

✓ Problem Statement

Context

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

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

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

Objective

To predict whether a liability customer will buy personal loans, to understand which customer attributes are most significant in driving purchases, and identify which segment of customers to target more.

NameError X

...



You

Please explain the error:

NameError: name '_____' is



Gemini

Hypothesis

The error NameError: name '_____' is not defined is occurring because the code is attempting to use a variable named '___' which has not been defined in the current scope. It seems like the user was prompted to fill in a value for ccp_alpha represented by the placeholder '___'.

Since the user's goal is to add the correct ccp_alpha value, and a variable ccp_alpha exists in their environment with a float value, it's highly likely they were expected to use this value when defining estimator_2.

Suggested Changes



```
estimator_2 = DecisionTreeClassifier(
    ccp_alpha=ccp_alpha, c1
```

Data Dictionary

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

✓ **Please read the instructions carefully before starting the project.**

```
)  
estimator_2.fit(X_train, y_
```

[Use code with caution](#)

► Sources

Rate this answer



This is a commented Jupyter IPython Notebook file in which all the instructions and tasks to be performed are mentioned.

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

✓ Importing necessary libraries

```
# Installing the libraries with the specified versions
!pip install numpy==1.25.2 pandas==1.5.3 matplotlib==3.5.2
```

Note:

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

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

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

# Library to split data
from sklearn.model_selection import train_test_split

# To build model for prediction
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# to tune different models
from sklearn.model_selection import GridSearchCV

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

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

✓ Loading the dataset

```
# uncomment the following lines if Google Colab is
from google.colab import drive
drive.mount('/content/drive')
```

 Drive already mounted at /content/drive; to at

```
Loan = pd.read_csv('/content/drive/MyDrive/AIML Co
```

```
# copying data to another variable to avoid any cl
data = Loan.copy()
```

✓ Data Overview

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

```
data.head() ## Complete the code to view top 5 rows
```



	ID	Age	Experience	Income	ZIPCode	Family
0	1	25	1	49	91107	4
1	2	45	19	34	90089	3
2	3	39	15	11	94720	1
3	4	35	9	100	94112	1
4	5	35	8	45	91330	4

Next
steps:

[code](#) [data](#)

☒ [recommended](#)

[interactiv](#)

```
data.tail() ## Complete the code to view last 5 rows
```



	ID	Age	Experience	Income	ZIPCode	Family
4995	4996	29	3	40	92697	
4996	4997	30	4	15	92037	
4997	4998	63	39	24	93023	
4998	4999	65	40	49	90034	
4999	5000	28	4	83	92612	

✓ Understand the shape of the dataset.

```
data.shape ## Complete the code to get the shape of
```

```
(5000, 14)
```

✓ Check the data types of the columns for the dataset

```
data.info() ## Complete the code to view the data
```

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

✓ Checking the Statistical Summary

```
data.describe().T ## Complete the code to print 1
```



	count	mean	st
ID	5000.0	2500.500000	1443.52000
Age	5000.0	45.338400	11.46316
Experience	5000.0	20.104600	11.46795
Income	5000.0	73.774200	46.03372
ZIPCode	5000.0	93169.257000	1759.45508
Family	5000.0	2.396400	1.14766
CCAvg	5000.0	1.937938	1.74765
Education	5000.0	1.881000	0.83986
Mortgage	5000.0	56.498800	101.71380
Personal_Loan	5000.0	0.096000	0.29462
Securities_Account	5000.0	0.104400	0.30580
CD_Account	5000.0	0.060400	0.23825
Online	5000.0	0.596800	0.49058
CreditCard	5000.0	0.294000	0.45563

```
data.duplicated().sum() ## Complete the code to c
```



```
0
```



```
data.isnull().sum() ## Complete the code to check
```



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

dtype: int64

✓ Dropping columns

```
data = data.drop(['ID'], axis=1) ## Complete the
```

✓ Data Preprocessing

✓ Checking for Anomalous Values

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

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

```
# checking for experience <0
```

```
data[data["Experience"] < 0]["Experience"].unique()
```

```
↵ array([-1, -2, -3])
```

```
# Correcting the experience values
```

```
data["Experience"].replace(-1, 1, inplace=True)
```

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

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

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

```
↵ array([1, 2, 3])
```

▼ Feature Engineering

```
# checking the number of uniques in the zip code
```

```
data["ZIPCode"].nunique()
```

```
↵ 467
```

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

data["ZIPCode"] = data["ZIPCode"].astype("category")
⇒ Number of unique values if we take first two c

## Converting the data type of categorical features
cat_cols = [
    "Education",
    "Personal_Loan",
    "Securities_Account",
    "CD_Account",
    "Online",
    "CreditCard",
    "ZIPCode",
]
data[cat_cols] = data[cat_cols].astype("category")
```

```
data.info() #checking the data to see if data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 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   categ  
 4   Family                 5000 non-null   int64  
 5   CCAvg                  5000 non-null   float  
 6   Education              5000 non-null   categ  
 7   Mortgage               5000 non-null   int64  
 8   Personal_Loan          5000 non-null   categ  
 9   Securities_Account     5000 non-null   categ  
10  CD_Account             5000 non-null   categ  
11  Online                 5000 non-null   categ  
12  CreditCard             5000 non-null   categ  
dtypes: category(7), float64(1), int64(5)
memory usage: 269.8 KB
```

✓ Exploratory Data Analysis (EDA)

✓ Univariate Analysis

```
def histogram_boxplot(data, feature, figsize=(12,
      """
      Boxplot and histogram combined

      data: dataframe
      feature: dataframe column
      figsize: size of figure (default (12,7))
      kde: whether to show the density curve (default False)
      bins: number of bins for histogram (default None)
      """
      f2, (ax_box2, ax_hist2) = plt.subplots(
          nrows=2, # Number of rows of the subplot
          sharex=True, # x-axis will be shared among subplots
          gridspec_kw={"height_ratios": (0.25, 0.75)},
          figsize=figsize,
      ) # creating the 2 subplots
      sns.boxplot(
          data=data, x=feature, ax=ax_box2, showmeans=True
      ) # boxplot will be created and a star will be placed at the mean
      sns.histplot(
          data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins
      ) if bins else sns.histplot(
          data=data, x=feature, kde=kde, ax=ax_hist2, bins=10
      ) # For histogram
      ax_hist2.axvline(
          data[feature].mean(), color="green", linestyle='solid'
      ) # Add mean to the histogram
      ax_hist2.axvline(
          data[feature].median(), color="black", linestyle='solid'
      ) # Add median to the histogram
```

function to create labeled barplots

```
def labeled_barplot(data, feature, perc=False, n=10):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of counts
    n: displays the top n category levels (default 10)
    """
```

```

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

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

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

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

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

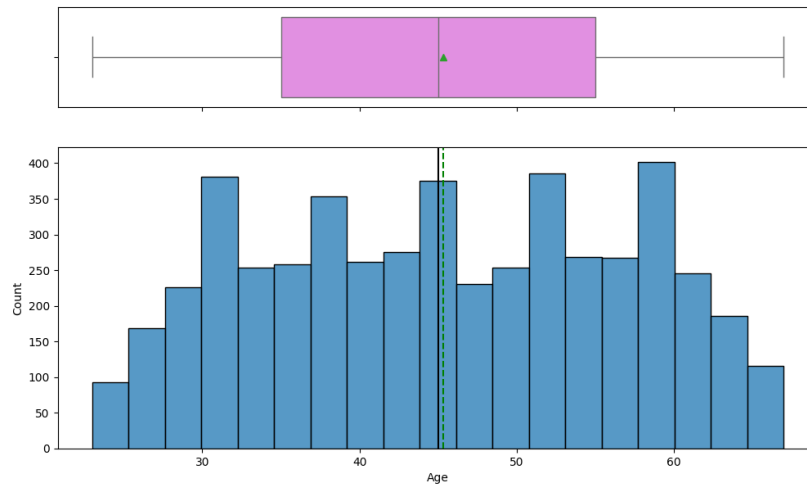
plt.show() # show the plot

```

✓ Observations on Age

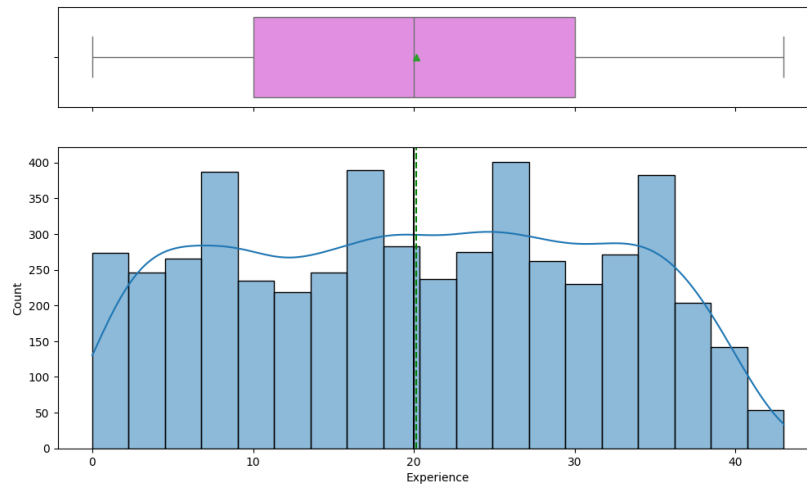
Start coding or [generate](#) with AI.

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



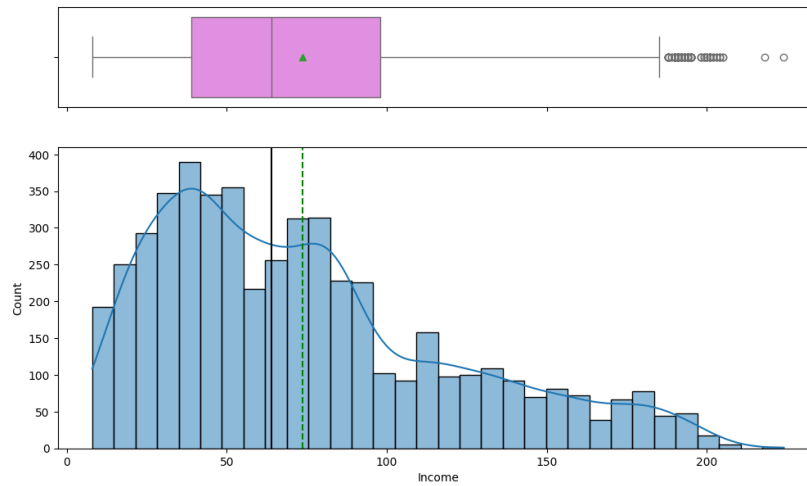
✓ Observations on Experience

```
histogram_boxplot(data, "Experience", kde=True) ##
```



✓ Observations on Income

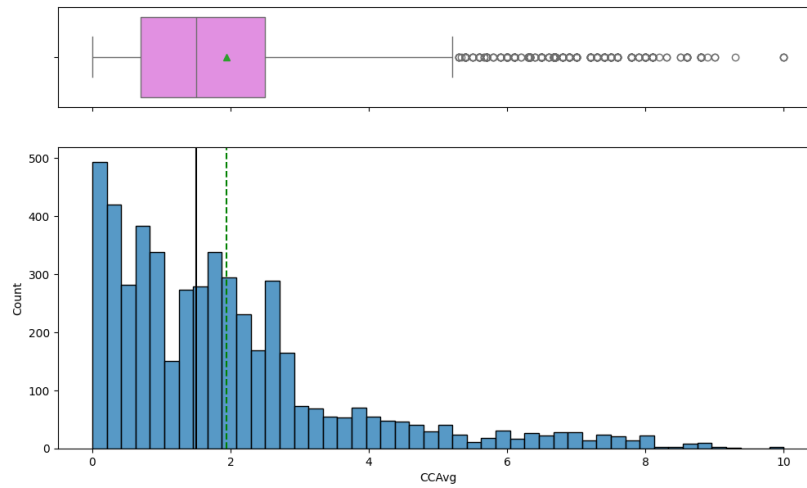

```
histogram_boxplot(data, 'Income', kde=True) ## Cc
```



Start coding or [generate](#) with AI.

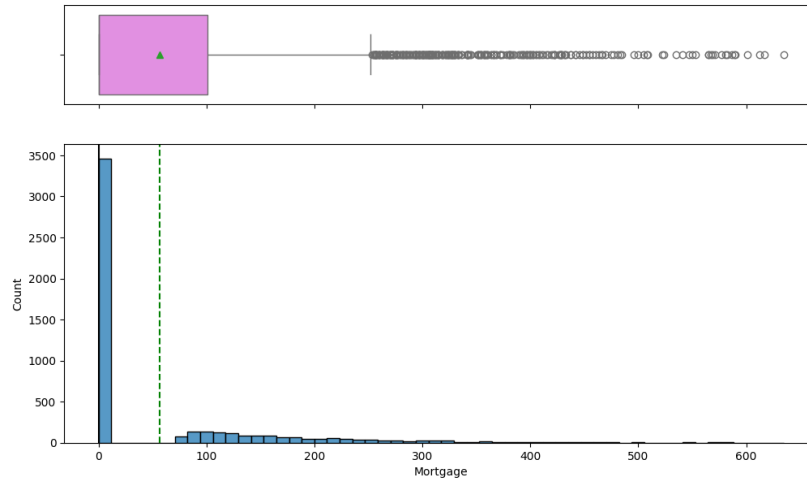
✓ Observations on CCAvg

```
histogram_boxplot(data, 'CCAvg') ## Complete the c
```



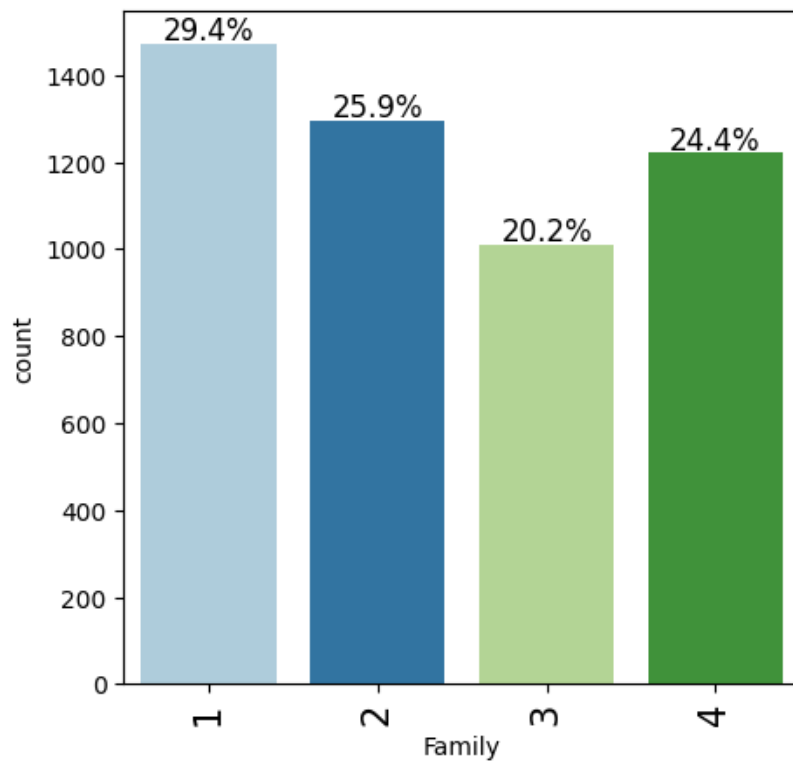
✓ Observations on Mortgage

```
histogram_boxplot(data, 'Mortgage') ## Complete t
```



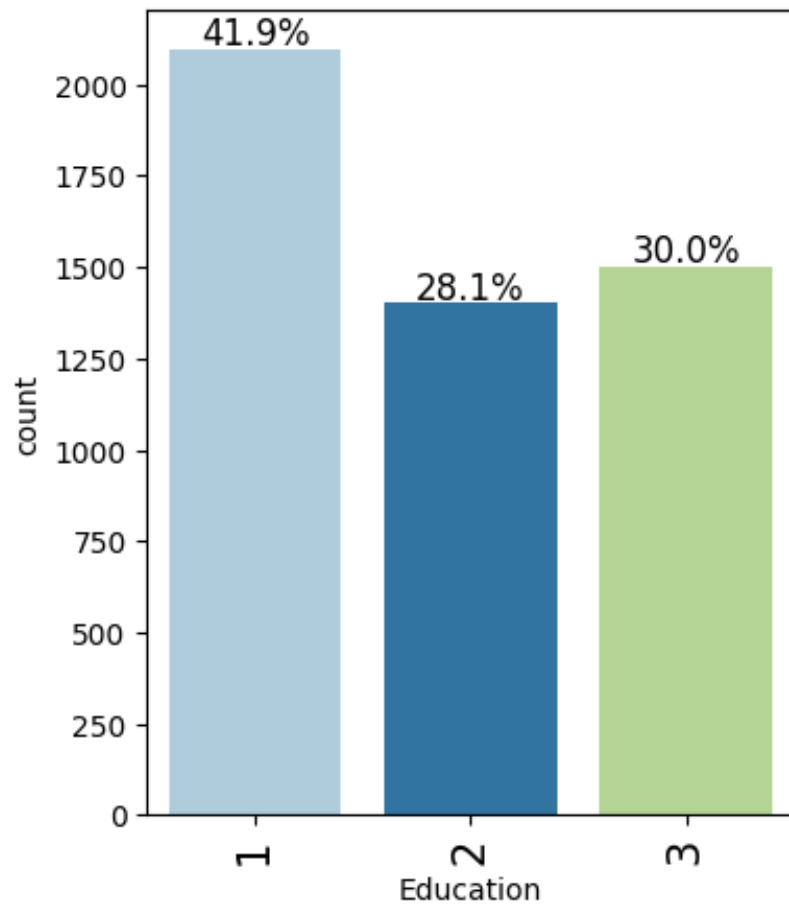
▼ Observations on Family

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



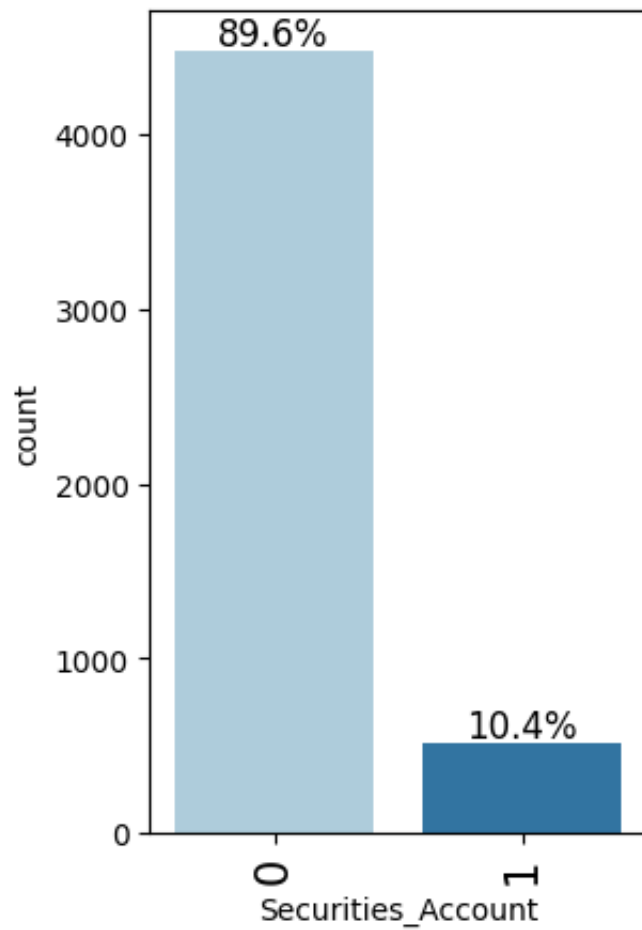
✓ Observations on Education

```
labeled_barplot(data, 'Education', perc=True)  ##
```



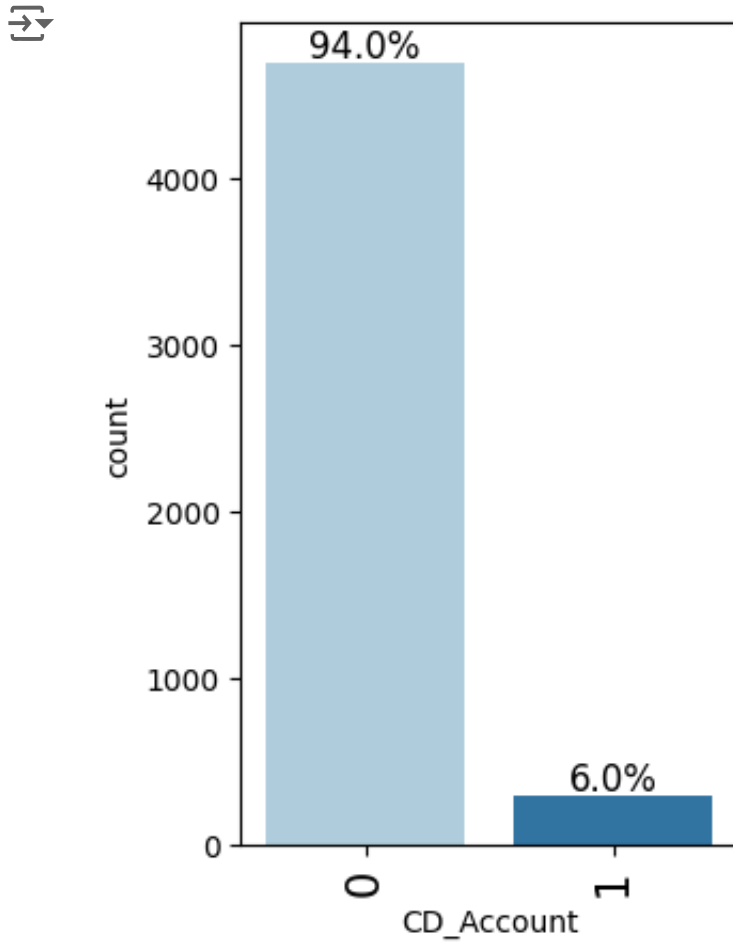
✓ Observations on Securities_Account

```
labeled_barplot(data, 'Securities_Account', perc=1
```



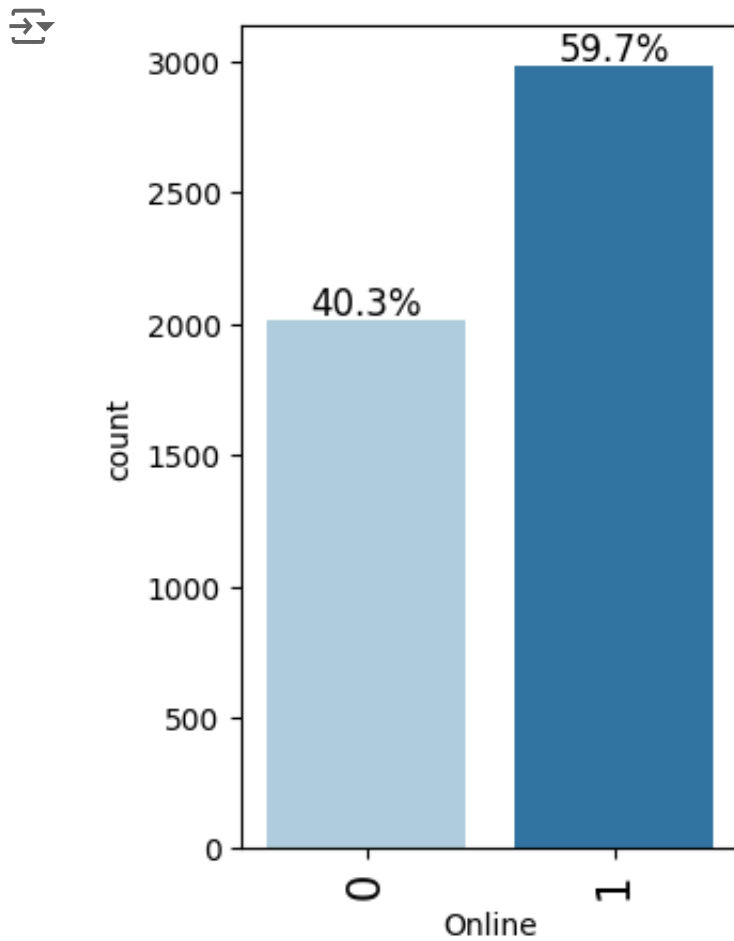
✓ Observations on CD_Account

```
labeled_barplot(data, 'CD_Account', perc=True) #
```



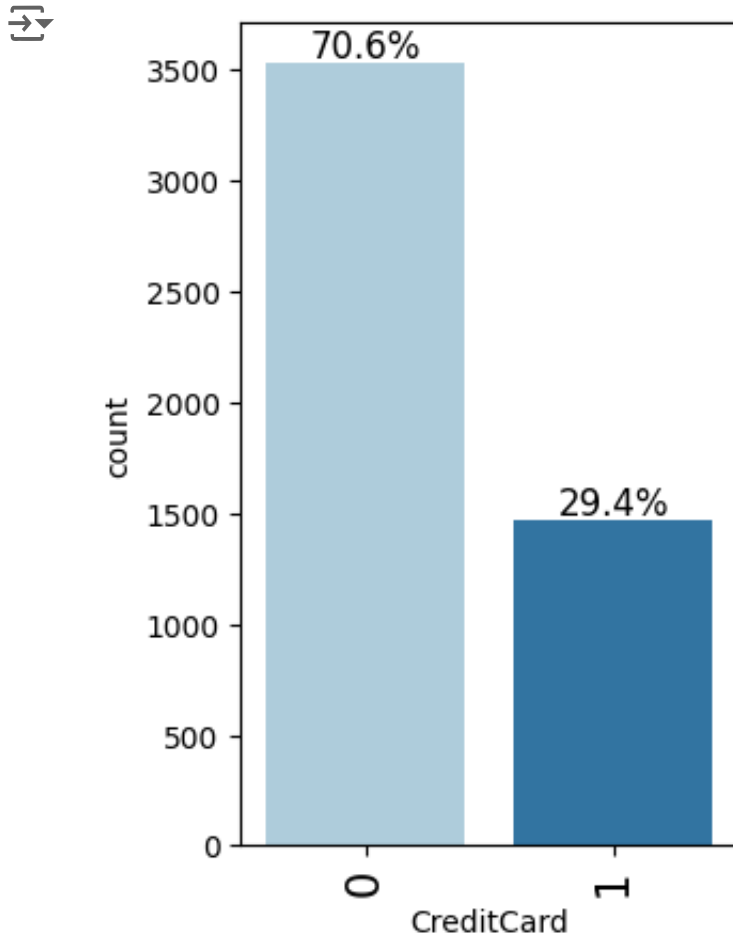
✓ Observations on Online

```
labeled_barplot(data, 'Online', perc=True)  ## Co
```



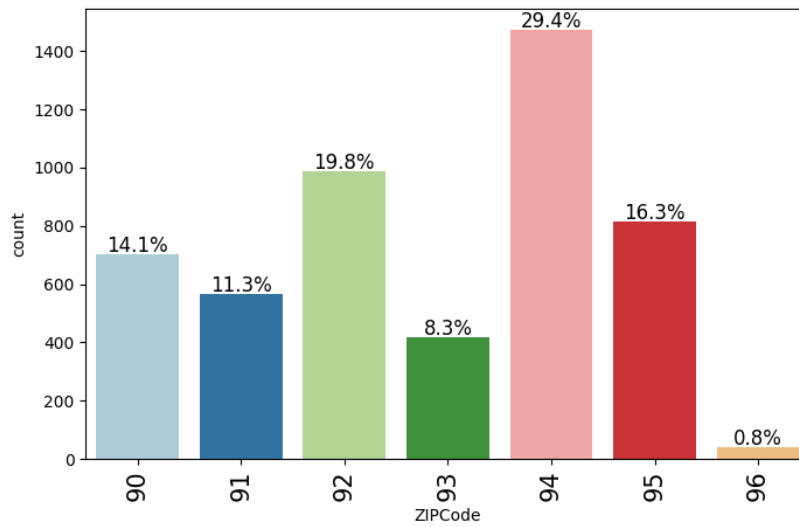
✓ Observation on CreditCard


```
labeled_barplot(data, 'CreditCard', perc=True) #
```



✓ Observation on ZIPCode

```
labeled_barplot(data, 'ZIPCode', perc=True)  ## (
```



✓ Bivariate Analysis

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

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target],
                       by=sorter, ascending=False)
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target],
                      by=sorter, ascending=False)
    tab.plot(kind="bar", stacked=True, figsize=(count, 10))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 0.5))
    plt.show()
```

function to plot distributions wrt target

```
def distribution_plot_wrt_target(data, predictor, target):
    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

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

```
axs[0, 1].set_title("Distribution of target for")
sns.histplot(
    data=data[data[target] == target_uniq[1]],
    x=predictor,
    kde=True,
    ax=axs[0, 1],
    color="orange",
    stat="density",
)

axs[1, 0].set_title("Boxplot w.r.t target")
sns.boxplot(data=data, x=target, y=predictor,

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

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

✓ Correlation check

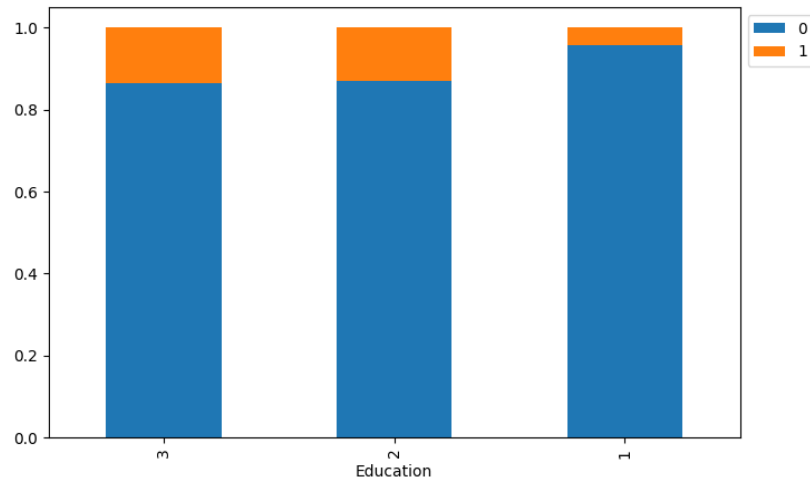
```
plt.figure(figsize=(15, 7))  
sns.heatmap(data.corr(numeric_only=True), annot=True,  
plt.show()
```



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

```
stacked_barplot(data, "Education", "Personal_Loan")
```

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

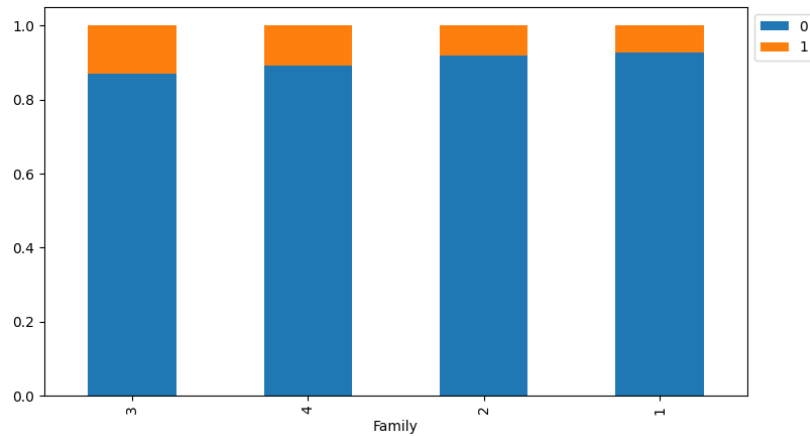


✓ Personal_Loan vs Family

```
stacked_barplot(data, 'Family','Personal_Loan') #
```

↔

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

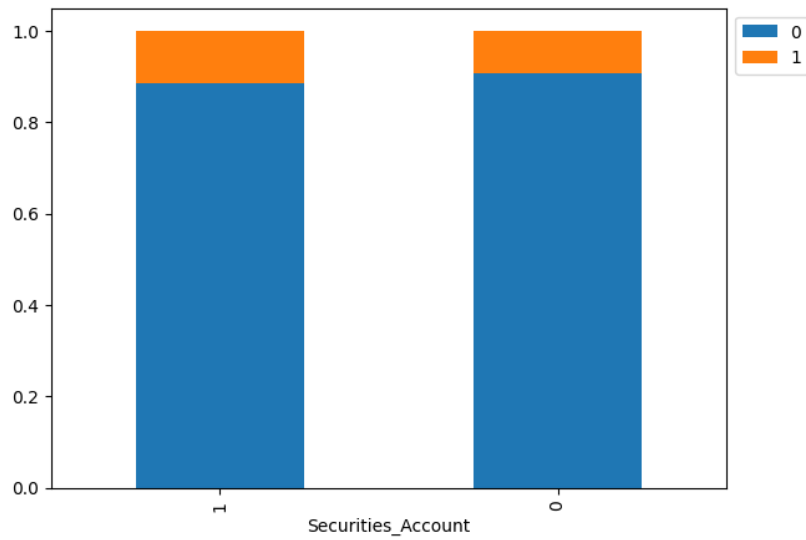


✓ Personal_Loan vs Securities_Account

```
stacked_barplot(data, 'Securities_Account', 'Personal_Loan')
```



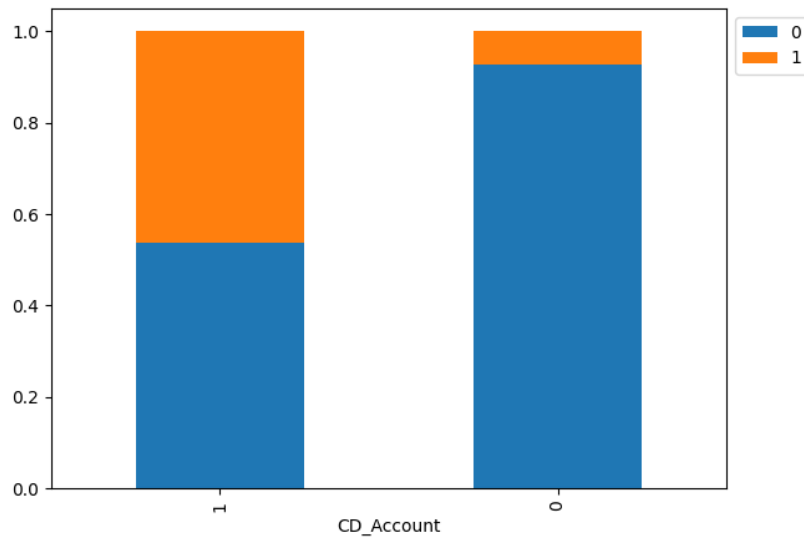
Personal_Loan	0	1	All
Securities_Account			
All	4520	480	5000
0	4058	420	4478
1	462	60	522



✓ Personal_Loan vs CD_Account


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

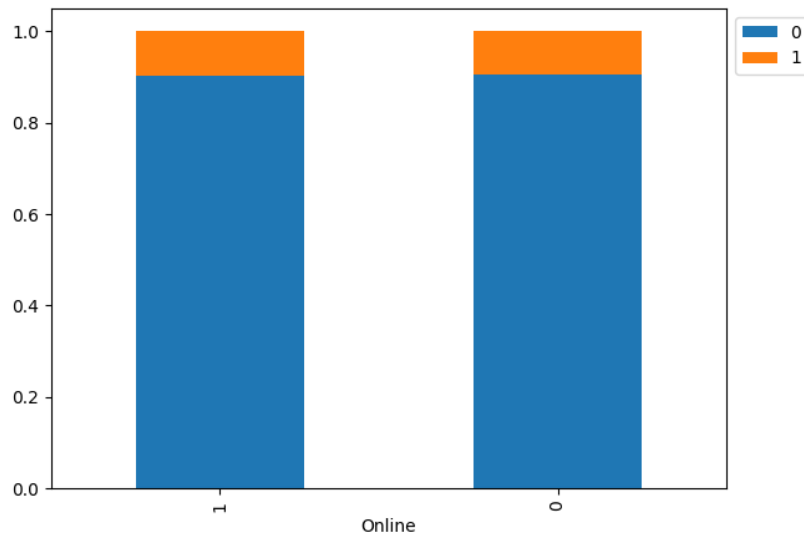
Personal_Loan	0	1	All
CD_Account			
All	4520	480	5000
0	4358	340	4698
1	162	140	302



✓ Personal_Loan vs Online

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

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

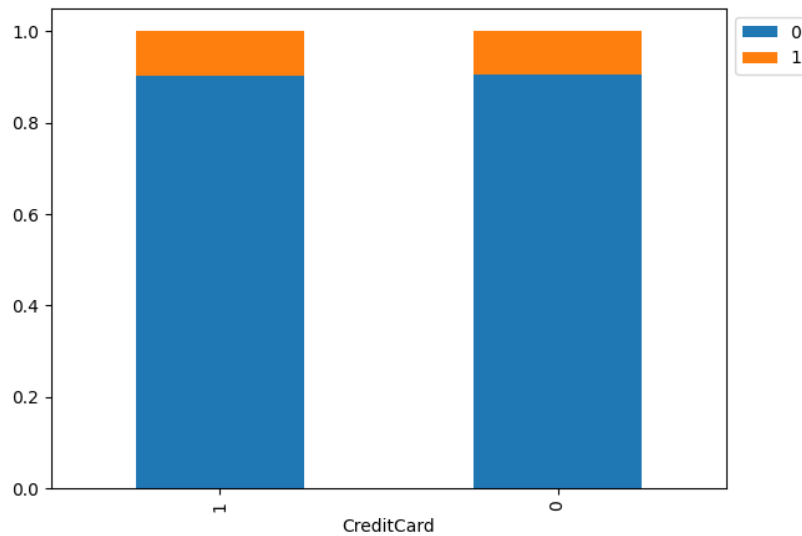


✓ Personal_Loan vs CreditCard

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

↗

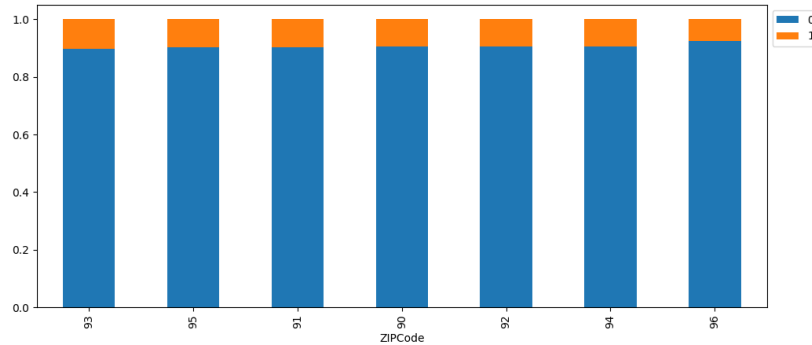
Personal_Loan	0	1	All
CreditCard			
All	4520	480	5000
0	3193	337	3530
1	1327	143	1470



✓ Personal_Loan vs ZIPCode

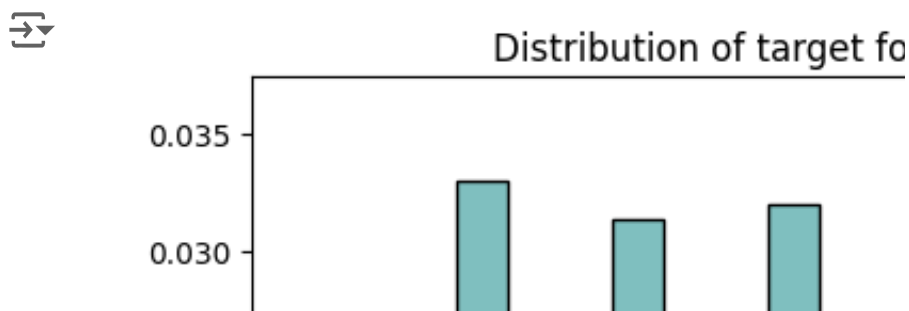
```
stacked_barplot(data, 'ZIPCode', 'Personal_Loan') #
```

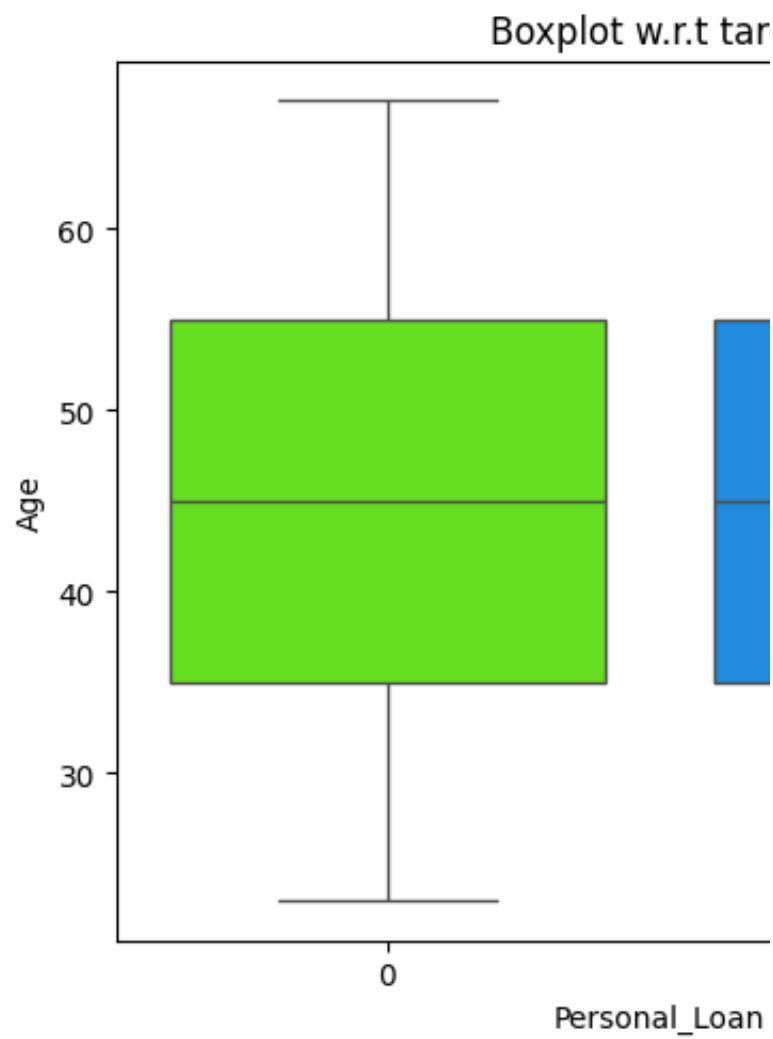
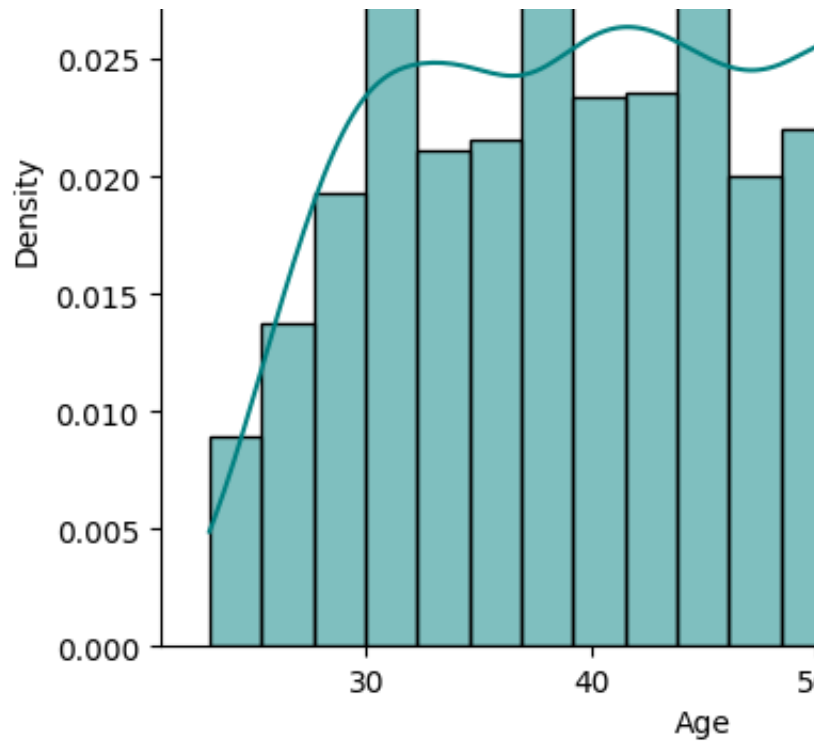
Personal_Loan	0	1	All
ZIPCode			
All	4520	480	5000
94	1334	138	1472
92	894	94	988
95	735	80	815
90	636	67	703
91	510	55	565
93	374	43	417
96	37	3	40



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

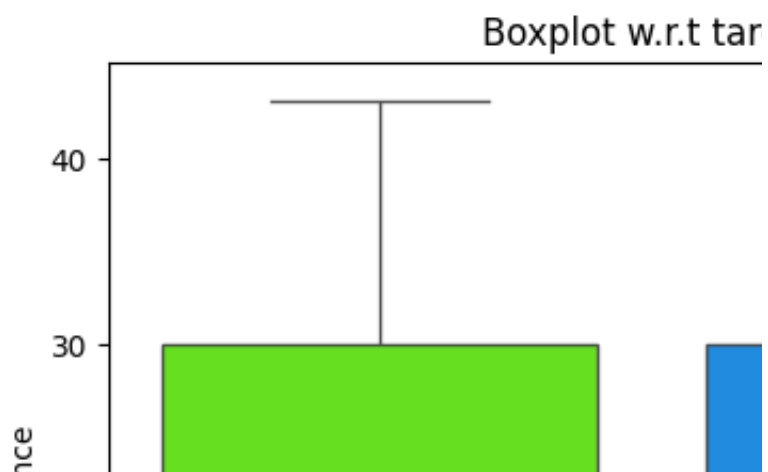
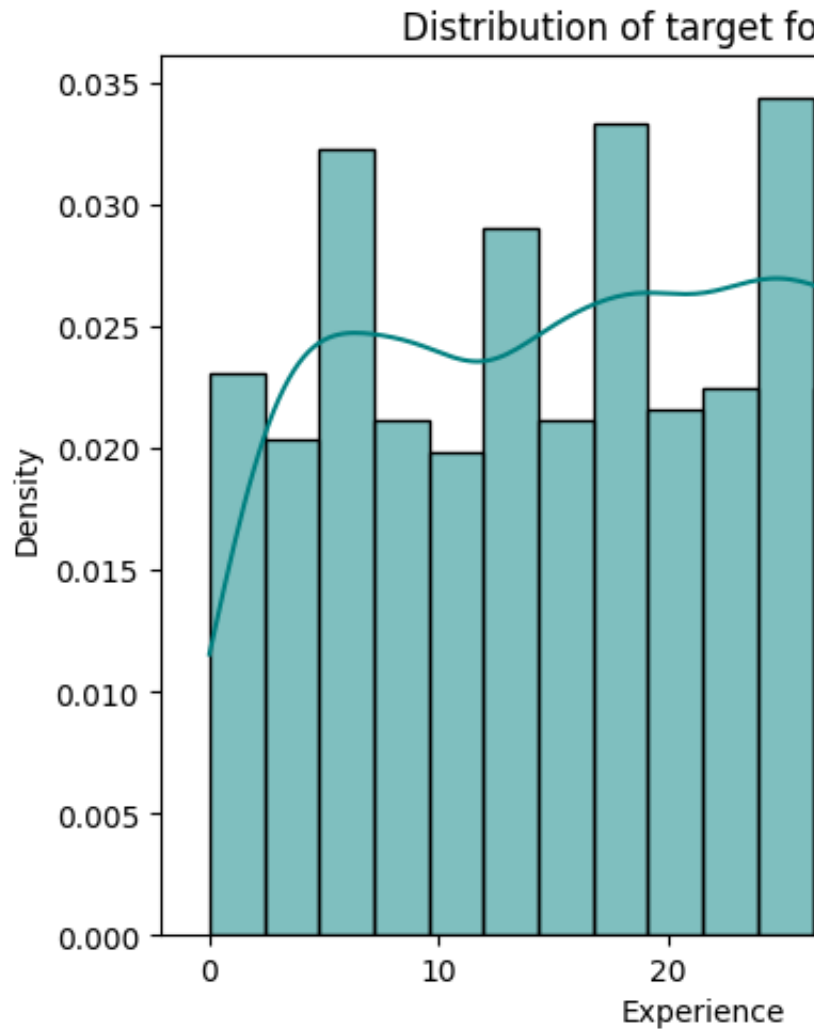
```
distribution_plot_wrt_target(data, "Age", "Personal_Loan")
```

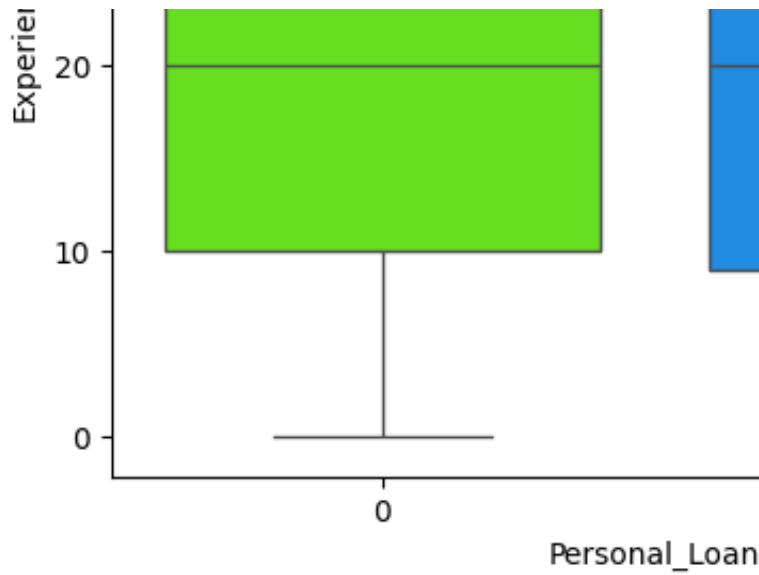




Personal Loan vs Experience

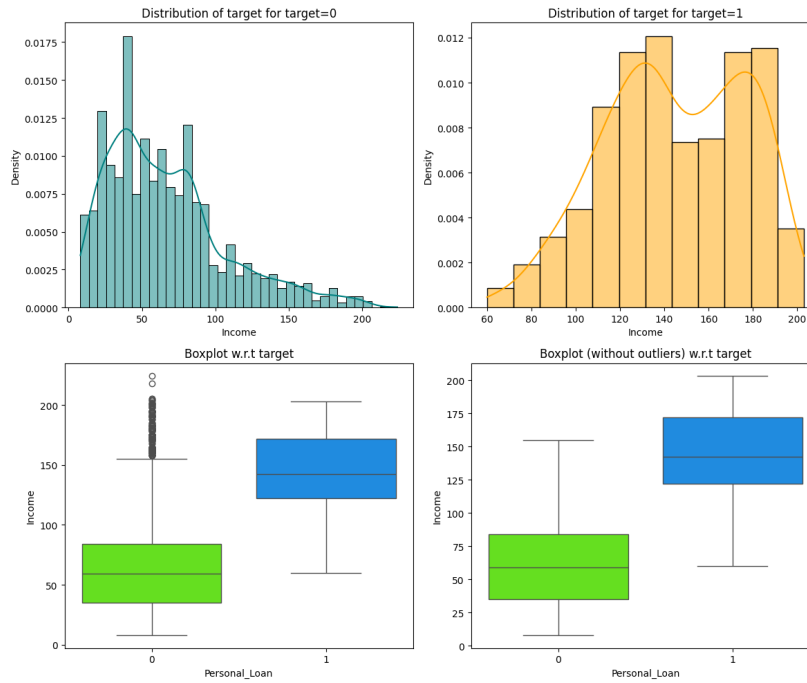
```
distribution_plot_wrt_target(data, 'Experience', 'Pe
```





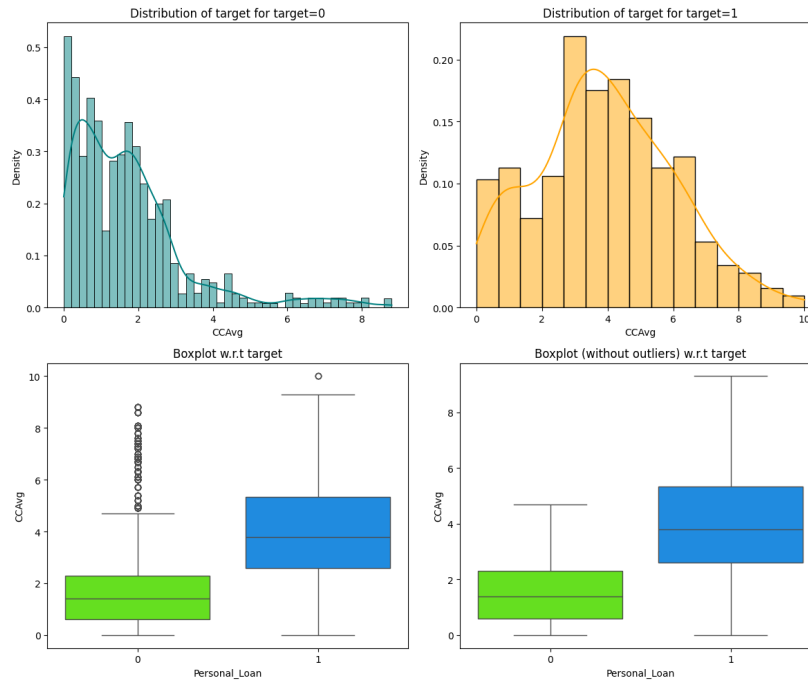
✓ Personal Loan vs Income

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



✓ Personal Loan vs CCAvg


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



✓ Data Preprocessing (contd.)

✓ Outlier Detection

```

Q1 = data.select_dtypes(include=["float64", "int64"])
Q3 = data.select_dtypes(include=["float64", "int64"])

IQR = Q3 - Q1 # Inter Quantile Range (75th percentil

lower = (
    Q1 - 1.5 * IQR
) # Finding lower and upper bounds for all values
upper = Q3 + 1.5 * IQR

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

```



	0
Age	0.00
Experience	0.00
Income	1.92
Family	0.00
CCAvg	6.48
Mortgage	5.82

dtype: float64

✓ Data Preparation for Modeling

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

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

X = X.astype(float)

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

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

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

✓ Model Building

✓ Model Evaluation Criterion

- *mention the model evaluation criterion here with proper reasoning*

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

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

```
# defining a function to compute different metrics
def model_performance_classification_sklearn(model):
    """
    Function to compute different metrics to check
    model performance.

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

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

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

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

    return df_perf
```

```
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%}"
            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")
```

Decision Tree (sklearn default)

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

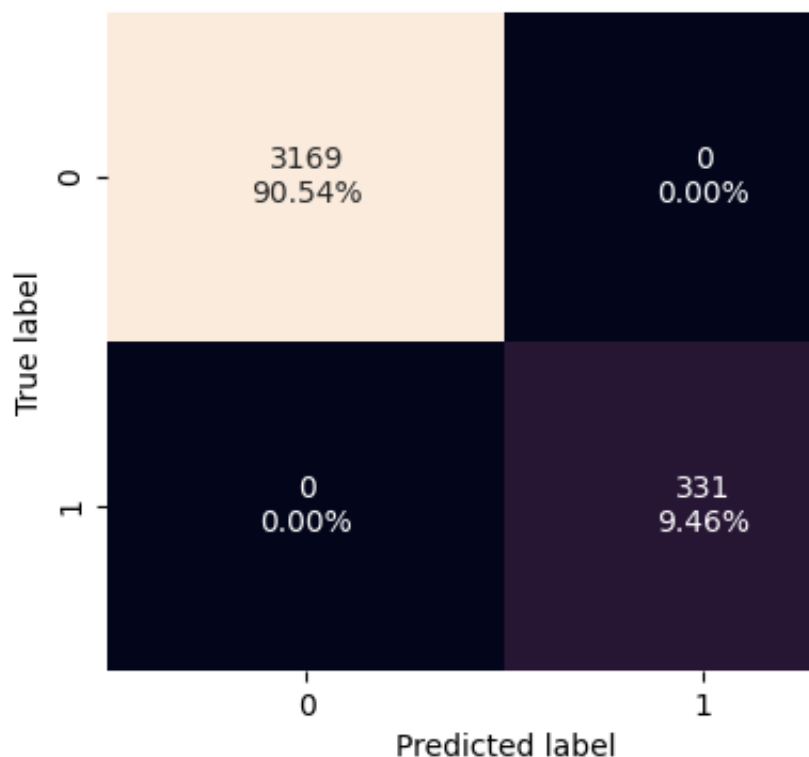


DecisionTreeClassifier
 i ?

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

Checking model performance on training data

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



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



	Accuracy	Recall	Precision	F1	
0	1.0	1.0	1.0	1.0	

Visualizing the Decision Tree

```
feature_names = list(X_train.columns)
print(feature_names)
```



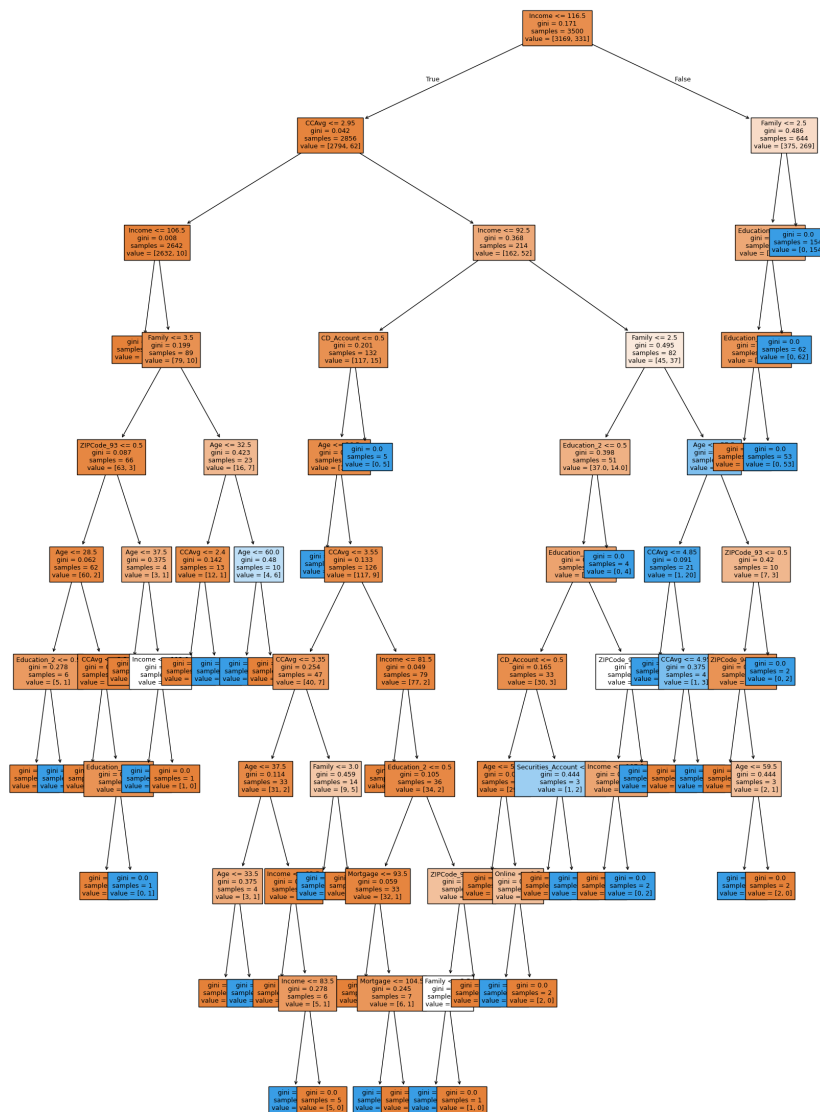
```
['Age', 'Income', 'Family', 'CCAvg', 'Mortgage']
```

```
plt.figure(figsize=(20, 30))
out = tree.plot_tree(
```

```

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

```




[illegible]

Page 49 of 74

```
# importance of features in the tree building ( Tf
# (normalized) total reduction of the criterion b

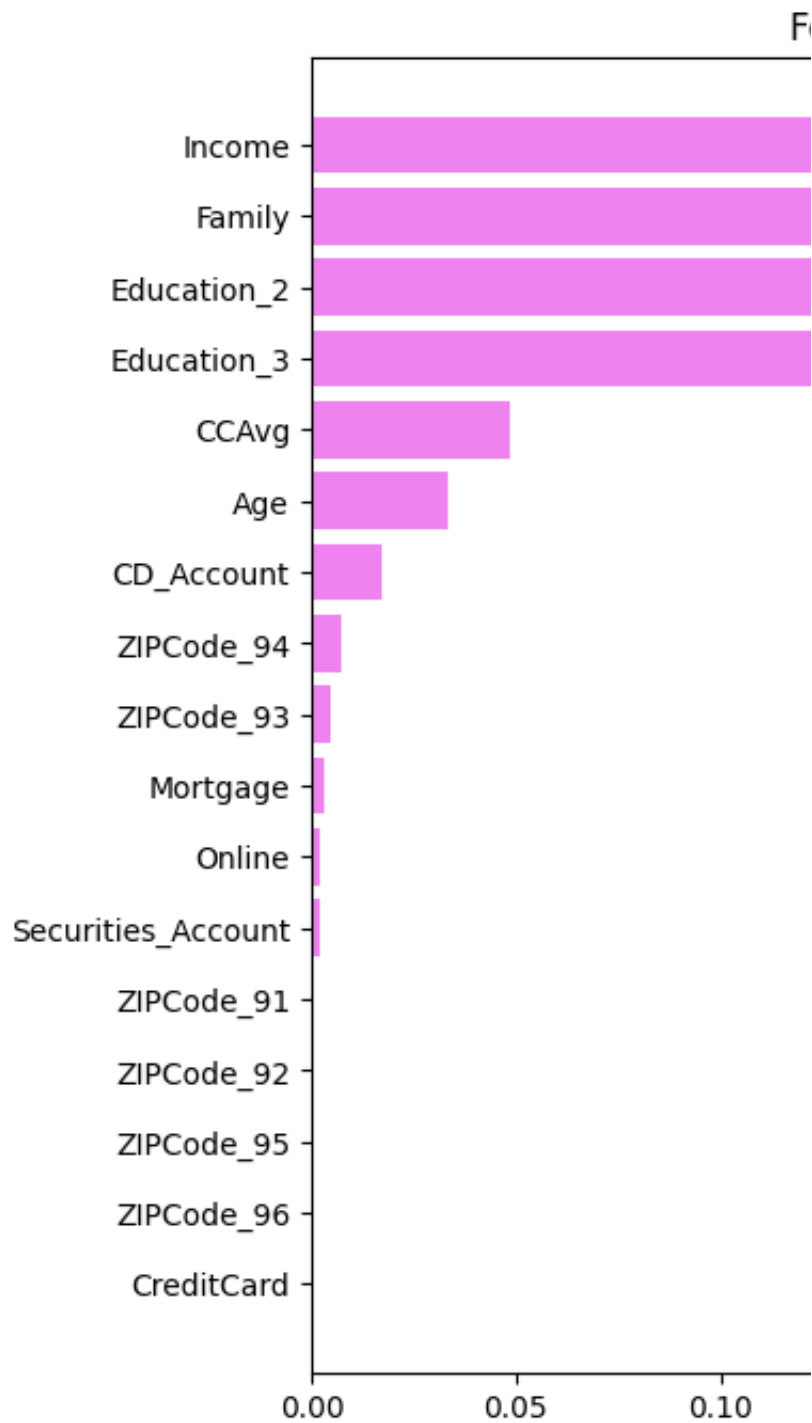
print(
    pd.DataFrame(
        model.feature_importances_, columns=["Imp'
    ).sort_values(by="Imp", ascending=False)
)
```



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

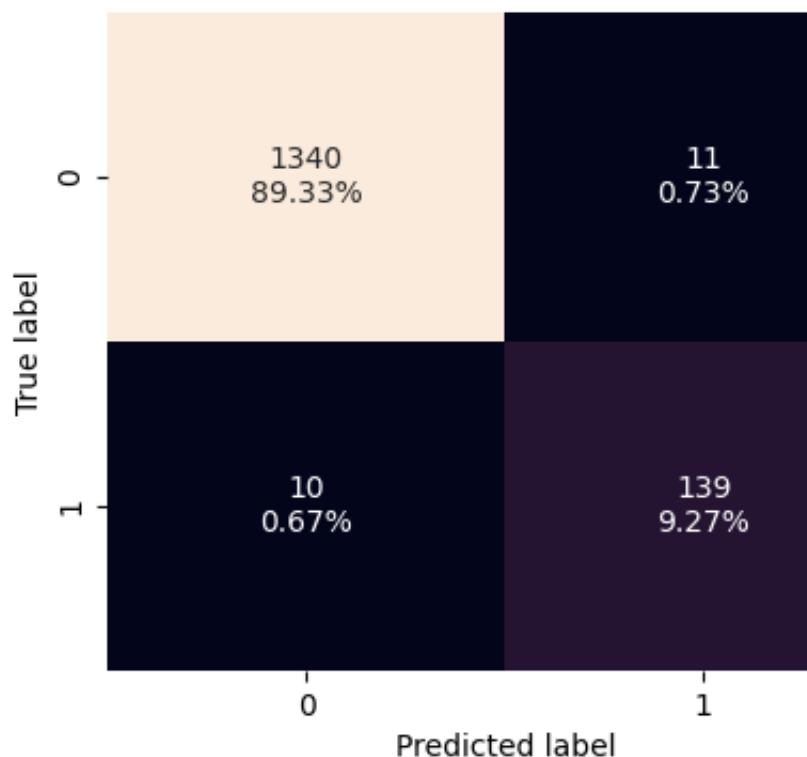
```
importances = model.feature_importances_
indices = np.argsort(importances)

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



✓ Checking model performance on test data

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



```
decision_tree_perf_test = model_performance_class:
decision_tree_perf_test
```



	Accuracy	Recall	Precision	F1	
0	0.986	0.932886	0.926667	0.929766	

✓ Model Performance Improvement

✓ Pre-pruning

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

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

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

# Iterate over all combinations of the specified parameters
for max_depth in max_depth_values:
    for max_leaf_nodes in max_leaf_nodes_values:
        for min_samples_split in min_samples_split_values:

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

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

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

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

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

            # Update the best estimator and best score if the current model is better
            if (score_diff < best_score_diff) & (test_recall_score > best_test_score):
                best_score_diff = score_diff
                best_test_score = test_recall_score
                best_estimator = estimator

# Print the best parameters
```

```
print("Best parameters found:")
print(f"Max depth: {best_estimator.max_depth}")
print(f"Max leaf nodes: {best_estimator.max_leaf_r
print(f"Min samples split: {best_estimator.min_sar
print(f"Best test recall score: {best_test_score}'
```

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

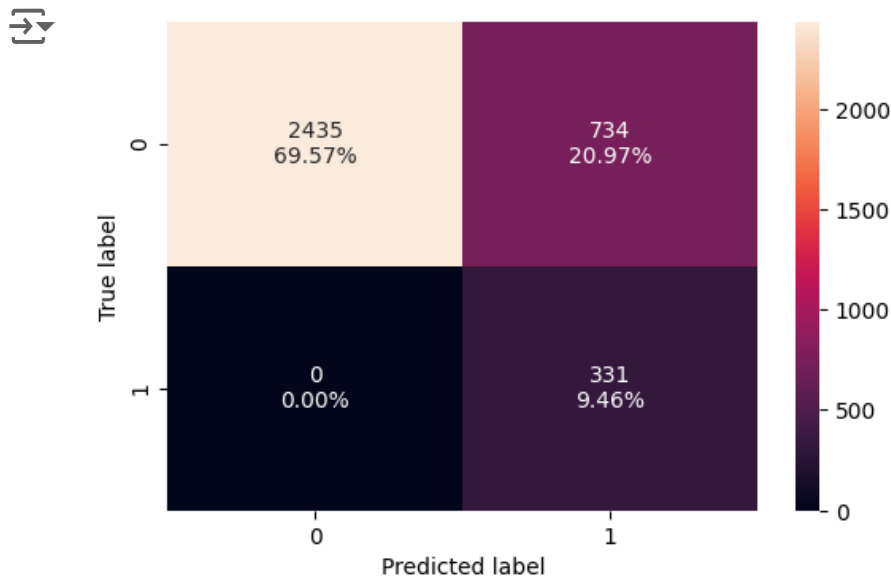
```
# Fit the best algorithm to the data.
estimator = best_estimator
estimator.fit(X_train,y_train) ## Complete the coo
```

➞


```
▼ DecisionTreeClassi
DecisionTreeClassifier(class_weight='balanced
                        min_samples_split=10, 1
```

Checking performance on training data



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



```
decision_tree_tune_perf_train = model_performance_
decision_tree_tune_perf_train
```



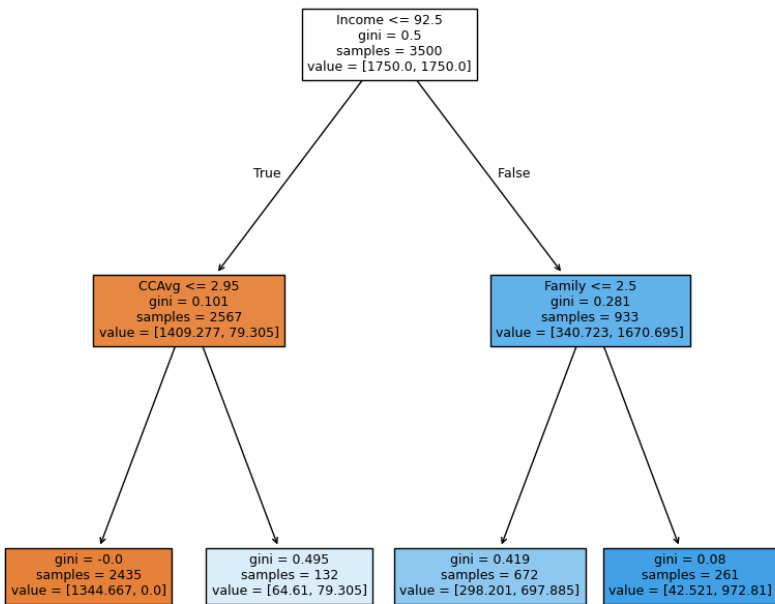
	Accuracy	Recall	Precision	F1
0	0.790286	1.0	0.310798	0.474212

Visualizing the Decision Tree

```
plt.figure(figsize=(10, 10))
out = tree.plot_tree(
    estimator,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# below code will add arrows to the decision tree
```

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




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

```
print(tree.export_text(estimator, feature_names=fe
```

```

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

```

```
# importance of features in the tree building ( T
# (normalized) total reduction of the criterion b
```

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

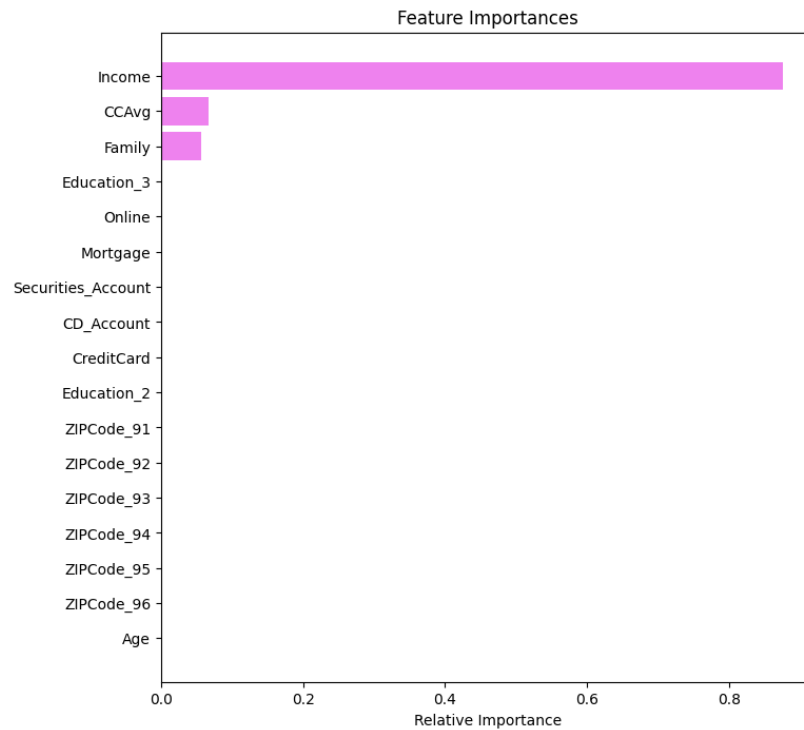
```

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

```

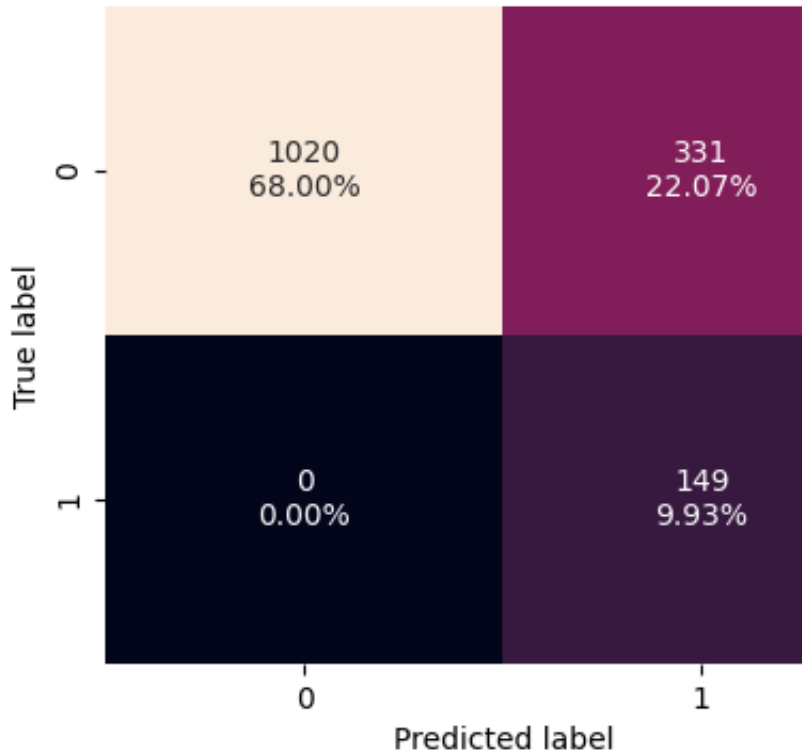
```
importances = estimator.feature_importances_
indices = np.argsort(importances)

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



Checking performance on test data

```
confusion_matrix_sklern(estimator, X_test, y_test)
```



```
decision_tree_tune_perf_test = model_performance_
decision_tree_tune_perf_test
```



	Accuracy	Recall	Precision	F1	
0	0.779333	1.0	0.310417	0.473768	



✓ Post-pruning

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

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

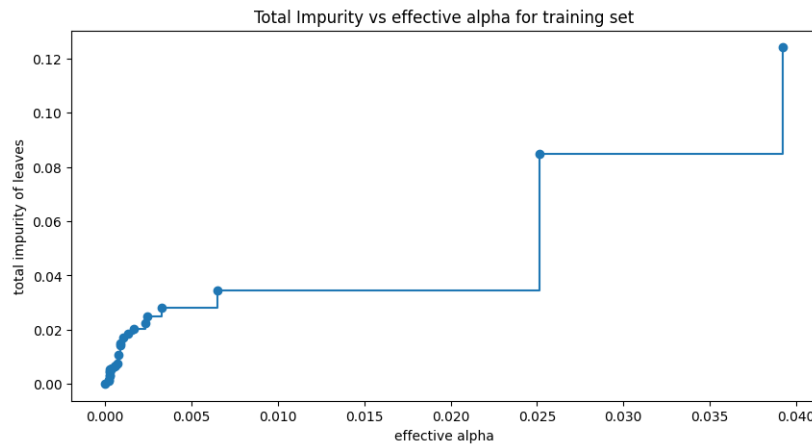


ccp_alphas **impurities**



0	0.000000	0.000000
1	0.000186	0.001114
2	0.000214	0.001542
3	0.000242	0.002750
4	0.000250	0.003250
5	0.000268	0.004324
6	0.000272	0.004868
7	0.000276	0.005420
8	0.000381	0.005801
9	0.000527	0.006329
10	0.000625	0.006954
11	0.000700	0.007654
12	0.000769	0.010731
13	0.000882	0.014260
14	0.000889	0.015149
15	0.001026	0.017200
16	0.001305	0.018505
17	0.001647	0.020153
18	0.002333	0.022486
19	0.002407	0.024893
20	0.003294	0.028187
21	0.006473	0.034659
22	0.025146	0.084951
23	0.039216	0.124167
24	0.047088	0.171255

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



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

```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, c
    clf.fit(X_train, y_train)    ## Complete the
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

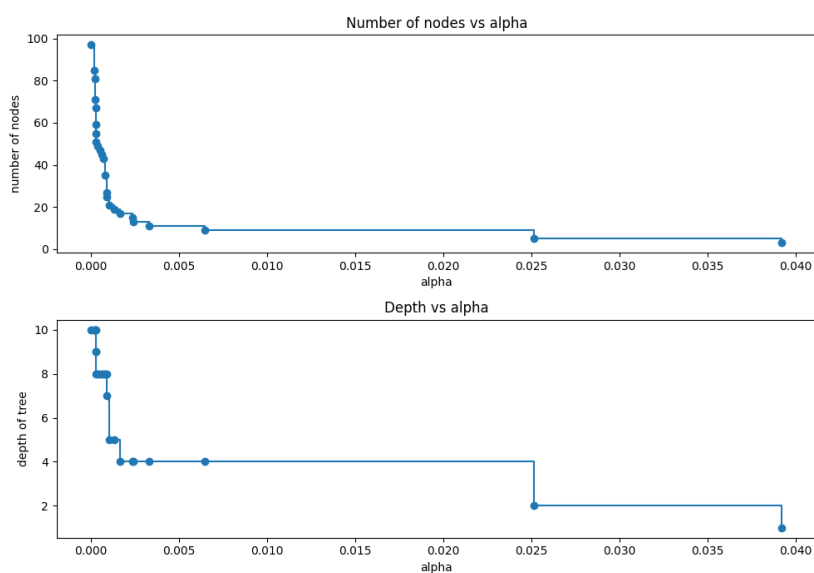
⇒ Number of nodes in the last tree is: 1 with cc

```

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="step")
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="step")
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

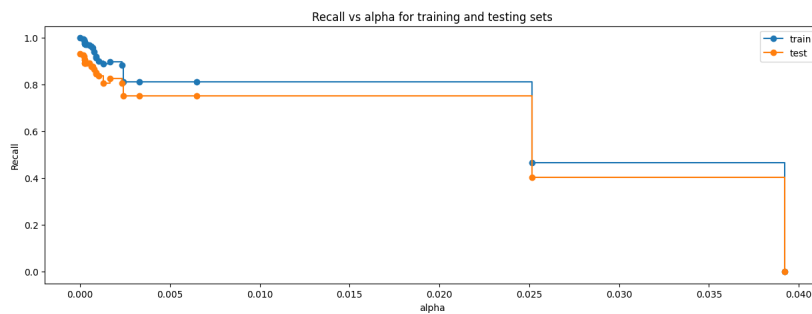
```

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

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

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")
ax.plot(ccp_alphas, recall_test, marker="o", label="test")
ax.legend()
plt.show()

```




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

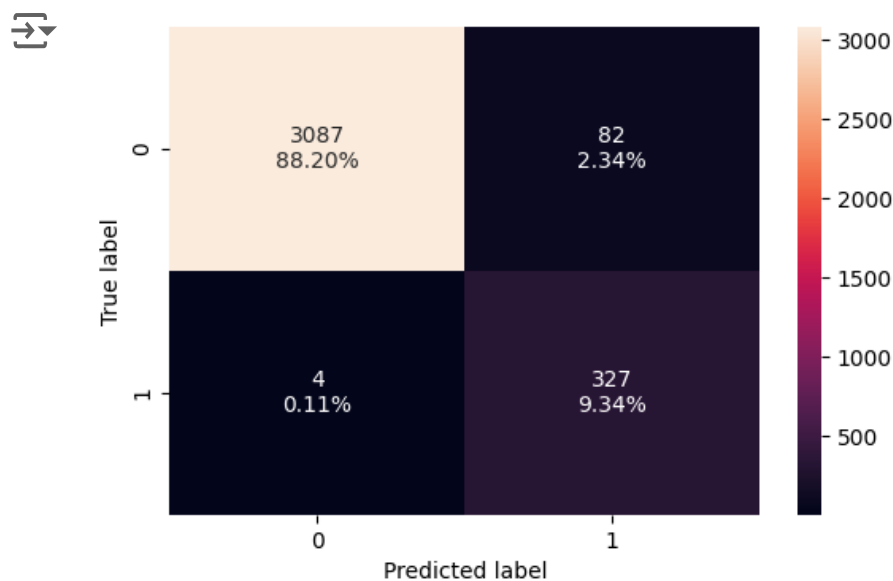
```
DecisionTreeClassifier(random_state=1)
```

```
estimator_2 = DecisionTreeClassifier(
    ccp_alpha=.002407, class_weight={0: 0.15, 1: 0.
)
estimator_2.fit(X_train, y_train)
```

```
DecisionTreeClassifier(
    ccp_alpha=0.002407, class_weight={0: 0.15, 1: 0.
    random_state=1)
```

Checking performance on training data

```
confusion_matrix_sklern(estimator_2, X_train, y_train)
```



```
decision_tree_tune_post_train = model_performance_
decision_tree_tune_post_train
```

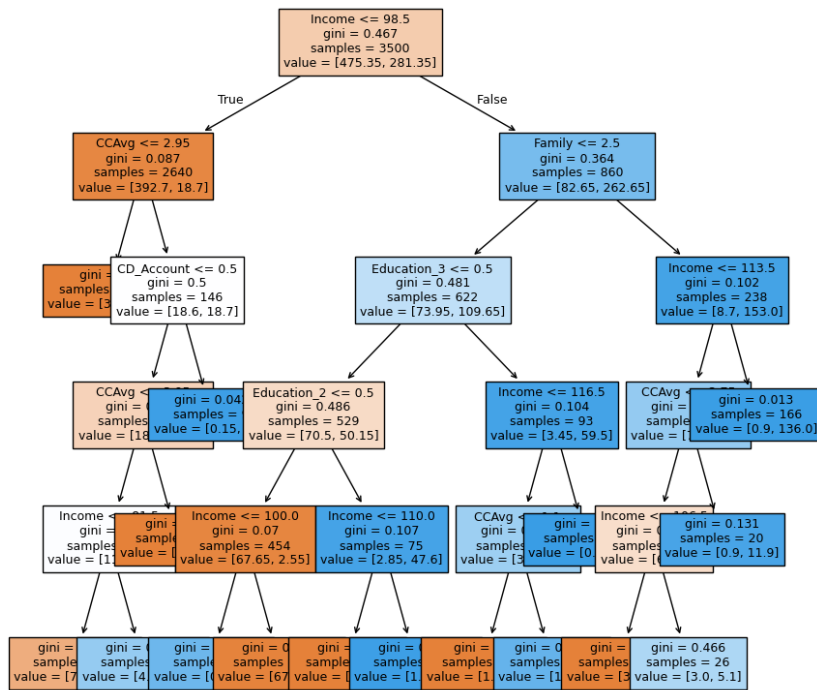


	Accuracy	Recall	Precision	F1
0	0.962857	0.97281	0.726862	0.832041



Visualizing the Decision Tree

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




```
# Text report showing the rules of a decision tree
print(tree.export_text(estimator_2, feature_names=
```



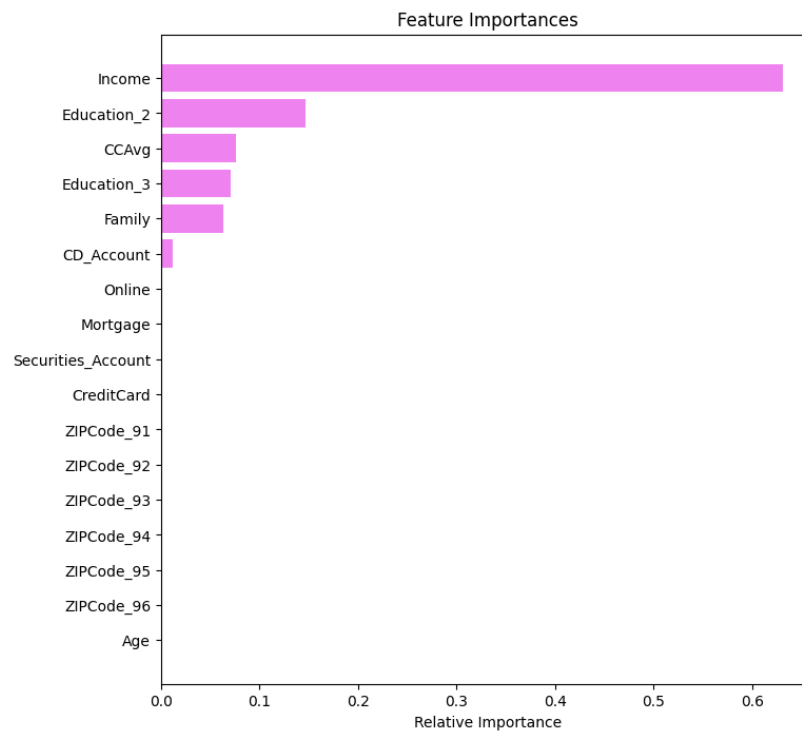
```
# importance of features in the tree building ( T
# (normalized) total reduction of the criterion b

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



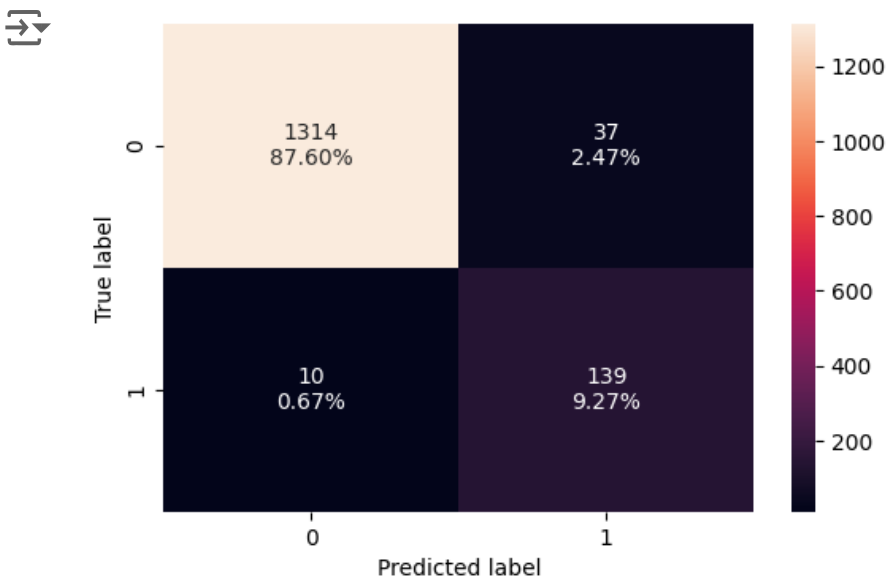
	Imp
Income	0.631552
Education_2	0.146714
CCAvg	0.075895
Education_3	0.070443
Family	0.063589
CD_Account	0.011806
ZIPCode_93	0.000000
ZIPCode_96	0.000000
ZIPCode_95	0.000000
ZIPCode_94	0.000000
Age	0.000000
ZIPCode_92	0.000000
ZIPCode_91	0.000000
Online	0.000000
Securities_Account	0.000000
Mortgage	0.000000
CreditCard	0.000000

```
importances = estimator_2.feature_importances_  
indices = np.argsort(importances)  
  
plt.figure(figsize=(8, 8))  
plt.title("Feature Importances")  
plt.barh(range(len(indices)), importances[indices])  
plt.yticks(range(len(indices)), [feature_names[i]  
plt.xlabel("Relative Importance")  
plt.show()
```



Checking performance on test data

```
confusion_matrix_sklearn(estimator_2, X_test, y_test)
```



```
decision_tree_tune_post_test = model_performance_0  
decision_tree_tune_post_test
```




	Accuracy	Recall	Precision	F1
0	0.968667	0.932886	0.789773	0.855385

Model Performance Comparison and Final Model Selection

```
# training performance comparison
```

```
models_train_comp_df = pd.concat(
    [decision_tree_perf_train.T, decision_tree_tur
)
models_train_comp_df.columns = ["Decision Tree (sklearn
print("Training performance comparison:")
models_train_comp_df
```

↔ Training performance comparison:

	Decision Tree (sklearn default)	Decision Tree (Pre- Pruning)	Decision Tree (Post- Pruning)	  
Accuracy	1.0	0.790286	0.962857	
Recall	1.0	1.000000	0.972810	
Precision	1.0	0.310798	0.726862	

Next
steps:

[code](#) [models_train_comp_df](#)

[recomm](#)


```
# testing performance comparison

models_test_comp_df = pd.concat(
    [decision_tree_perf_test.T, decision_tree_tune_
)
models_test_comp_df.columns = ["Decision Tree (skle
print("Test set performance comparison:")
models_test_comp_df
```

↗ Test set performance comparison:

	Decision Tree (sklearn default)	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.986000	0.779333	0.968667
Recall	0.932886	1.000000	0.932886
Precision	0.926667	0.310417	0.789773

Next steps:

code

models_test_comp_df

recommen

Actionable Insights and Business Recommendations

What recommendations would you suggest to the bank?

Enter a prompt here

0 / 1000

Responses may display inaccurate or offensive information that doesn't represent Google's views. [Learn more](#)