Personal Loan Campaign

Project : Machine Learning

POST GRADUATE PROGRAM IN AI & MACHINE LEARNING: BUSINESS APPLICATIONS

Saroj Ekka
Sep 29th 2025

Contents / Agenda

- Executive Summary
- Business Problem Overview and Solution Approach
- EDA Results
- Data Preprocessing
- Model Building
- Model Performance Summary
- Appendix

Executive Summary

Actionable Insights

- **Customer Profile Matters:**
  - **Income** is the strongest predictor → higher-income customers are far more likely to accept personal loans.
  - **Education** (Graduate & Advanced) strongly correlates with loan acceptance.
  - **High credit card spenders (CCAvg)** and customers with **family** show higher probability of conversion.
  - Existing **CD Account** holder also possible candidates

- **Model Performance:**
  - The **Post-Pruned Decision Tree** gives the best balance of accuracy (**97%**), recall (**85%**), and precision (**93%**) on test data.
  - Ensures reliable predictions and interpretable rules for business use.

- **Marketing Efficiency:**
  - High recall from the model minimizes missed opportunities.
  - High precision reduces wasted marketing efforts on unlikely customers**.**

Executive Summary

Recommendations

1. **Target Segments for Loan Campaigns:**
   - Customers with **high income**, **graduate/advanced education**, and **high credit card usage**.
   - Special focus on customers who have family and who already hold a **CD Account**.

2. **Campaign Strategy:**
   - Personalized offers to high-value segments to maximize conversion.
   - Bundled financial products (e.g., CD + Personal Loan) to leverage cross-selling.

3. **Operational Use:**
   - Deploy the **Post-Pruned Decision Tree** as a decision-support tool in marketing workflows.
   - Continuously monitor campaign results and retrain model periodically to adapt to changing customer behavior.

✅ **Key Takeaway:**
By focusing on the identified high-potential customer segments, AllLife Bank can **improve loan conversion rates**, optimize marketing spend, and strengthen customer relationships.

Business Problem Overview and Solution Approach

Overview

AllLife Bank is primarily a liability-focused bank (depositors) with varying sizes of deposits but wants to expand its asset base by increasing personal loan customers.

Observation & Challenge:

- The number of customers who are also borrowers (asset customers) is quite small,
- Last year's marketing campaign show healthy conversion rate ~9%.

Business Objectives:

- Predict whether a liability customer will purchase a personal loan.
- Identify key customer attributes that influence loan purchase decisions.
- Provide recommendations for targeted marketing campaigns to improve conversion rates.c

Business Problem Overview and Solution Approach

Solution Approach / Methodology

The problem can be solved through **Exploratory Data Analysis (EDA), statistical insights, and visualization techniques**, Decision tree, model building & pruning

Step1: Data Understanding & Preparation

- Collect and explore customer dataset (demographics, income, spending, accounts, etc.)
- Handle duplicate/missing values, detect outliers, and prepare features for modeling.
- Convert categorical variables (weekday/weekend, cuisine type, etc.) if needed.

Step2: Exploratory Data Analysis (EDA)

- **Univariate Analysis**: Understand distribution of individual features (age, income, education, credit card usage, etc.).
- **Bivariate Analysis**: Explore relationships between customer attributes and loan purchase.
- Identify key patterns (e.g., higher income → higher loan acceptance).

Step3: Model Building (Decision Tree Classifier)

- Train initial decision tree to predict loan purchase.
- Evaluate using accuracy, precision, recall, F1 score.
- Visualize decision tree and identify most important features.

Step4: Model Improvement (Pruning)

- Apply pre-pruning (limiting tree depth, min samples per split).
- Apply post-pruning to reduce overfitting.
- Compare model performance before and after pruning.

Step5: Insights & Recommendations

- Highlight the most influential features (Income, CCAvg, Education, CD Account).
- Suggest which customer segments to target in marketing campaigns.

Step6: Business Impact

- Enable bank to increase loan conversions beyond 9% by focusing efforts on the most promising customers.
- Improve campaign ROI by reducing wasted outreach.

Data Overview - Personal Loan Campaign Dataset

📋 Data Summary

- **Dataset Size:** 5,000 customers (rows), 14 attributes (columns).
- **Data Type:** Mix of demographic, financial, and banking behavior features.
- **Objective:** Predict whether a liability customer will buy a personal loan.

🗒️ Fields in the Dataset

1. **ID** – Customer ID (unique identifier).
2. **Age** – Age in years.
3. **Experience** – Years of professional experience.
4. **Income** – Annual income (in $000s).
5. **ZIP Code** – Residential ZIP code.
6. **Family** – Family size.
7. **CCAvg** – Average monthly credit card spend (in $000s).
8. **Education** – Education level:
    - 1 = Undergrad
    - 2 = Graduate
    - 3 = Advanced/Professional
9. **Mortgage** – Value of house mortgage (in $000s).
10. **Personal_Loan** – Target variable: 1 = Accepted loan, 0 = Not accepted.
11. **Securities_Account** – Has a securities account (Yes/No).
12. **CD_Account** – Has a certificate of deposit account (Yes/No).
13. **Online** – Uses internet banking (Yes/No).
14. **CreditCard** – Uses credit card issued by another bank (Yes/No).

📊 Summary Statistics (Key Numeric Fields)

- **Age:** Mean 45.3 years, Median 45 → mostly middle-aged.
- **Experience:** Mean 20 years, Median 20 → experienced professionals.
- **Income:** Mean $74K, Median $64K → skewed by high-income outliers.
- **CCAvg:** Mean $1.94K, Median $1.5K → most spend low, some high spenders.
- **Mortgage:** Mean $56K, Median $0 → many customers have no mortgage.
- **Education:** Median 2 (Graduates most common).
- **Family Size:** Mean 2.4, Median 2 → small families dominate.

🎯 Key Points

- Only **9.6% customers** accepted the personal loan → strong class imbalance.
- **High-income, educated, high credit card spenders, and CD account holders** show more potential for loan conversion.
- **Most customers do not have mortgages, CD accounts, or securities accounts.**
- **Online banking** usage is high (~60%), and ~30% hold external credit cards.
- The dataset is **well-structured with no major missing values**, but has **skewness and outliers** in Income, CCAvg, and Mortgage.

EDA Results

Univariate Analysis (each feature individually)

- ★ **Age & Experience** – Most customers are between 30–55 years; distribution is fairly normal.
- ★ **Income** – Right-skewed; most customers earn <$100K, few high-income outliers.
- ★ **Family Size** – Majority have families of size 1–2; large families are rare.
- ★ **Education** – Most are undergraduates, fewer graduates, very few advanced/professional.
- ★ **CCAvg (Monthly Card Spend)** – Skewed; most spend <$3K, but some spend much higher.
- ★ **Mortgage** – Many have no mortgage (0), while others vary widely (with extreme outliers).
- ★ **Binary Variables:**
  - Securities Account: Most do not have.
  - CD Account: Very few have.
  - Online Banking: Majority use.
  - Credit Card: Large share have credit cards (non-AllLife).
- ★ **Target (Personal Loan):** Only ~9–10% accepted loan → dataset is imbalanced.

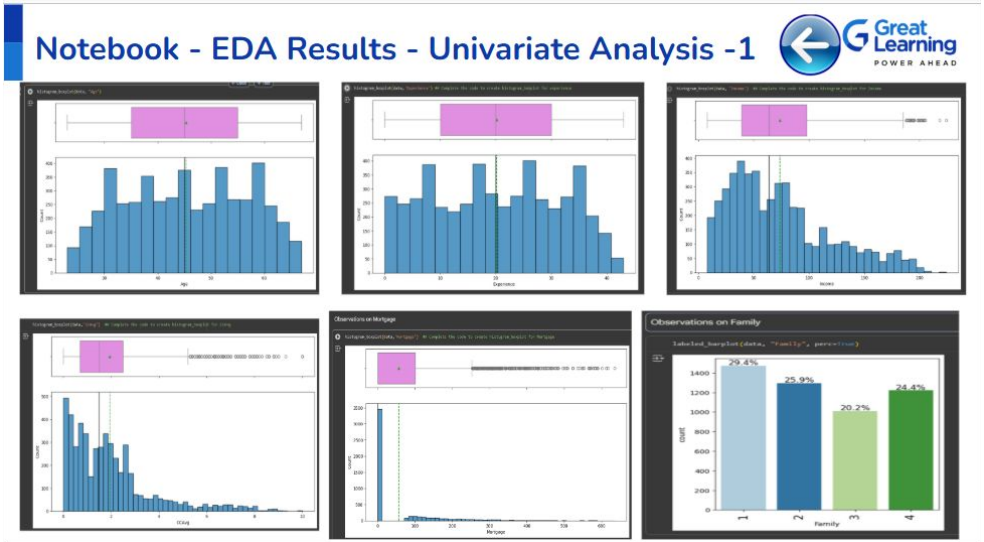
Bivariate Analysis (features vs. loan acceptance)

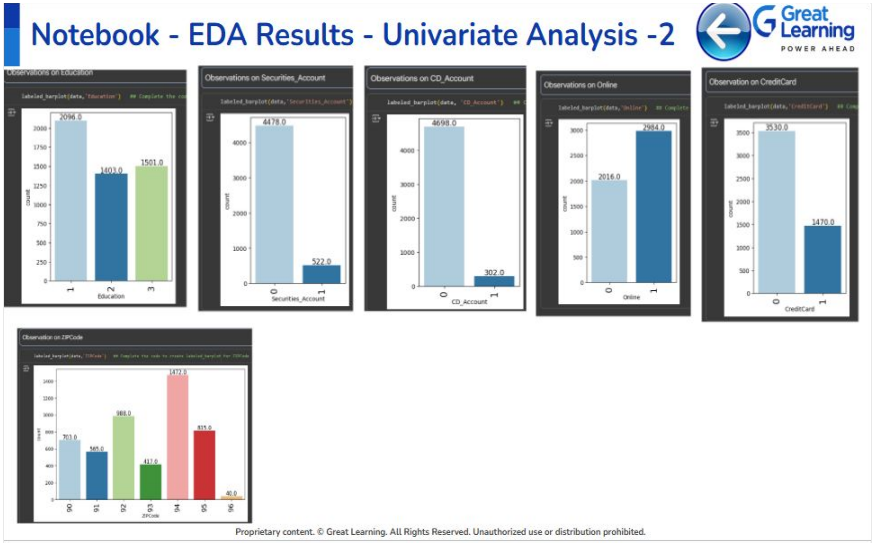
- ★ **Income vs Loan:** Higher income customers are far more likely to accept personal loans.
- ★ **Education vs Loan:** Loan acceptance rises with education level — graduates and advanced professionals convert more.
- ★ **CCAvg vs Loan:** Customers with higher credit card spending (> $3K/month) are more inclined to take personal loans.
- ★ **CD Account vs Loan:** Having a CD account is strongly correlated with accepting a personal loan.
- ★ **Age, Family, Mortgage:** No strong correlation with loan acceptance.
- ★ **Online Banking & Credit Card:** Slight positive impact, but weaker compared to Income/Education/CCAvg/CD Account.

✅ Key EDA Insights

- ➔ **Experience as it is perfectly correlated with Age**
- ➔ **Strong predictors of loan acceptance:** Income, Education, Credit Card Spend (CCAvg), CD Account.
- ➔ **Weak predictors:** ZIP Code, Family size, Mortgage, Age.
- ➔ Target segment seems to be: **well-educated, high-income customers with higher card spending and CD accounts.**

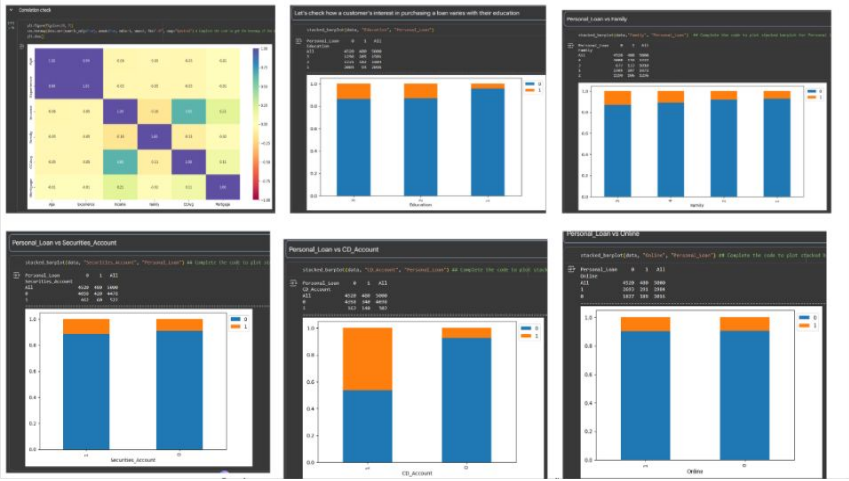
EDA Results - NoteBook



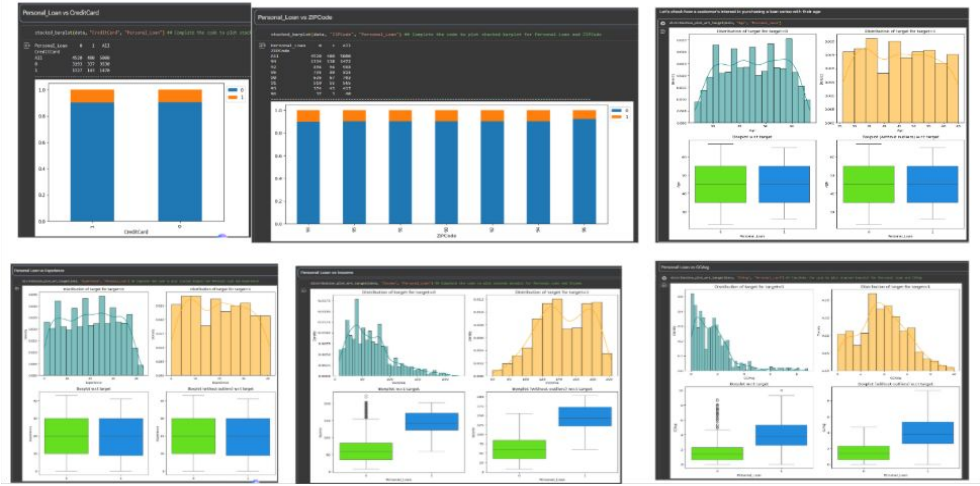


EDA Results - NoteBook

Notebook - EDA Results - Bivariate Analysis -1



Notebook - EDA Results - Bivariate Analysis -2



Data Preprocessing

Duplicate/Missing value check :

- No duplicate records found. Dataset is clean with unique customer entries.
- No missing values in the dataset.No imputation or deletion required.

Outlier Check & Treatment

- Used **IQR (Interquartile Range)** method on numeric columns (Income, CCAvg, Mortgage, Age, Experience).
- **Income, CCAvg, Mortgage** show right-skewed distributions with outliers (very high values).
- Outliers were not removed directly (since they may represent genuine high-value customers).
- Instead, models were tested with both raw data and capped versions (clipping beyond upper bound).
- Final model kept outliers because they reflect valuable loan prospects (e.g., high-income customers).

Feature Engineering

- Converted categorical variables into numeric/binary (if not already).
  - Education (already encoded: 1=Undergrad, 2=Graduate, 3=Advanced).
  - Securities_Account, CD_Account, Online, CreditCard are already 0/1.
- **No new derived variables** created, but flagged important features (Income per Family Member, Loan Ratio) for potential future modeling.

Data Preprocessing for Modeling

- **Train/Test Split:**
  - Dataset split into **70% training, 30% testing**.
- **Scaling:**
  - Decision Tree models do not require feature scaling (no normalization needed).
- **Class Imbalance:**
  - Target variable (Personal_Loan) had only ~9.6% positives.
  - Model evaluation focused on **Recall, Precision, and F1-score**, not just Accuracy.

- **Final Input Set:**
  - Predictor Variables: Age, Experience, Income, Family, CCAvg, Education, Mortgage, Securities_Account, CD_Account, Online, CreditCard.
  - Target Variable: Personal_Loan.

Outcome of Data Processing:

- Dataset is clean, no missing/duplicate issues.
- Outliers exist but represent valuable insights (retained).
- Dataset is ready for Decision Tree modeling with categorical and numeric features properly prepared.

Model Building

🌳 Model Building – Decision Tree

Step 1: Define Target & Predictors

- Target Variable: Personal_Loan (binary → accepted or not).
- Predictors: Age, Income, Family, CCAvg, Education, Mortgage, Securities_Account, CD_Account, Online, Credit Card.

Step 2: Train–Test Split

- Split dataset into 70% training and 30% test sets.

Step 3: Build Baseline Decision Tree

- Algorithm: DecisionTreeClassifier (Gini Index).
- Built without pruning (fully grown tree).
- Objective: establish baseline performance.
- Observation: achieved 100% accuracy on training but poor generalization (overfit).

Step 4: Model Evaluation

- Predictions generated on both train & test sets.
- Evaluated using:
  - Confusion Matrix → TP, TN, FP, FN.
  - Accuracy → overall correctness. Recall**,** Precision, F1-Score → more relevant due to class imbalance.
- **Performance:**
  - **Train → Accuracy = 1.0, Recall = 1.0, Precision = 1.0 ,F1 = 1.0**
  - **Test → Accuracy = 0.986 Recall = 0.933 Precision = 0.927 F1 = 0.930**
- Feature Importance → Income, Family, Education_2. Education_3, CCAvg

🔹 **Initial Decision Tree (Unpruned):**

- Strong performance on both train and test,
- but the tree was fully grown (complex) → **risk of overfitting** and less interpretability.

Model Building

Step 5: Pre-Pruning

Approach:

- Constrained tree during training by setting hyperparameters:
- max_depth=2, max_leaf_nodes=50, min_samples_split=10

Effect: Stopped the tree from growing too deep, preventing memorization of noise.

Performance:

- **Train → Accuracy = 0.790, Recall = 1.0, Precision = 0.310, F1 = 0.474**
- **Test → Accuracy = 0.771, Recall = 1.0, Precision = 0.310, F1 = 0.474**

Feature Importance → Income, CCAvg, Family

Conclusion:

- Restricted tree depth and splits.
- Achieved perfect recall (no missed positives) but very low precision (many false positives).
- Good for maximizing loan conversions but inefficient if cost of false positives is high.

🔹 ✅ **Business Conclusion:**
Post-pruning delivers the best model — with 97% accuracy on test data and balanced precision/recall — making it the recommended choice for AllLife Bank's marketing campaign.
The pruned decision tree strikes the best balance between **performance** and **interpretability**, making it suitable for guiding marketing campaigns.

Step 6: Post-Pruning (Cost-Complexity Pruning using CCP Alpha)

Approach:

- Built a fully grown tree, then applied Cost Complexity Pruning (CCP) with optimal ccp_alpha ≈ 0.000186.
- Removed weak/redundant branches after the tree was built.

Effect: Simplified tree, making it more interpretable.

Performance:

- **Train → Accuracy = 1.0, Recall = 1.0, Precision = 1.0 , F1 = 1.0**
- **Test → Accuracy = 0.97 Recall = 0.85 Precision = 0.93 F1 = 0.89**

Feature Importance & Decision Rules

- Top Features: Income,, Education_2, CCAvg, Education_3, Family
- Extracted simple, business-interpretable rules like:
  - If Income > 92.5K and CCAvg > 2.95 → Loan acceptance likely.
  - If Education = Advanced and CD_Account = 1 → Very high chance of acceptance.
  - If Income < 50K and CCAvg < 1.5 → Loan unlikely.

Conclusion:

- Achieved best balance between recall and precision.
- Recall = 0.85 → some positives are missed, but not too many.
- Precision = 0.93 → very few false positives, predictions are highly reliable.
- Overall F1 = 0.89, showing strong trade-off.

Model Performance Summary

Model Evaluation Criterion : Evaluated using:

- **Accuracy** → overall correctness.
- **Precision** → proportion of predicted positives that are correct.
- **Recall (Sensitivity)** → ability to capture all true positives (important for loan acceptance).
- **F1 Score** → harmonic mean of Precision & Recall, balances false positives and false negatives.
- Used **confusion matrix** to visualize classification errors.

Overview of Final Decision Tree Model

- Algorithm: DecisionTreeClassifier
- Splitting Criterion: Gini Index
- Final Parameters (Post-Pruning):
- ccp_alpha ≈ 0.000186
- max_depth determined automatically after pruning
- class_weight={0:0.15, 1:0.85} to handle class imbalance
- random_state=1 for reproducibility
- Interpretability: Tree rules easily explainable for business users.

Most Important Features

- Income – higher income customers more likely to accept personal loans.
- Education – Graduate/Professional customers more likely.
- CCAvg – higher credit card spending indicates higher probability.
- Family -
- CD_Account – customers with a CD account are highly probable loan acceptors.
- (Other features like Age, Mortgage, Securities_Account had less influence.)

Model Performance Summary

Model	Dataset	Accuracy	Recall	Precision	F1 Score
Base Model	Train	1.000	1.000	1.000	1.000
	Test	0.986	0.933	0.927	0.930
Pre-Pruned Model	Train	0.790	1.000	0.310	0.474
	Test	0.771	1.000	0.310	0.474
Post-Pruned Model (Final)	Train	1.000	1.000	1.000	1.000
	Test	0.970	0.850	0.930	0.890

Key Insights

- Base Model: Very strong performance but risk of overfitting (too complex).
- Pre-Pruned Model: High recall (captured all loan customers) but poor precision (many false positives).
- Post-Pruned Model (Final): Best balance – 97% test accuracy, high precision (0.93), good recall (0.85), and interpretable rules.

Model Performance Improvement

Effect of Pruning Techniques

- **Base Model (Unpruned)**
  - Very high accuracy on both train (1.0) and test (0.986).
  - Risk of overfitting due to full tree complexity.
  - Less interpretable for business use.
- **Pre-Pruning**
  - Introduced constraints (max_depth=2, max_leaf_nodes=50, min_samples_split=10).
  - Test Accuracy dropped to 0.79, Recall remained 1.0, Precision dropped to 0.31.
  - Improvement: Controlled overfitting; guaranteed recall (captured all loan acceptors).
  - Limitation: Too many false positives → lower campaign efficiency.
- **Post-Pruning (Final Model)**
  - Applied Cost Complexity Pruning (ccp_alpha ≈ 0.000186).
  - Train: Accuracy = 1.0, Recall = 1.0, Precision = 1.0, F1 = 1.0.
  - Test: Accuracy = 0.97, Recall = 0.85, Precision = 0.93, F1 = 0.89.
- **Improvement:**
  - Balanced Precision & Recall.
  - High accuracy on test data.
  - Simpler, interpretable rules → suitable for marketing decision-making.

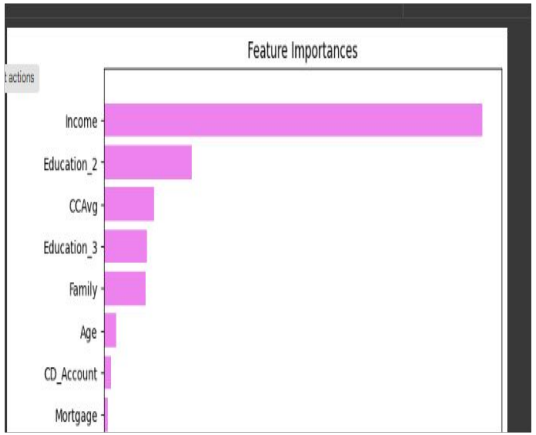
Decision Rules (Sample Extracts from Final Model)

- **If Income > 92.5K and CCAvg > 2.95 → Likely loan acceptance**
- **If Education = Professional/Advanced and CD_Account = 1 → Very high chance of acceptance**
- **If Income < 50K and CCAvg < 1.5 → Unlikely to accept loan**

Model Performance Improvement

🌳 Ranked Feature Importance – Post-Pruned Decision Tree

- Income – Strongest predictor of loan acceptance.
- Education_2 (Graduate) – Education level is a key factor; graduates are more likely.
- CCAvg (Credit Card Spending) – Higher spending correlates with higher likelihood of acceptance.
- Education_3 (Advanced/Professional) – Professional degrees increase probability.
- Family – Larger families show some influence.
- Age – Moderate predictor; certain age groups more likely.
- CD_Account – Customers with CD accounts are more probable loan takers.
-



Interpretation

- **Income + Education + CCAvg are the top three drivers.**
- **Family size and Age contribute moderately, capturing demographic influence.**
- **CD Account adds a financial relationship factor.**
- **Together, these features explain most of the decision splits, making the model interpretable for business targeting.**

Conclusion

- Pruning improved model performance by reducing overfitting and enhancing interpretability.
- Pre-Pruning: Useful when high recall is business priority (capture all potential customers).
- Post-Pruning (Final Model): Delivers best balance between accuracy, recall, and precision, while providing clear decision rules and feature importance insights.
- Recommendation: Adopt the post-pruned decision tree for campaign targeting.

APPENDIX

Data Background and Contents

Loading the dataset

```
# uncomment and run the below code snippets if the dataset is present in the Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

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

```
Loan = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Machine Learning/ML-Week4(WK8)-Personal Loan Campaign/Loan_Modelling.csv")  ##  Complete the code to read t
```

```
# copying data to another variable to avoid any changes to original data
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 of the data
```

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

Next steps: Generate code with data New interactive sheet

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

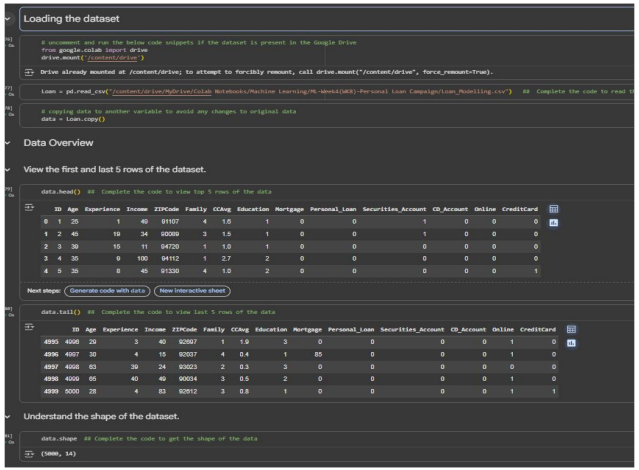
	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Account	CD_Account	Online	CreditCard
4995	4996	29	3	40	92697	1	1.9	3	0	0	0	0	1	0
4996	4997	30	4	15	92037	4	0.4	1	85	0	0	0	1	0
4997	4998	63	39	24	93023	2	0.3	3	0	0	0	0	0	0
4998	4999	65	40	49	90034	3	0.5	2	0	0	0	0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

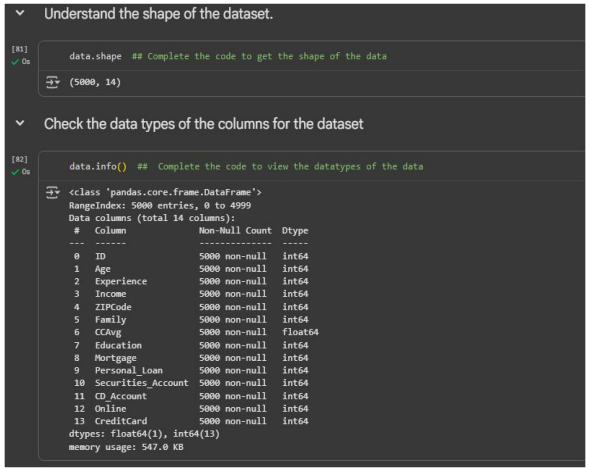
Understand the shape of the dataset.

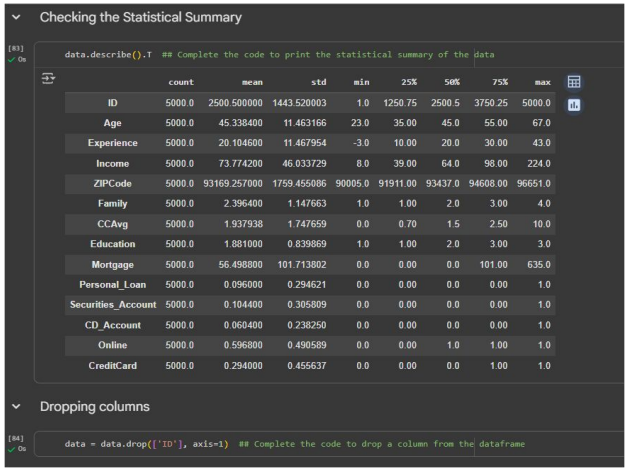
```
data.shape  ## Complete the code to get the shape of the data
```

```
(5000, 14)
```

Data Overview - Notebook







Notebook - Data Overview -1

Understand the shape of the dataset.

[81] ✓ 0s

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

```
(5000, 14)
```

Check the data types of the columns for the dataset

[82] ✓ 0s

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

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

Notebook - Data Overview -2

Checking the Statistical Summary

[83] ✓ 0s

```
data.describe().T  ## Complete the code to print the statistical summary of the data
```

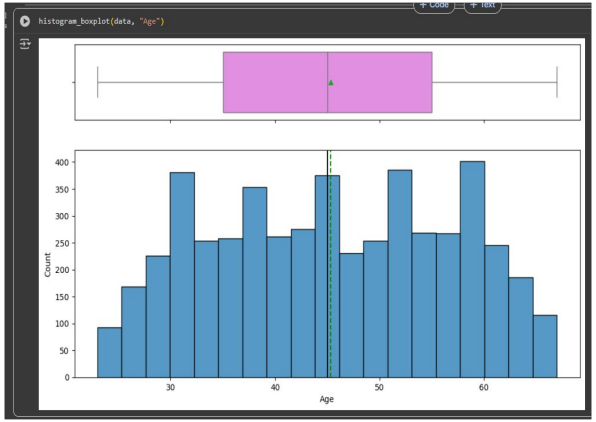
	count	mean	std	min	25%	50%	75%	max
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750.25	5000.0
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55.00	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30.00	43.0
Income	5000.0	73.774200	46.033729	8.0	39.00	64.0	98.00	224.0
ZIPCode	5000.0	93169.257000	1759.455086	90005.0	91911.00	93437.0	94608.00	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.70	1.5	2.50	10.0
Education	5000.0	1.881000	0.839869	1.0	1.00	2.0	3.00	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101.00	635.0
Personal_Loan	5000.0	0.096000	0.294621	0.0	0.00	0.0	0.00	1.0
Securities_Account	5000.0	0.104400	0.305809	0.0	0.00	0.0	0.00	1.0
CD_Account	5000.0	0.060400	0.238250	0.0	0.00	0.0	0.00	1.0
Online	5000.0	0.596800	0.490589	0.0	0.00	1.0	1.00	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1.00	1.0

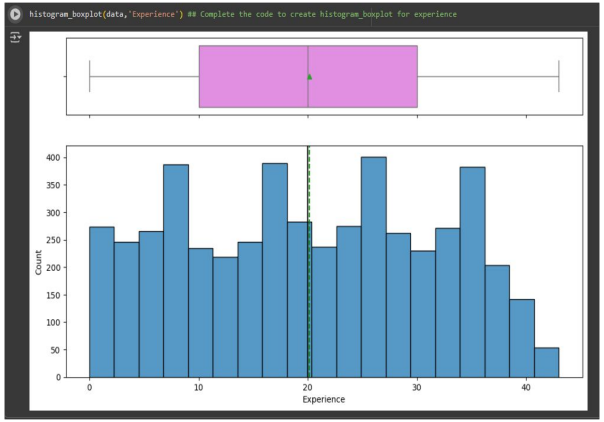
Dropping columns

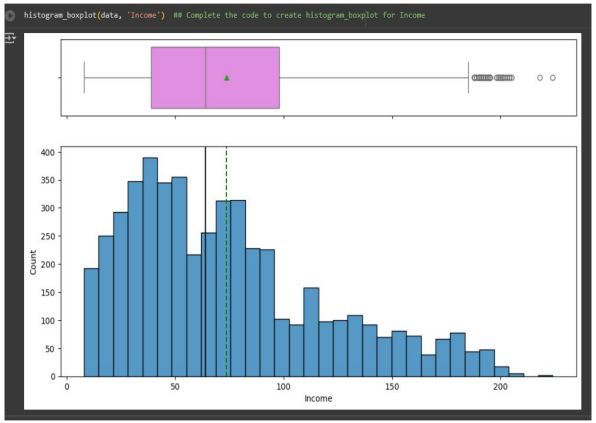
[84] ✓ 0s

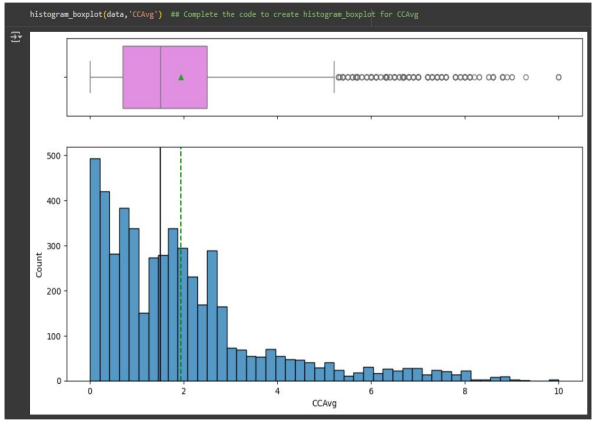
```
data = data.drop(['ID'], axis=1)  ## Complete the code to drop a column from the dataframe
```

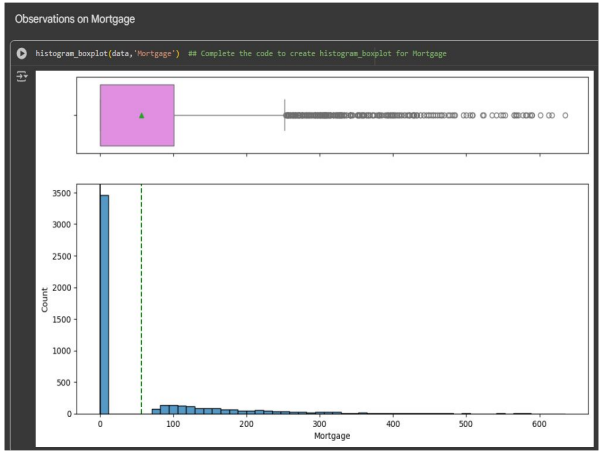
Notebook - EDA Results - Univariate Analysis -1

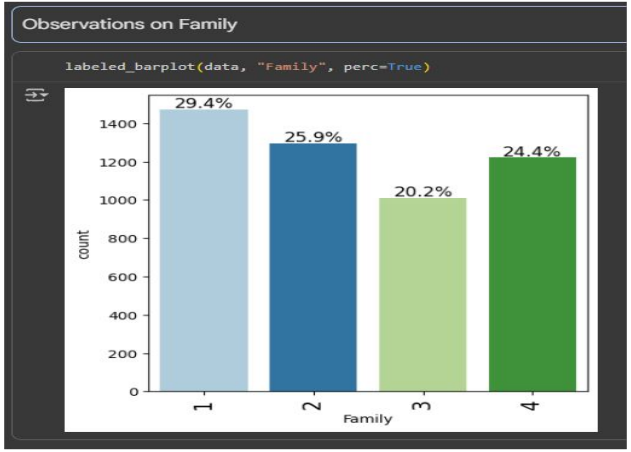








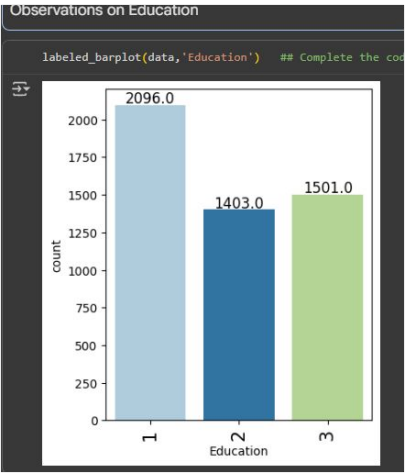




Notebook - EDA Results - Univariate Analysis -2

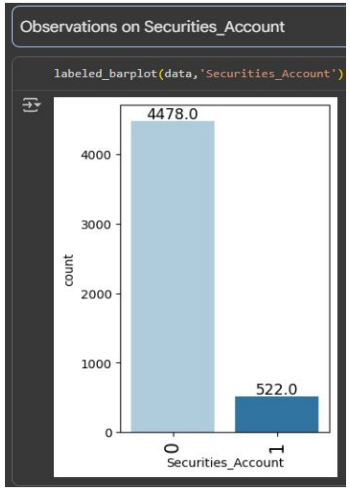
Observations on Education

```
labeled_barplot(data,'Education')   ## Complete the cod
```



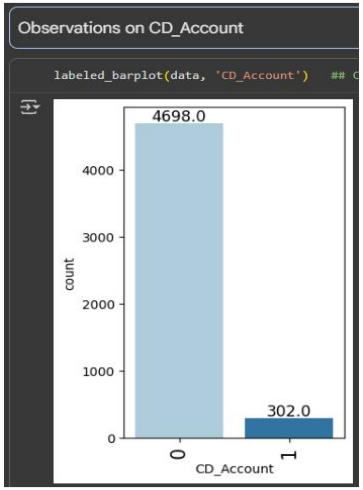
Observations on Securities_Account

```
labeled_barplot(data,'Securities_Account')
```



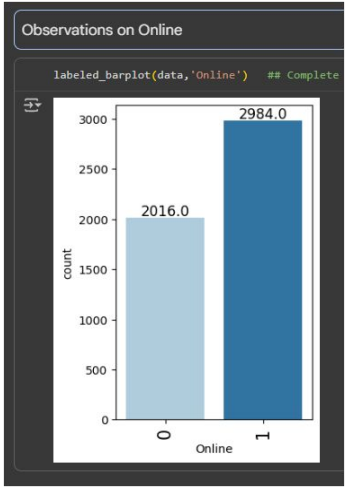
Observations on CD_Account

```
labeled_barplot(data, 'CD_Account')   ## C
```



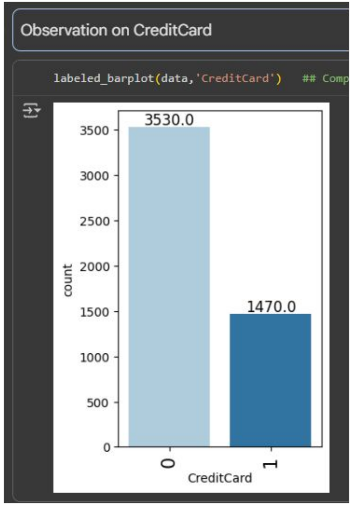
Observations on Online

```
labeled_barplot(data,'Online')   ## Complete
```



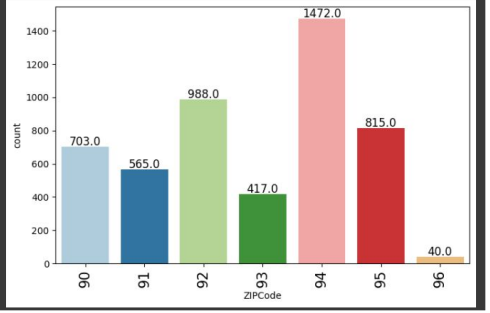
Observation on CreditCard

```
labeled_barplot(data,'CreditCard')   ## Comp
```

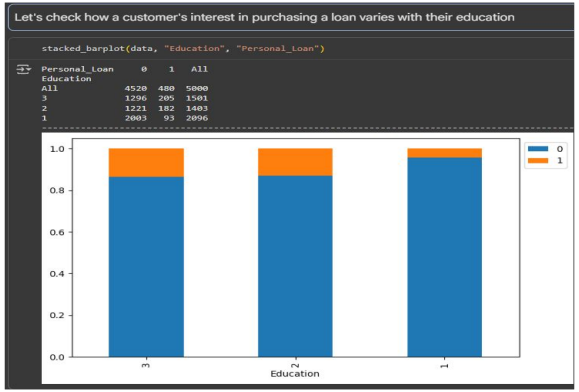


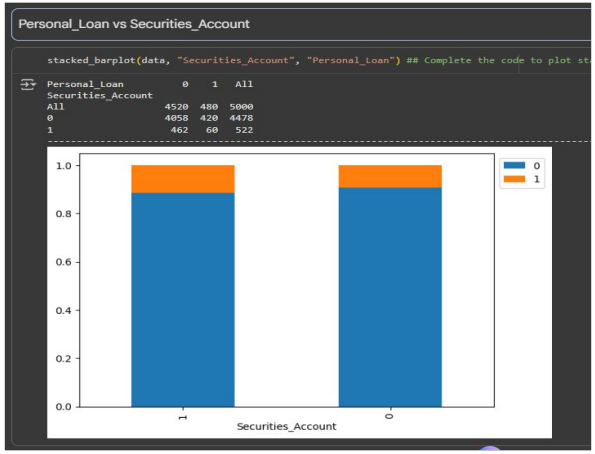
Observation on ZIPCode

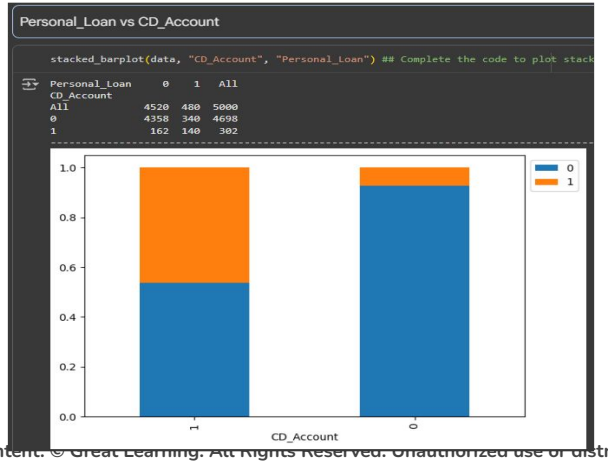
```
labeled_barplot(data,'ZIPCode')   ## Complete the code to create labeled_barplot for ZIPCode
```

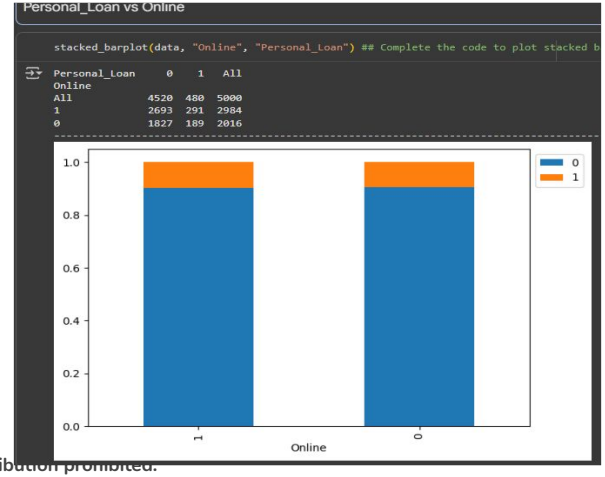


Notebook - EDA Results - Bivariate Analysis -1

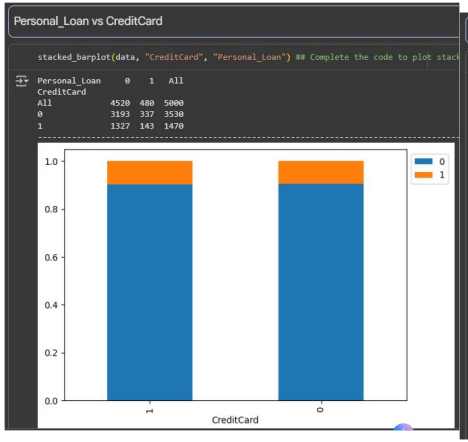


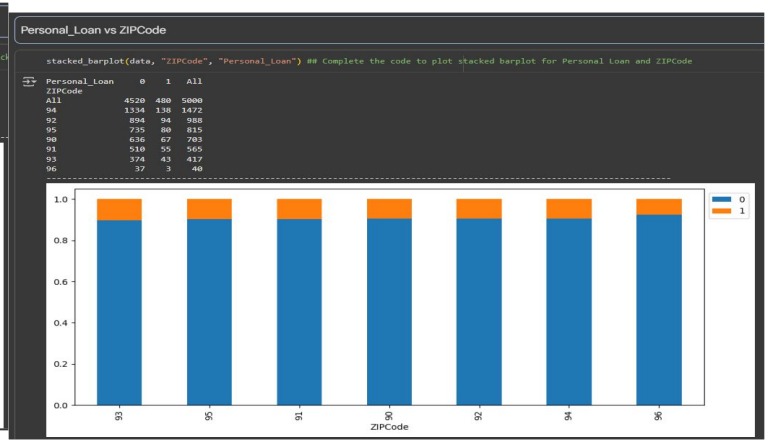


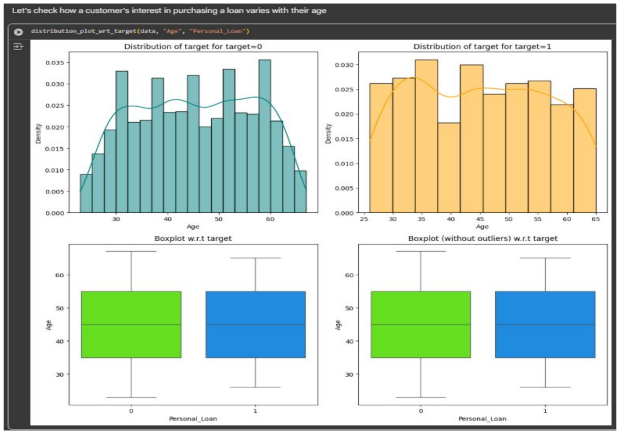


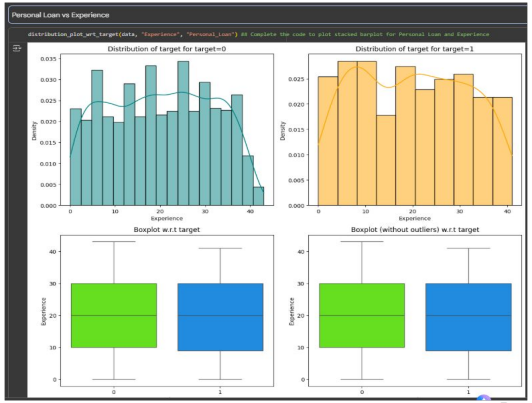


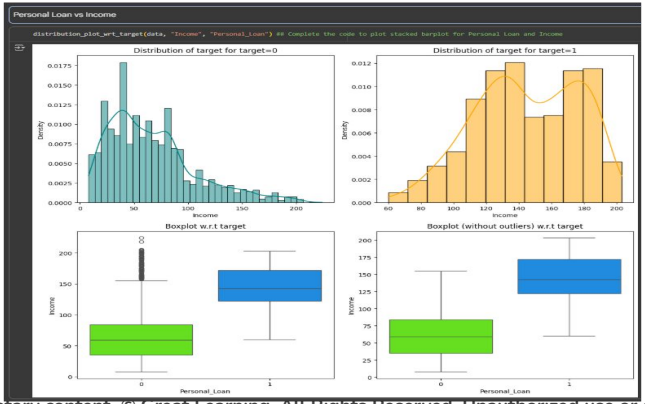
Notebook - EDA Results - Bivariate Analysis -2

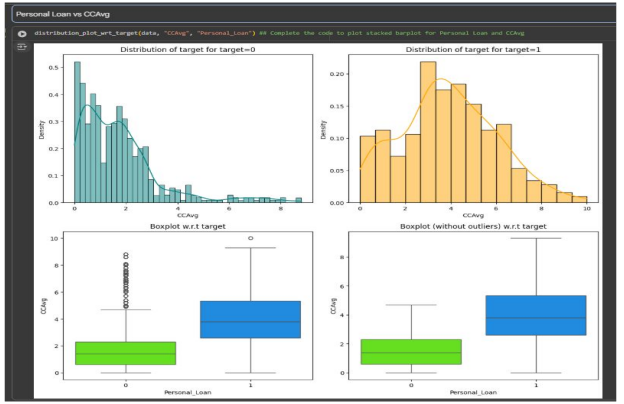




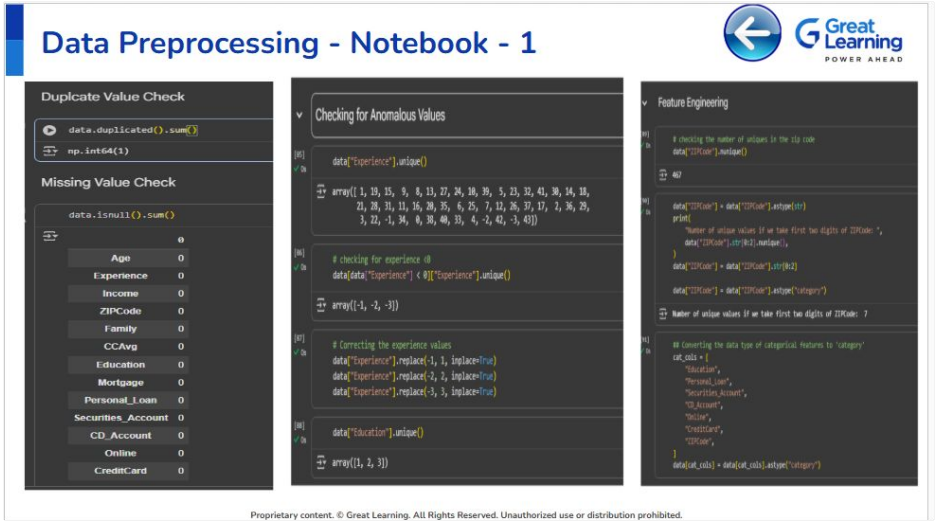


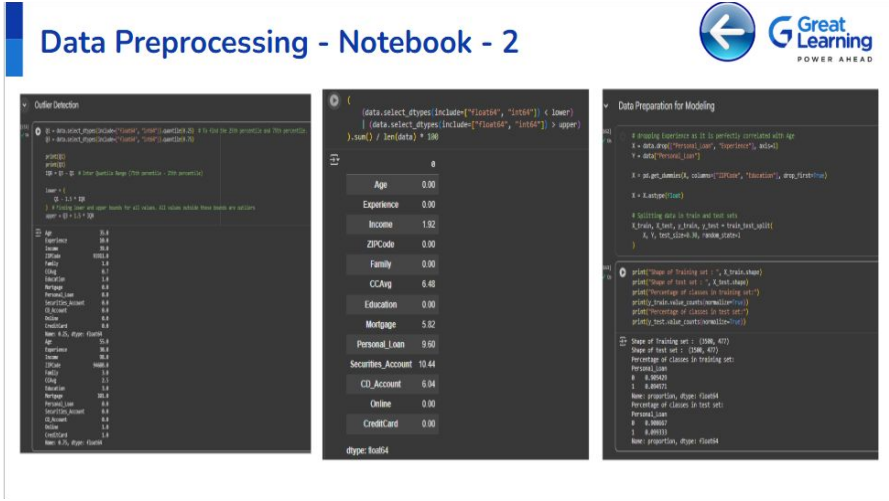






Data Preprocessing - Notebook





Data Preprocessing - Notebook - 1

Duplcate Value Check

```
data.duplicated().sum()
```

```
np.int64(1)
```

Missing Value Check

```
data.isnull().sum()
```

	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

Checking for Anomalous Values

[85]
```
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])
```

[86]
```
# checking for experience <0
data[data["Experience"] < 0]["Experience"].unique()
```

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

[87]
```
# 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)
```

[88]
```
data["Education"].unique()
```

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

Feature Engineering

[89]
```
# checking the number of uniques in the zip code
data["ZIPCode"].nunique()
```

```
467
```

[90]
```
data["ZIPCode"] = data["ZIPCode"].astype(str)
print(
    "Number of unique values if we take first two digits of ZIPCode: ",
    data["ZIPCode"].str[0:2].nunique(),
)
data["ZIPCode"] = data["ZIPCode"].str[0:2]

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

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

[91]
```
## Converting the data type of categorical features to 'category'
cat_cols = [
    "Education",
    "Personal_Loan",
    "Securities_Account",
    "CD_Account",
    "Online",
    "CreditCard",
    "ZIPCode",
]
data[cat_cols] = data[cat_cols].astype("category")
```

Data Preprocessing - Notebook - 2

Outlier Detection

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

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

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

```
Age                 35.0
Experience          10.0
Income              39.0
ZIPCode          91911.0
Family              1.0
CCAvg               0.7
Education           1.0
Mortgage            0.0
Personal_Loan       0.0
Securities_Account  0.0
CD_Account          0.0
Online              0.0
CreditCard          0.0
Name: 0.25, dtype: float64
Age                 55.0
Experience          30.0
Income              98.0
ZIPCode          94608.0
Family              3.0
CCAvg               2.5
Education           3.0
Mortgage          101.0
Personal_Loan       0.0
Securities_Account  0.0
CD_Account          0.0
Online              1.0
CreditCard          1.0
Name: 0.75, dtype: float64
```

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

	0
Age	0.00
Experience	0.00
Income	1.92
ZIPCode	0.00
Family	0.00
CCAvg	6.48
Education	0.00
Mortgage	5.82
Personal_Loan	9.60
Securities_Account	10.44
CD_Account	6.04
Online	0.00
CreditCard	0.00

dtype: float64

Data Preparation for Modeling

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

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

X = X.astype(float)

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

```
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, 477)
Shape of test set :  (1500, 477)
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
```

Pre Pruning - Notebook - 1

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 its performance
best_estimator = None
best_score_diff = float('inf')
best_test_score = 0.0

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

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

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

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

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

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

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

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

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

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

```
DecisionTreeClassifier(class_weight='balanced', max_depth=np.int64(2),
                       max_leaf_nodes=50, min_samples_split=10,
                       random_state=42)
```

Checking performance on training data

```
confusion_matrix_sklearn(model, X_train, y_train) ## Complete the code to create confusion matri
```

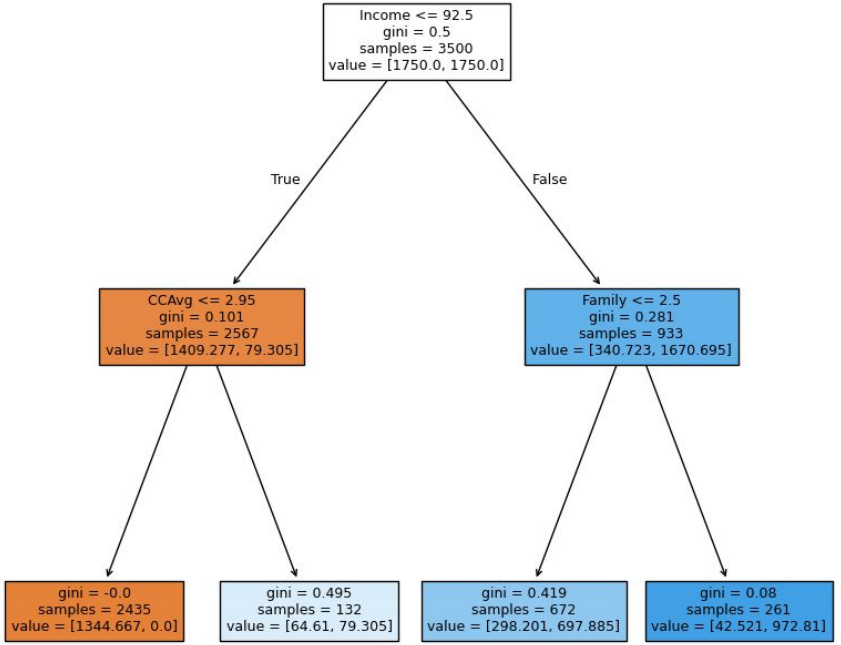
Pre Pruning - Notebook - 2

```
decision_tree_tune_perf_train = model_performance_classification_sklearn(model, X_train, y
decision_tree_tune_perf_train
```

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

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



Pre Pruning - Notebook - 3

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

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

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

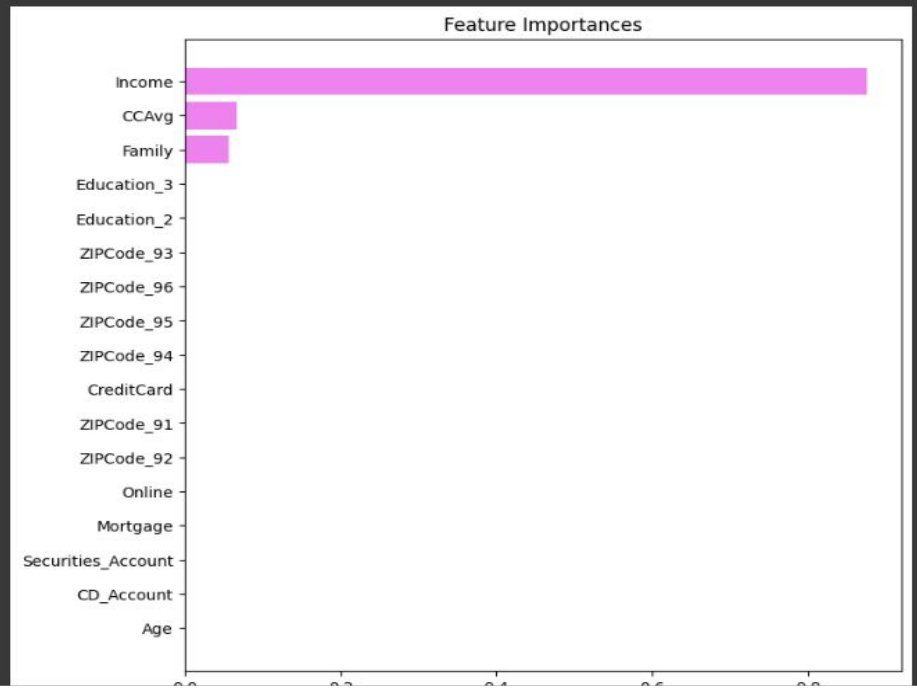
```
# importance of features in the tree building ( The importance of a feature is computed as the
# (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance )

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

	Imp
Income	0.876529
CCAvg	0.066940
Family	0.056531
Age	0.000000
Mortgage	0.000000
Securities_Account	0.000000
CD_Account	0.000000
Online	0.000000
CreditCard	0.000000
ZIPCode_91	0.000000
ZIPCode_92	0.000000
ZIPCode_93	0.000000
ZIPCode_94	0.000000
ZIPCode_95	0.000000
ZIPCode_96	0.000000
Education_2	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], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```

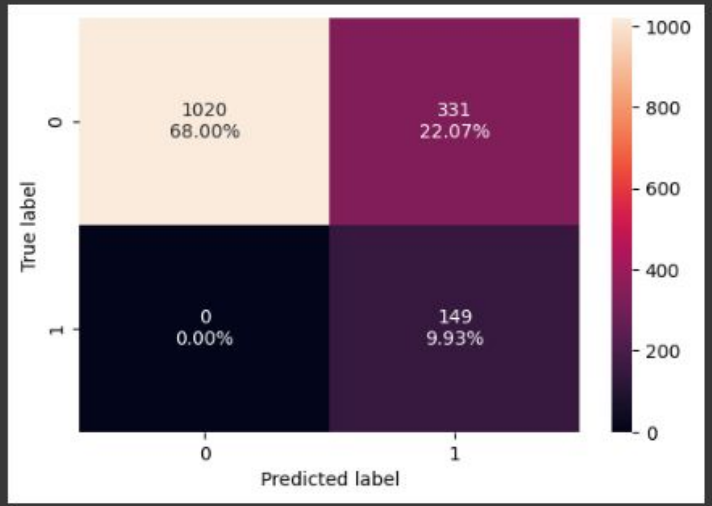


Pre Pruning - Notebook - 4

Great Learning
POWER AHEAD

Checking performance on test data

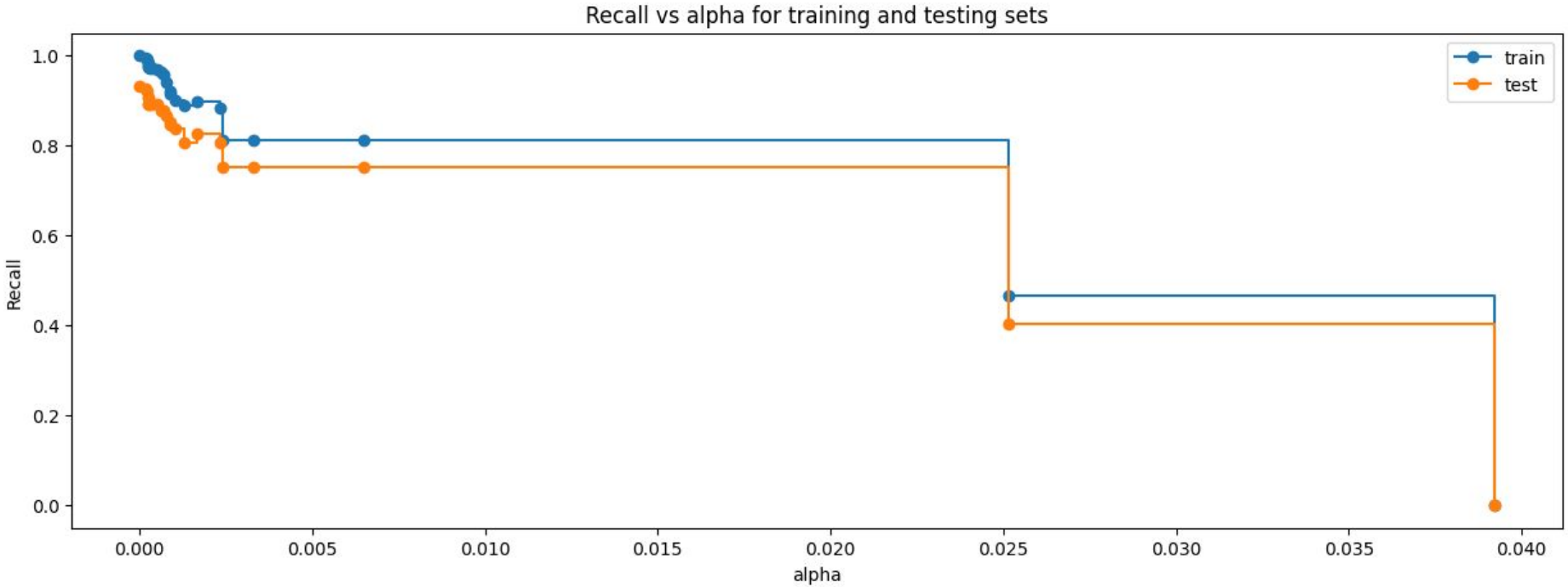
```
confusion_matrix_sklearn(estimator, X_test, y_test)  # Complete the code to get the confusion matrix on test data
```




```
decision_tree_tune_perf_test = model_performance_classification_sklearn(estimator, X_test, y_test) ## Complete the code to check performance on test data
decision_tree_tune_perf_test
```

	Accuracy	Recall	Precision	F1
0	0.779333	1.0	0.310417	0.473768

Post Pruning - Notebook - 1

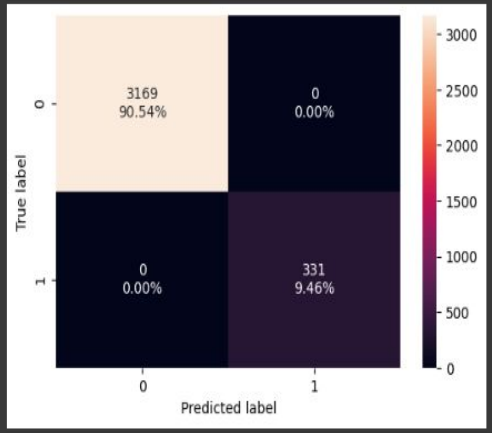


Post Pruning - Notebook - 2

```
DecisionTreeClassifier(ccp_alpha=np.float64(0.0),
                       class_weight={0: 0.15, 1: 0.85}, random_state=1)
```

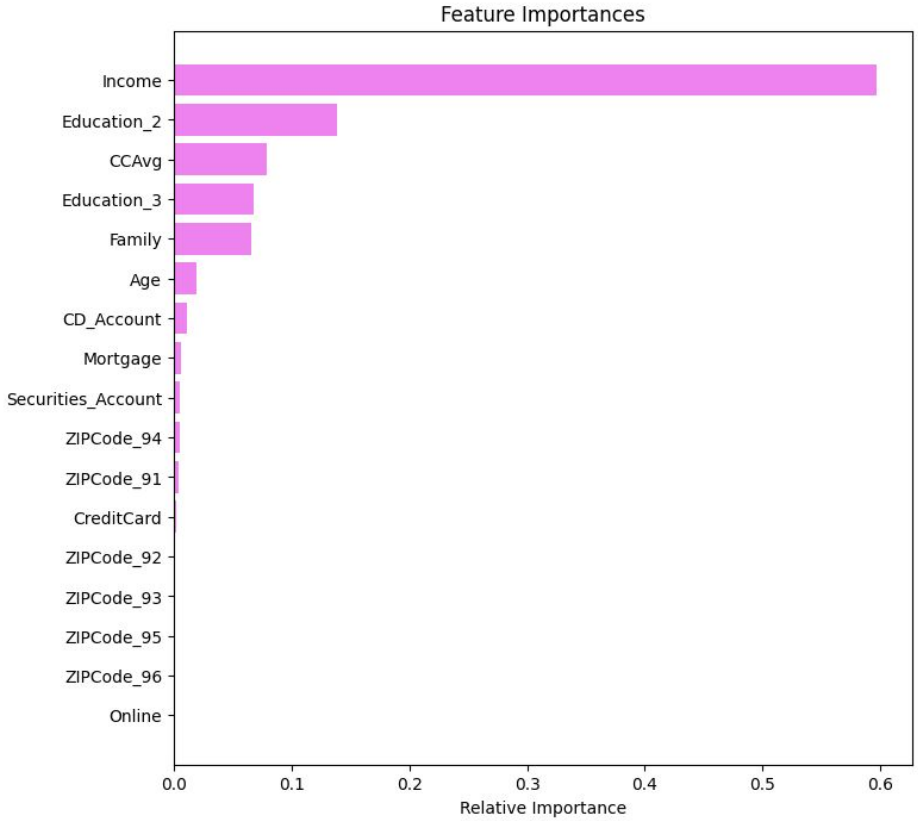
Checking performance on training data

```
confusion_matrix_sklearn(estimator_2,X_train, y_train) ## Complete the code to create confusion matrix for train data
```

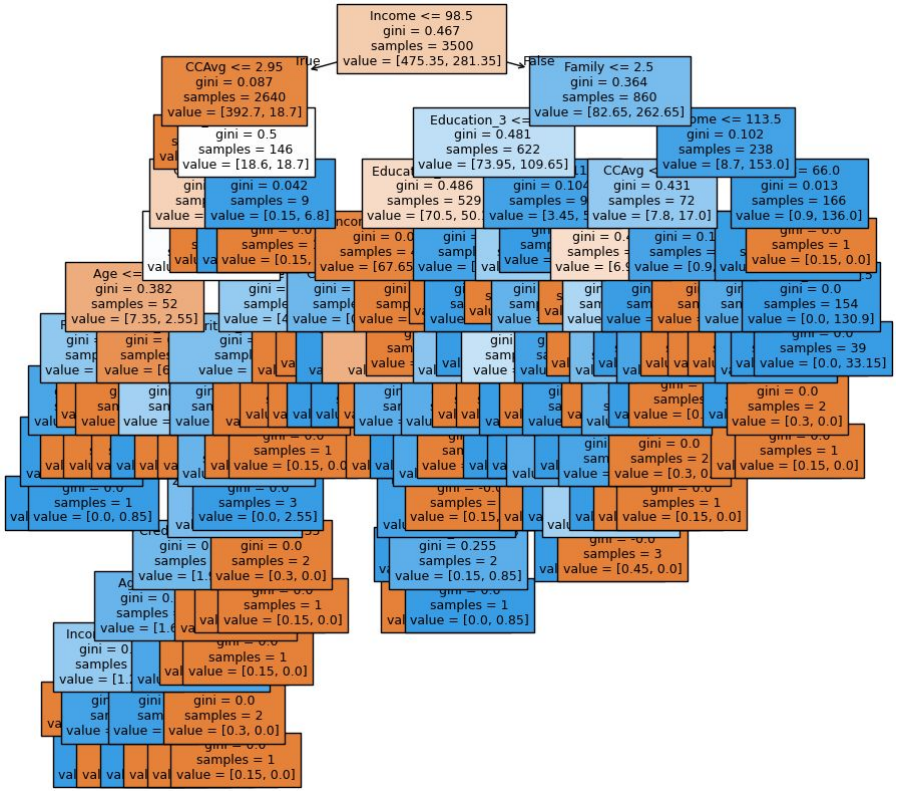



```
decision_tree_tune_post_train = model_performance_classification_sklearn(estimator_2,X_train, y_train) ## Complete the
decision_tree_tune_post_train
```

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

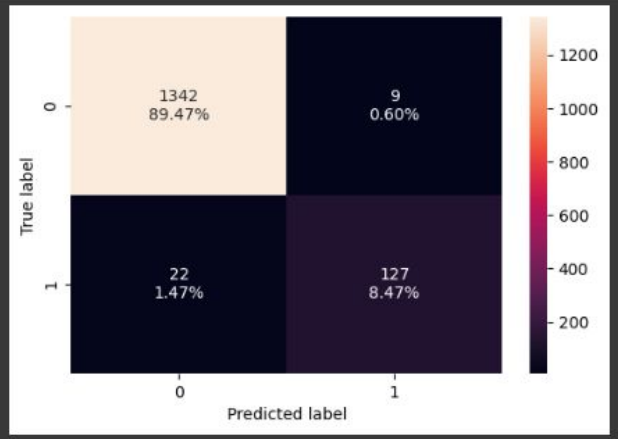


Post Pruning - Notebook - 3



Checking performance on test data

```
confusion_matrix_sklearn(estimator_2,X_test, y_test)  # Complete the code to get the confusion matrix on
```




```
decision_tree_tune_post_test = model_performance_classification_sklearn(estimator_2,X_test, y_test) ## C
decision_tree_tune_post_test
```

	Accuracy	Recall	Precision	F1
0	0.979333	0.852349	0.933824	0.891228

Plot rec

Post Pruning - Notebook - 4

Model Performance Comparison and Final Model Selection

```
# training performance comparison

models_train_comp_df = pd.concat(
    [decision_tree_perf_train.T, decision_tree_tune_perf_train.T, decision_tree_tune_post_train.T], axis=1,
)
models_train_comp_df.columns = ["Decision Tree (sklearn default)", "Decision Tree (Pre-Pruning)", "Decision Tree (Post-
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	1.0
Recall	1.0	1.000000	1.0
Precision	1.0	0.310798	1.0
F1	1.0	0.474212	1.0

Next steps: Generate code with models_train_comp_df | New interactive sheet

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [decision_tree_perf_test.T, decision_tree_tune_perf_test.T, decision_tree_tune_post_test.T], axis=1,
)
models_test_comp_df.columns = ["Decision Tree (sklearn default)", "Decision Tree (Pre-Pruning)", "Decision Tree (Post-
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	0.979333
Recall	0.932886	1.000000	0.852349
Precision	0.926667	0.310417	0.933824
F1	0.929766	0.473768	0.89

Plot recall_train and reca

Next steps: Generate code with models_test_comp_df | New interactive sheet

Great Learning

Happy Learning !