

# Grid of parameters to choose from

parameters = {
    "max_depth": [np.arange(2, 50, 5), None],
    "criterion": ["entropy", "gini"],
    "splitter": ["best", "random"],
    "min_impurity_decrease": [0.000001, 0.00001, 0.0001],
}

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

# Run the grid search
```

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

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

# Fit the best algorithm to the data.
estimator.fit(X_train, y_train)
```

Out[240…

```
DecisionTreeClassifier(min_impurity_decrease=1e-06, random_state=1)
```

Checking performance on training set

In [241…

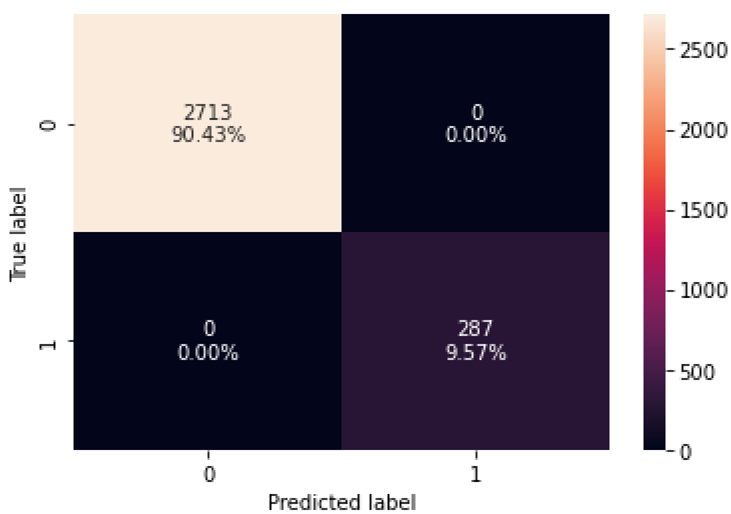
```
decision_tree_tune_perf_train = model_performance_classification_sklearn(
    estimator, X_train, y_train
)
decision_tree_tune_perf_train
```

Out[241…

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

In [242…

```
confusion_matrix_sklearn(estimator, X_train, y_train)
```



In [243…

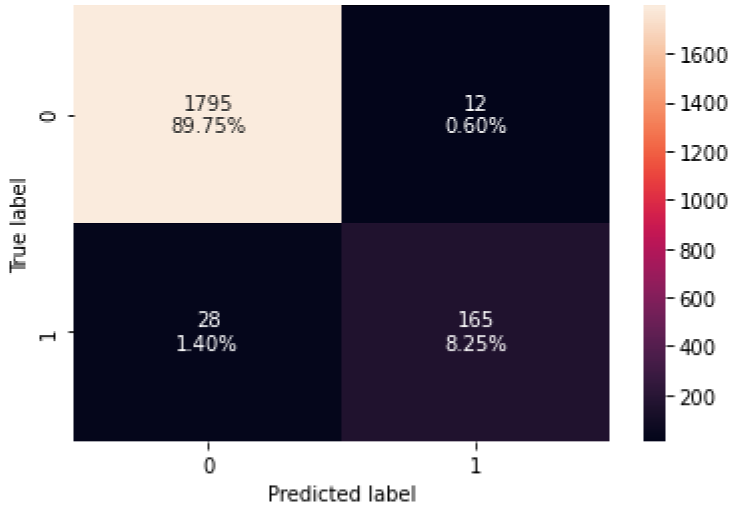
```
decision_tree_tune_perf_test = model_performance_classification_sklearn(
    estimator, X_test, y_test
)

decision_tree_tune_perf_test
```

Out[243…

	Accuracy	Recall	Precision	F1
0	0.9795	0.860104	0.922222	0.89008

In [176…

```
confusion_matrix_sklearn(estimator, X_test, y_test)
```

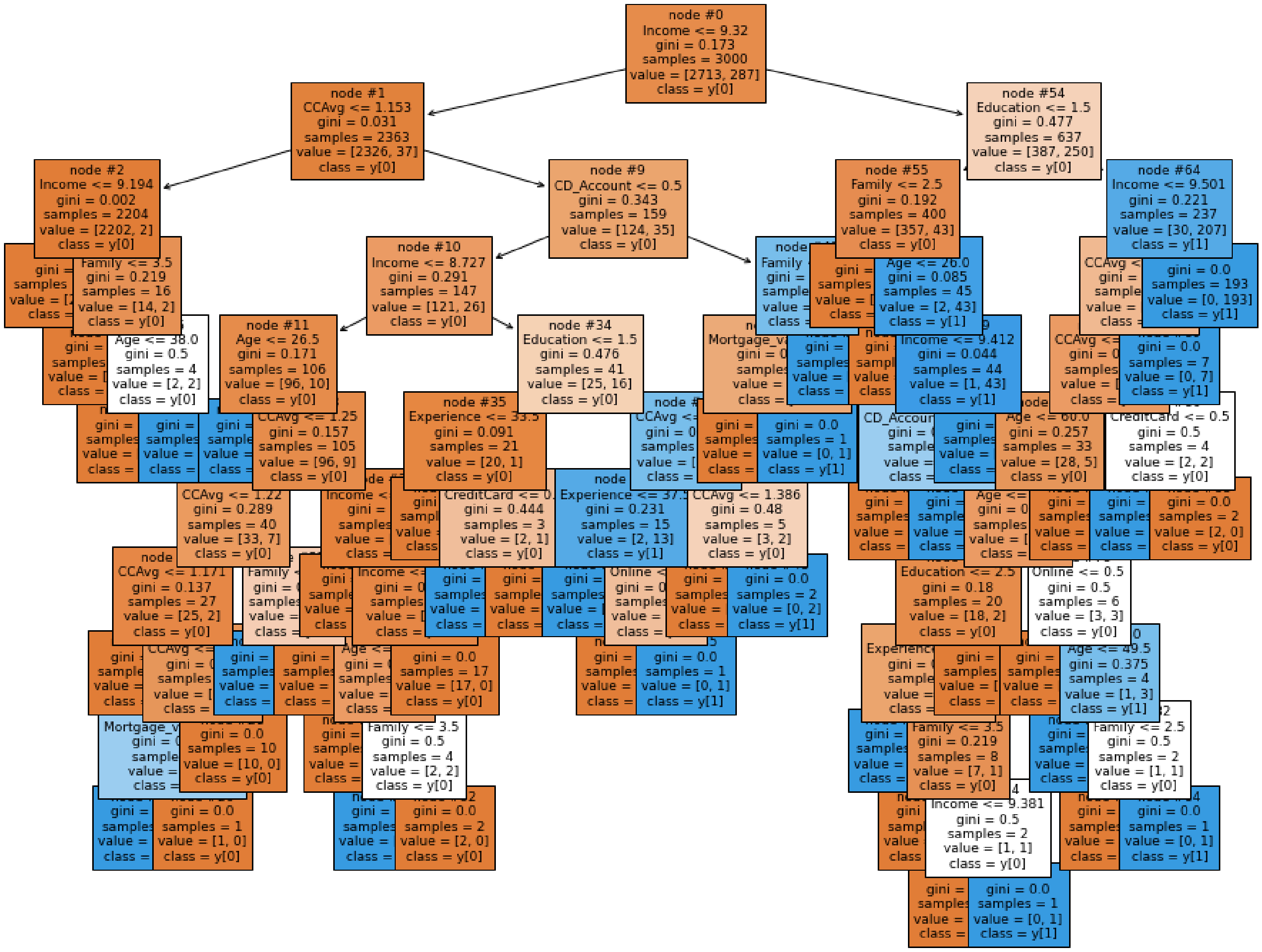


Observation:

- The Recall has improved on the training set as compared to the initial model.
- After hyperparameter tuning the model has performance has remained same and the model has become simpler.

In [244…

```
plt.figure(figsize=(15, 12))

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



Observation:

- We are getting a simplified tree after pre-pruning.

Cost Complexity Pruning

In [245…

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

In [180…

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

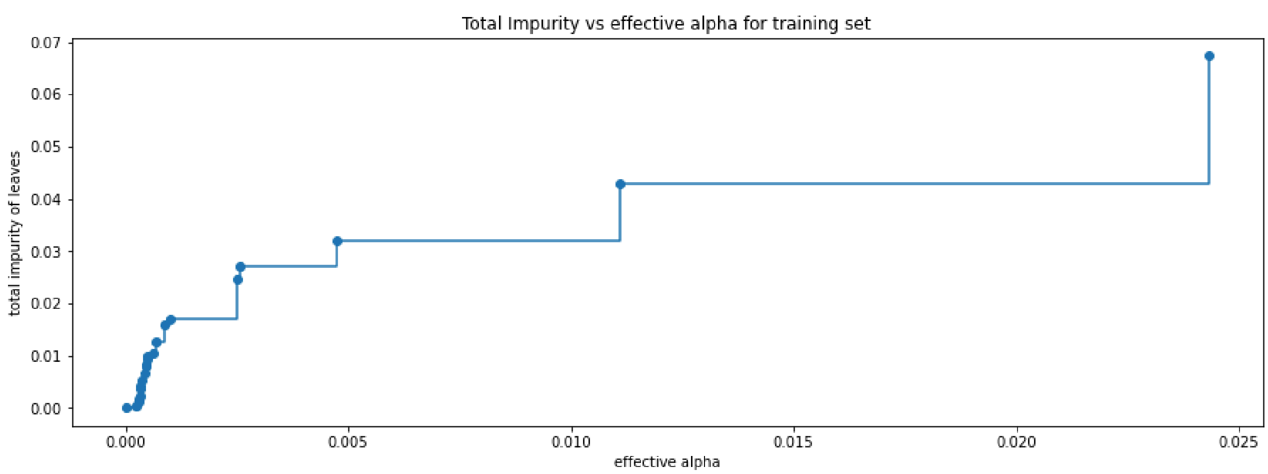
Out[180…

	ccp_alphas	impurities
0	0.000000	0.000000
1	0.000250	0.000500
2	0.000292	0.001083
3	0.000308	0.001700
4	0.000310	0.002319
5	0.000317	0.002954
6	0.000323	0.004246
7	0.000326	0.004898

	ccp_alphas	impurities
8	0.000376	0.006026
9	0.000412	0.007261
10	0.000444	0.008593
11	0.000478	0.009548
12	0.000500	0.010048
13	0.000537	0.010585
14	0.000623	0.011207
15	0.000672	0.012552
16	0.000878	0.016063
17	0.001000	0.017063
18	0.002508	0.024587
19	0.002580	0.027167
20	0.004751	0.031918
21	0.011105	0.043023
22	0.024311	0.067334
23	0.052848	0.173029

In [246…

```
fig, ax = plt.subplots(figsize=(15, 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()
```



In [247…

```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
```

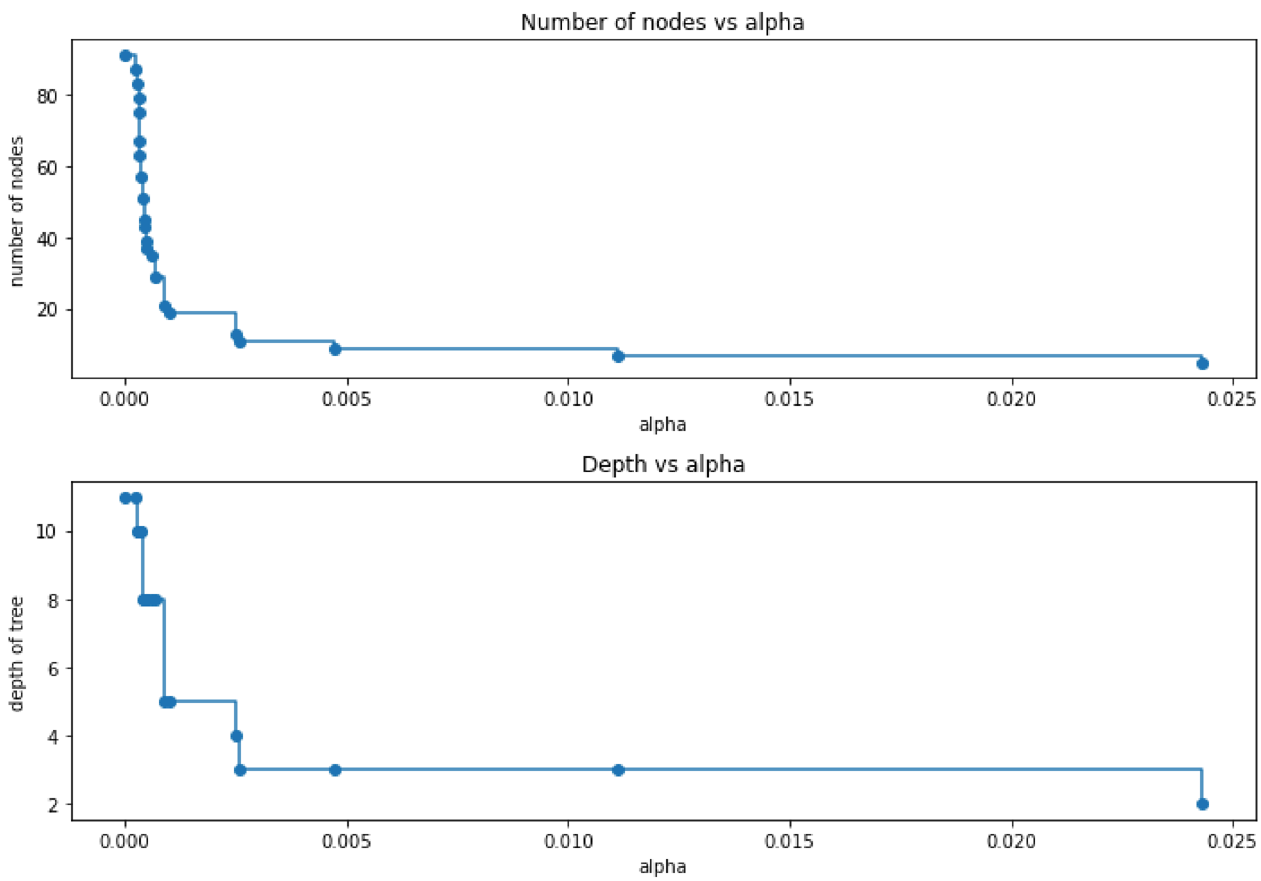
```
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.05284766110239135
```

- For the remainder, we remove the last element in clfs and ccp_alphas, because it is the trivial tree with only one node. Here we show that the number of nodes and tree depth decreases as alpha increases.

In [248…

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

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



Recall vs alpha for training and testing sets

In [249…

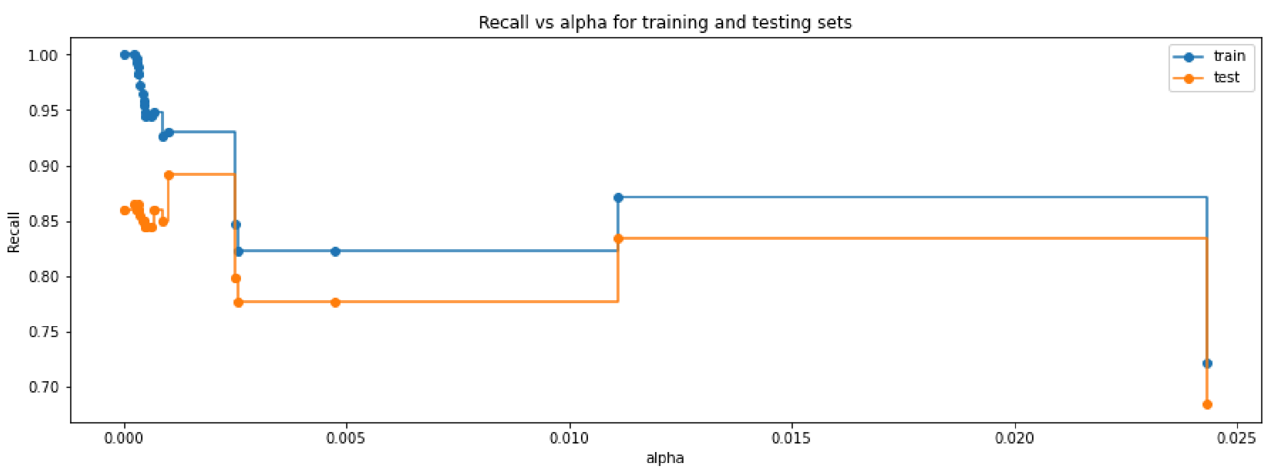
```
recall_train = []
for clf in clfs:
    pred_train = clf.predict(X_train)
    values_train = recall_score(y_train, pred_train)
    recall_train.append(values_train)
```

In [250…

```
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 [251…

```
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-pc
ax.plot(ccp_alphas, recall_test, marker="o", label="test", drawstyle="steps-post
ax.legend()
plt.show()
```



In [252…

```
# creating the model where we get highest train and test recall
index_best_model = np.argmax(recall_test)
best_model = clfs[index_best_model]
print(best_model)
```

```
DecisionTreeClassifier(ccp_alpha=0.001, random_state=1)
```

Checking model performance on training set

In [253…

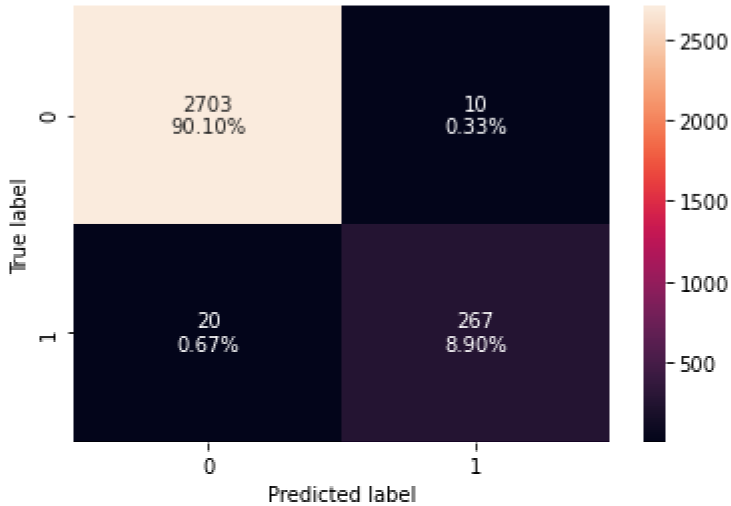
```
decision_tree_postpruned_perf_train = model_performance_classification_sklearn(
    best_model, X_train, y_train
)
decision_tree_postpruned_perf_train
```

Out[253…

	Accuracy	Recall	Precision	F1
0	0.99	0.930314	0.963899	0.946809

In [191…

```
confusion_matrix_sklearn(best_model, X_train, y_train)
```



In [254…

```
decision_tree_postpruned_perf_test = model_performance_classification_sklearn(
    best_model, X_test, y_test
)
decision_tree_postpruned_perf_test
```

Out[254…

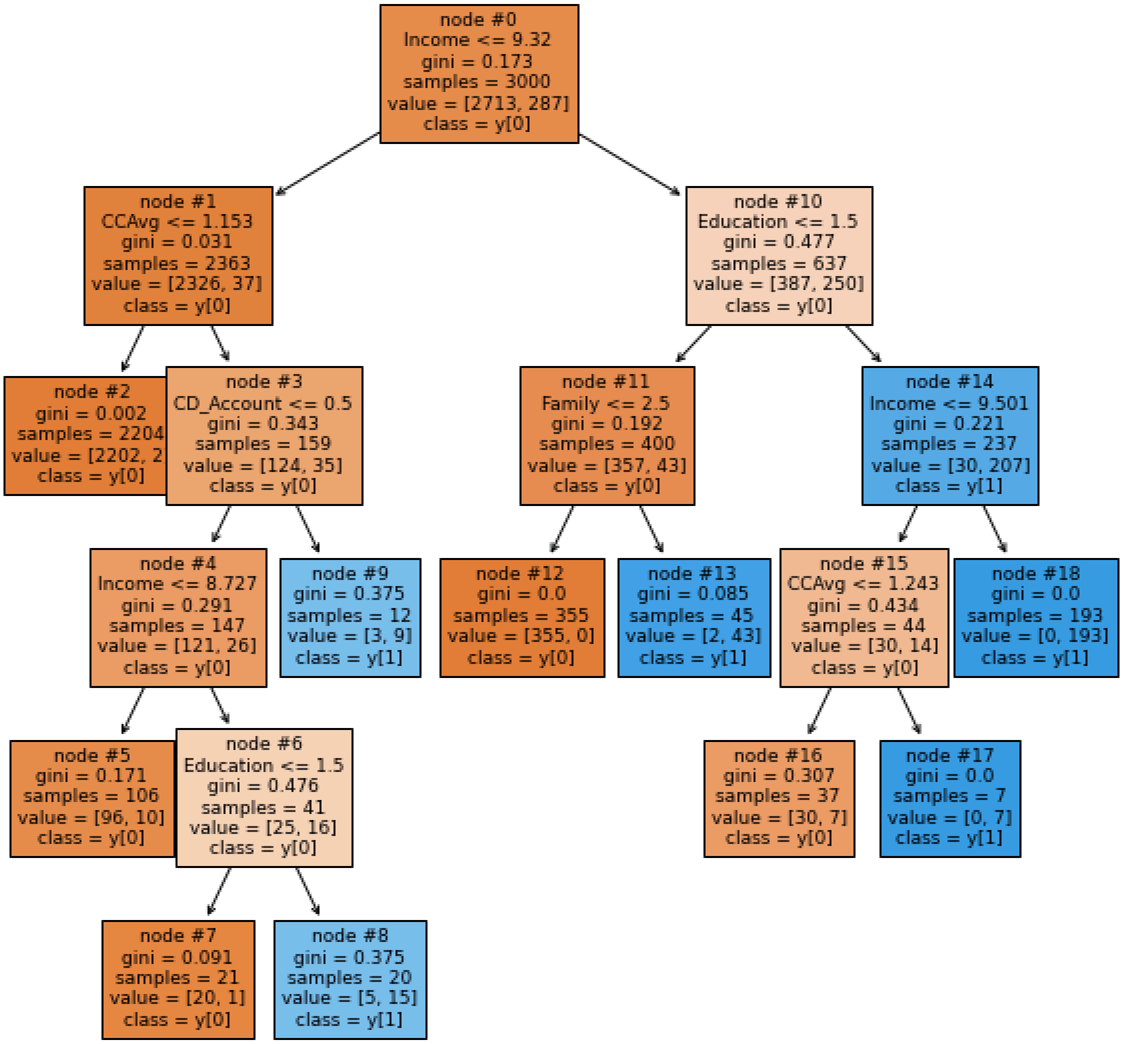
	Accuracy	Recall	Precision	F1
0	0.9825	0.891192	0.924731	0.907652

- With post-pruning we are getting good and generalized model performance on both training and test set.
- The recall has improved further.

Visualizing the Decision Tree

In [193…

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

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



In [194…

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

print(tree.export_text(best_model, feature_names=feature_names, show_weights=Tru
```

```
|--- Income <= 9.32
|   |--- CCAvg <= 1.15
|   |   |--- weights: [2202.00, 2.00] class: 0
|   |--- CCAvg >  1.15
|   |   |--- CD_Account <= 0.50
|   |   |   |--- Income <= 8.73
|   |   |   |   |--- weights: [96.00, 10.00] class: 0
|   |   |   |--- Income >  8.73
|   |   |   |   |--- Education <= 1.50
|   |   |   |   |   |--- weights: [20.00, 1.00] class: 0
|   |   |   |   |--- Education >  1.50
|   |   |   |   |   |--- weights: [5.00, 15.00] class: 1
|   |   |--- CD_Account >  0.50
|   |   |   |--- weights: [3.00, 9.00] class: 1
|--- Income >  9.32
|   |--- Education <= 1.50
|   |   |--- Family <= 2.50
|   |   |   |--- weights: [355.00, 0.00] class: 0
|   |   |--- Family >  2.50
|   |   |   |--- weights: [2.00, 43.00] class: 1
|   |--- Education >  1.50
```

```
|   |   |--- Income <= 9.50
|   |   |   |--- CCAvg <= 1.24
|   |   |   |   |--- weights: [30.00, 7.00] class: 0
|   |   |   |--- CCAvg >  1.24
|   |   |   |   |--- weights: [0.00, 7.00] class: 1
|   |   |--- Income >  9.50
|   |   |   |--- weights: [0.00, 193.00] class: 1
```

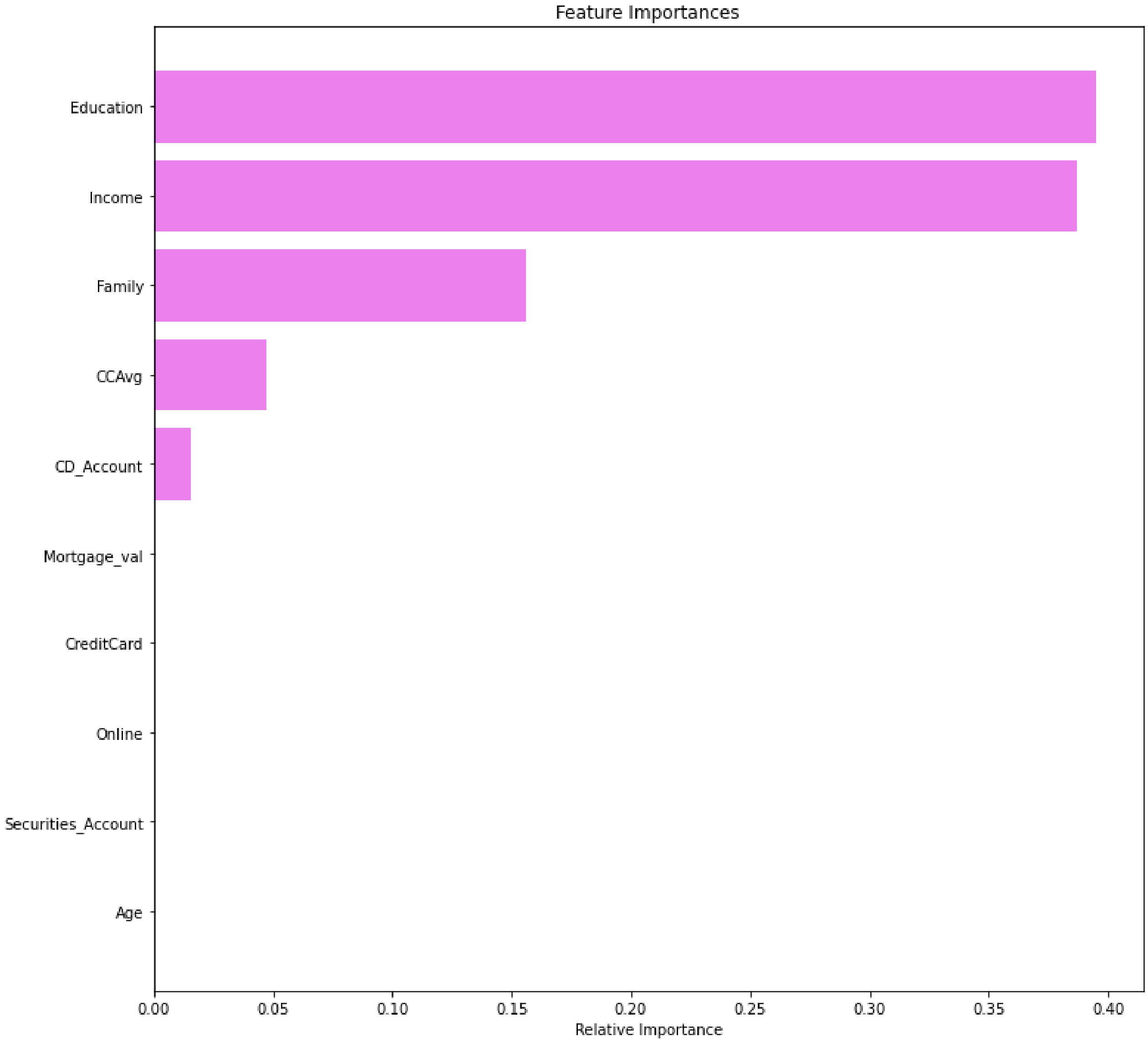
In [195…

```
# importance of features in the tree building ( The importance of a feature is c
# (normalized) total reduction of the 'criterion' brought by that feature. It is

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

```
                        Imp
Education          0.394775
Income             0.386771
Family             0.155873
CCAvg              0.047004
CD_Account         0.015577
Age                0.000000
Securities_Account 0.000000
Online             0.000000
CreditCard         0.000000
Mortgage_val       0.000000
```

In [196…

```
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()
```



- Education, Income, Family and CCAvg remain the most important feature with post-pruning too.

Comparing all the decision tree models

In [255…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        decision_tree_perf_train.T,
        decision_tree_tune_perf_train.T,
        decision_tree_postpruned_perf_train.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree sklearn",
    "Decision Tree (Pre-Pruning)",
    "Decision Tree (Post-Pruning)",
]
```

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

Training performance comparison:

Out[255...

	Decision Tree sklearn	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.997667	1.0	0.990000
Recall	0.979094	1.0	0.930314
Precision	0.996454	1.0	0.963899
F1	0.987698	1.0	0.946809

In [258...

```
# test performance comparison

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

Test set performance comparison:

Out[258...

	Decision Tree sklearn	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.980000	0.979500	0.982500
Recall	0.870466	0.860104	0.891192
Precision	0.918033	0.922222	0.924731
F1	0.893617	0.890080	0.907652

- Decision tree with post-pruning is giving the highest recall on the test set.

Business Insights

- The aim of the universal bank is to convert there liability customers into loan customers. They want to set up a new marketing campaign. Hence, they need information about the connection between the variables given in the data. Two classification algorithms were used in this project. From the implementation, it seems like Decision Tree have the highest accuracy and we can choose that as our final model.
- Recall is more important where "False Negatives" are more costly than "False Positive". The focus in these problems is finding the positive customers. So recall is our evaluation metrics.

- Decision tree with post-pruning model is giving the highest recall.
- Education, Income, Family and CCAvg remain the most important feature
- It seems the customers whose education level is 1 is having more income. However customers who has taken the personal loan have the same income levels
- Majority of customers who does not have loan have securities account
- Family size does not have any impact in personal loan. But it seems families with size of 3 are more likely to take loan. So we need to consider this during the campaign
- Customers who does not have CD account , does not have loan as well. This seems to be majority. But almost all customers who has CD account has loan as well