# 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.

## 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)

# Importing necessary libraries

In [4]:
```
# Installing the libraries with the specified version.
!pip install numpy==1.25.2 pandas==1.5.3 matplotlib==3.7.1 seaborn==0.13.1 s
```

**Note**:

1. After running the above cell, kindly restart the notebook kernel (for Jupyter Notebook) or runtime (for Google Colab), write the relevant code for the project from the next cell, 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.

In [5]:
```
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

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

# Library to split data
from sklearn.model_selection import train_test_split

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

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

# to suppress unnecessary warnings
```

```
import warnings
warnings.filterwarnings("ignore")
```

# Loading the dataset

In [6]:
```
# uncomment the following lines if Google Colab is being used
from google.colab import drive
drive.mount('/content/drive')
Loan = pd.read_csv('/content/drive/My Drive/Loan_Modelling.csv')
```

```
Mounted at /content/drive
```

# Data Overview

- Observations
- Sanity checks

In [7]:
```
# Display the first 5 rows of the dataset
Loan.head()
```

Out[7]:

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

In [8]:
```
# Display the last 5 rows of the dataset
Loan.tail()
```

Out[8]:

| | ID | Age | Experience | Income | ZIPCode | Family | CCAvg | Education | Mortgag |
|---|---|---|---|---|---|---|---|---|---|
| **4995** | 4996 | 29 | 3 | 40 | 92697 | 1 | 1.9 | 3 | |
| **4996** | 4997 | 30 | 4 | 15 | 92037 | 4 | 0.4 | 1 | 8 |
| **4997** | 4998 | 63 | 39 | 24 | 93023 | 2 | 0.3 | 3 | |
| **4998** | 4999 | 65 | 40 | 49 | 90034 | 3 | 0.5 | 2 | |
| **4999** | 5000 | 28 | 4 | 83 | 92612 | 3 | 0.8 | 1 | |

In [9]:
```
# Understand the shape of the dataset
Loan.shape
```

Out[9]:
```
(5000, 14)
```

In [10]:
```
# Check the data types of the columns for the dataset.
Loan.dtypes
```

Out[10]:

| | 0 |
|---|---|
| **ID** | int64 |
| **Age** | int64 |
| **Experience** | int64 |
| **Income** | int64 |
| **ZIPCode** | int64 |
| **Family** | int64 |
| **CCAvg** | float64 |
| **Education** | int64 |
| **Mortgage** | int64 |
| **Personal_Loan** | int64 |
| **Securities_Account** | int64 |
| **CD_Account** | int64 |
| **Online** | int64 |
| **CreditCard** | int64 |

**dtype:** object

In [11]:
```
# Checking the Statistical Summary
Loan.describe()
```

Out[11]:

| | ID | Age | Experience | Income | ZIPCode | F |
|---|---|---|---|---|---|---|
| **count** | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.00 |
| **mean** | 2500.500000 | 45.338400 | 20.104600 | 73.774200 | 93169.257000 | 2.39 |
| **std** | 1443.520003 | 11.463166 | 11.467954 | 46.033729 | 1759.455086 | 1.14 |
| **min** | 1.000000 | 23.000000 | -3.000000 | 8.000000 | 90005.000000 | 1.00 |
| **25%** | 1250.750000 | 35.000000 | 10.000000 | 39.000000 | 91911.000000 | 1.00 |
| **50%** | 2500.500000 | 45.000000 | 20.000000 | 64.000000 | 93437.000000 | 2.00 |
| **75%** | 3750.250000 | 55.000000 | 30.000000 | 98.000000 | 94608.000000 | 3.00 |
| **max** | 5000.000000 | 67.000000 | 43.000000 | 224.000000 | 96651.000000 | 4.00 |

Numeric features like Income, CCAvg, and Mortgage show a wide range

Mean Income: ~73k; Max Income: 224k

Mean CCAvg: ~1.94; Max CCAvg: 10

Mortgage values go up to 635k, indicating a mix of customer financial backgrounds

In [12]:
```
# What were the results of the campaign
loan_dist = Loan['Personal_Loan'].value_counts(normalize=True) # Calling val
print("Class distribution for Personal_Loan:")
print(loan_dist)
```

```
Class distribution for Personal_Loan:
Personal_Loan
0    0.904
1    0.096
Name: proportion, dtype: float64
```

90.4% of customers did not accept the loan (0)

9.6% of customers did accept the loan (1)

In [13]:
```
# Check for negative values in numeric columns
negatives = Loan[['Age', 'Experience', 'Income', 'Mortgage']] < 0
print("Count of negative values:")
print(negatives.sum())
```

```
Count of negative values:
Age            0
Experience    52
Income         0
Mortgage       0
dtype: int64
```

52 negative values found in Experience – likely due to data entry errors. These 52 records represent 1% of all of the data. I'm going to replace these values with the median value (see below).

In [14]:
```
# Check for duplicate IDs
duplicate_ids = Loan['ID'].duplicated().sum() # Call duplicated() on the 'ID
print(f"Number of duplicate IDs: {duplicate_ids}")
```

```
Number of duplicate IDs: 0
```

No duplicate IDs found

In [15]:
```
# Unique values in categorical columns
categorical_cols = ['Education', 'Securities_Account', 'CD_Account', 'Online
for col in categorical_cols:
    print(f"Unique values in {col}: {Loan[col].unique()}")
```

```
Unique values in Education: [1 2 3]
Unique values in Securities_Account: [1 0]
Unique values in CD_Account: [0 1]
Unique values in Online: [0 1]
Unique values in CreditCard: [0 1]
```

Categorical variables contain valid values:

Education: [1, 2, 3] (likely tiers such as High School, Undergrad, Graduate)

Family: [1, 2, 3, 4] (household size)

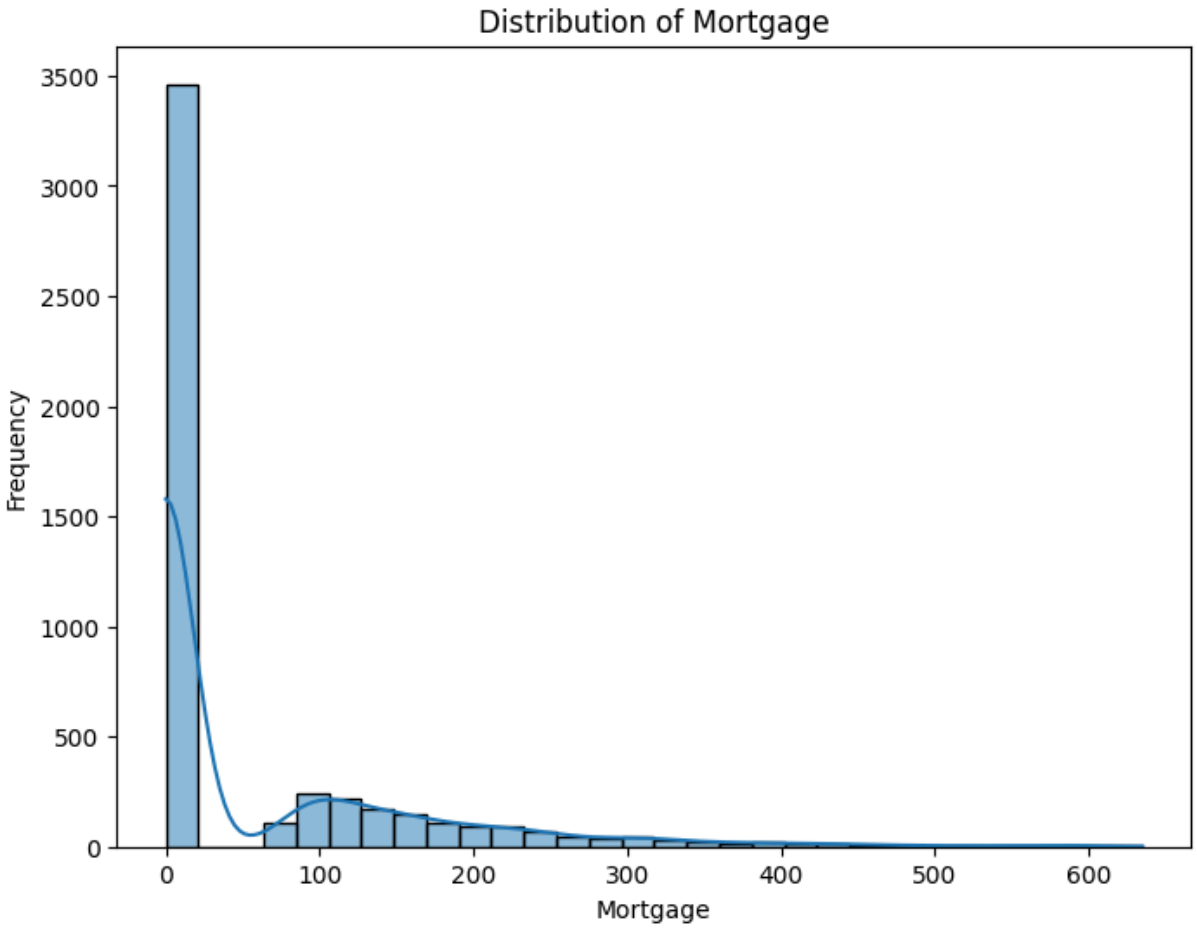
# Exploratory Data Analysis.

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

**Questions**:

1. What is the distribution of mortgage attribute? Are there any noticeable patterns or outliers in the distribution?
2. How many customers have credit cards?
3. What are the attributes that have a strong correlation with the target attribute (personal loan)?
4. How does a customer's interest in purchasing a loan vary with their age?
5. How does a customer's interest in purchasing a loan vary with their education?

In [16]:
```
# What is the distribution of mortgage attribute? Are there any noticeable p
plt.figure(figsize=(8, 6))
sns.histplot(Loan['Mortgage'], bins=30, kde=True)
plt.title('Distribution of Mortgage')
plt.xlabel('Mortgage')
plt.ylabel('Frequency')
plt.show()
```



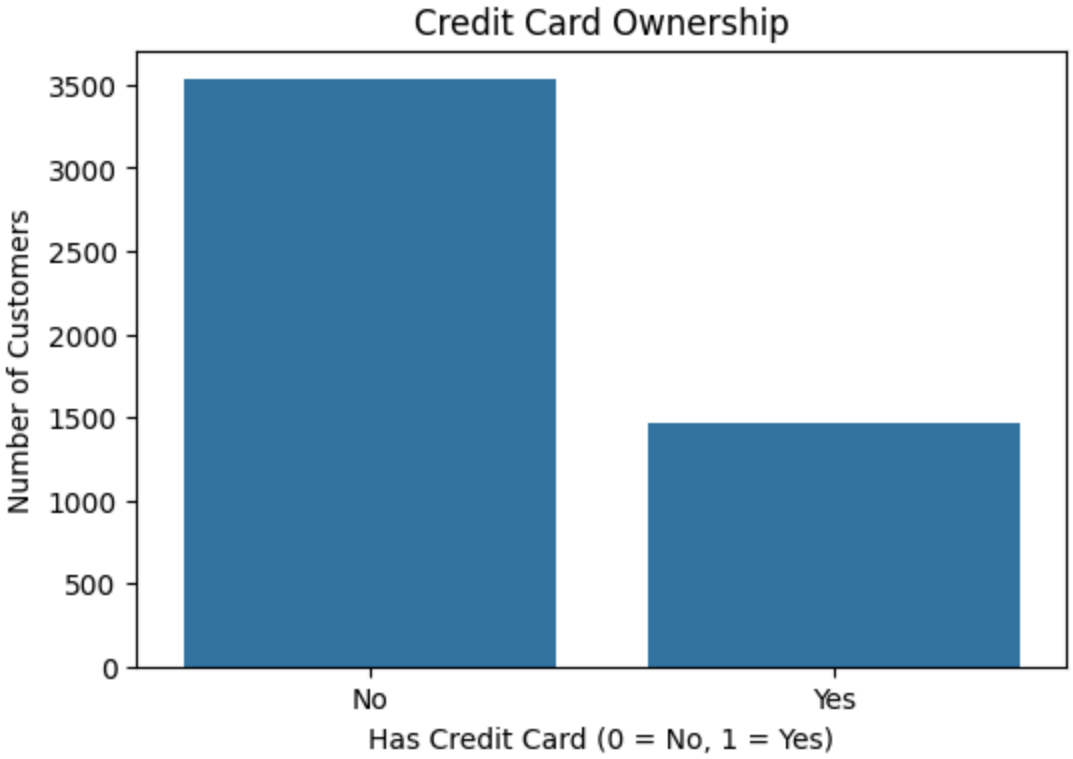

**Observations:**

Most customers have a mortgage value of $0, with a sharp drop-off beyond that.

There is a long tail of higher values, including some notable outliers.

Any mortgage value significantly above $240k is considered an outlier (based on the IQR method).

In [17]:
```
# How many customers have credit cards?
plt.figure(figsize=(6, 4))
sns.countplot(x='CreditCard', data=Loan) # Changed 'df' to 'Loan'
plt.title('Credit Card Ownership')
plt.xlabel('Has Credit Card (0 = No, 1 = Yes)')
plt.ylabel('Number of Customers')
plt.xticks([0, 1], ['No', 'Yes'])
plt.show()
```



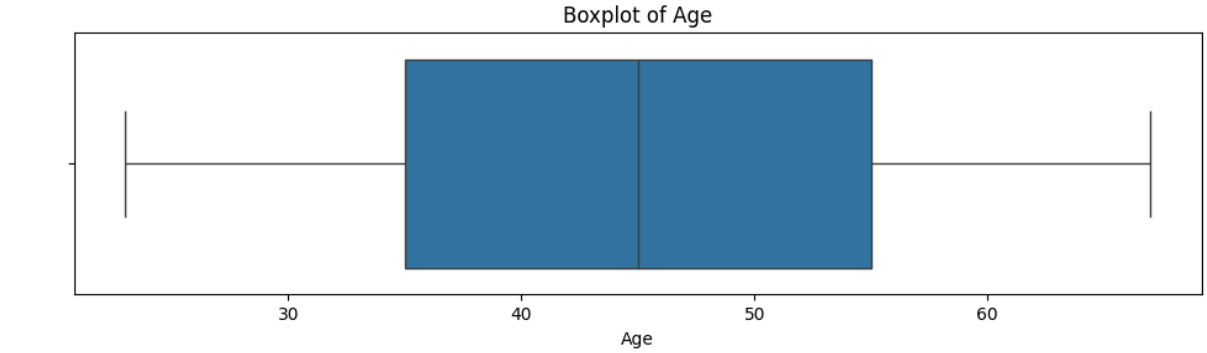

**Observations:**

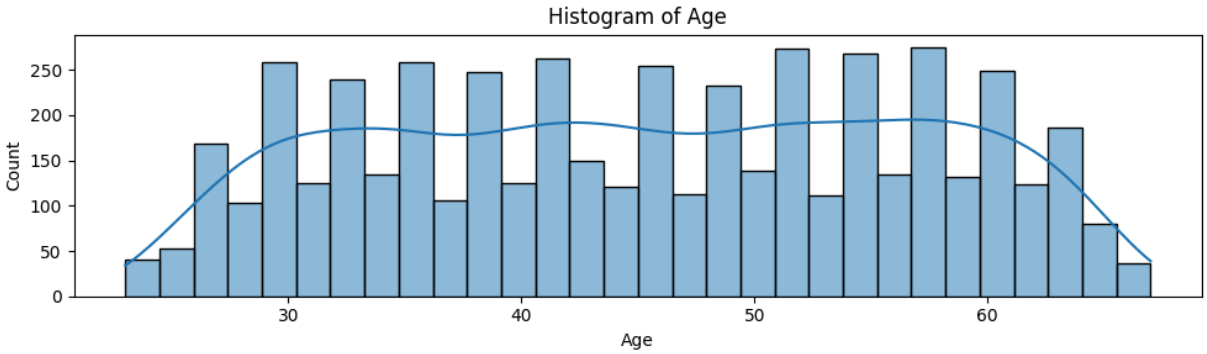
1,470 customers have a credit card

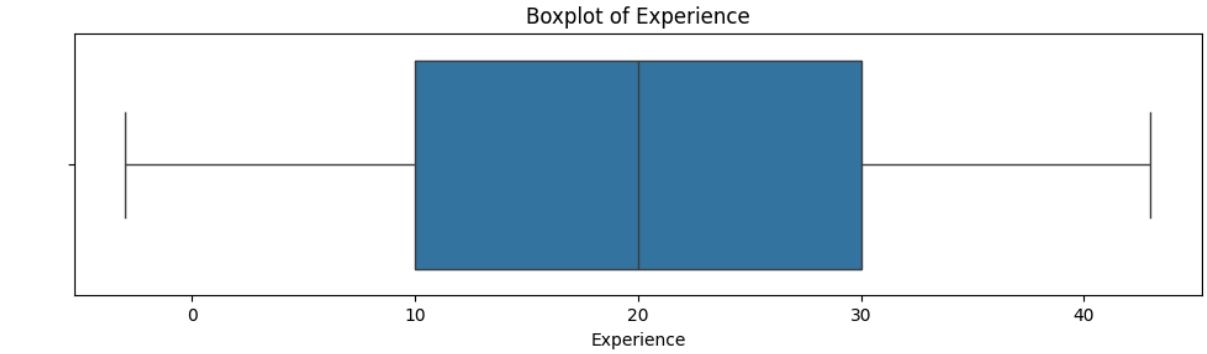
3,530 customers do not

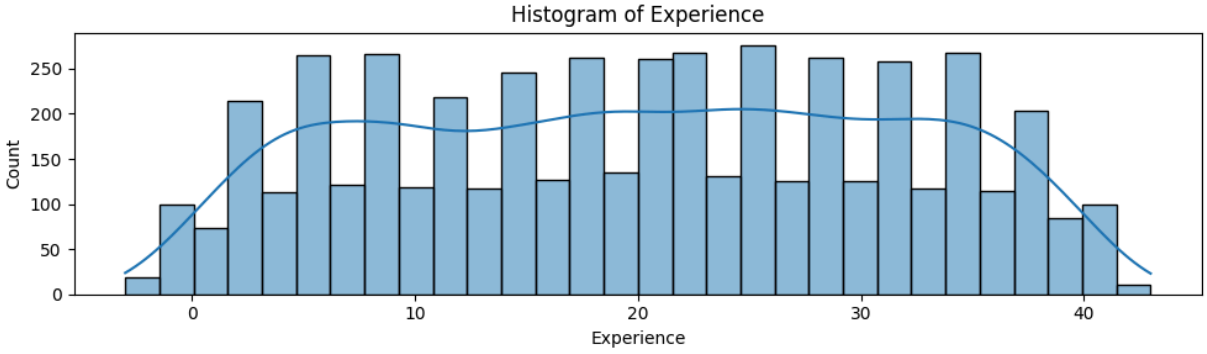
That means 29.4% of customers have a credit card

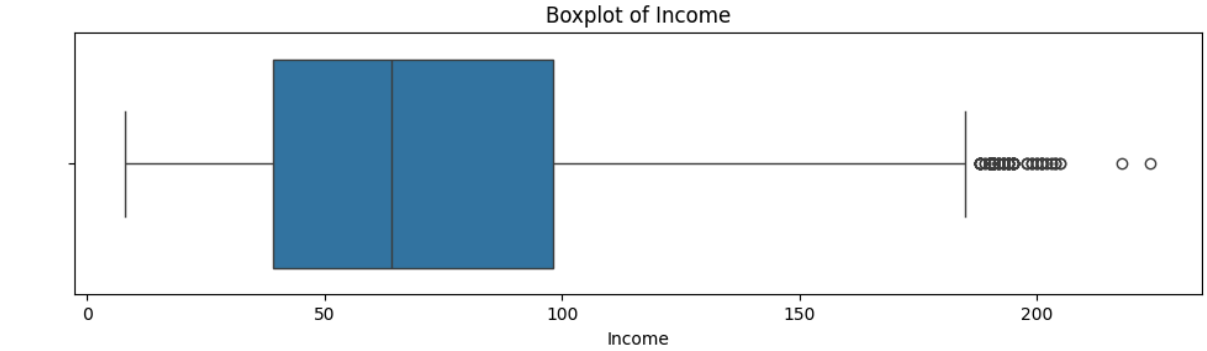
In [18]:
```
# Plot histogram + boxplot for each variable
variables = ['Age', 'Experience', 'Income', 'CCAvg', 'Mortgage', 'Education'
for var in variables:
    plt.figure(figsize=(10, 6))

    # Boxplot
    plt.subplot(2, 1, 1)
    sns.boxplot(x=Loan[var]) # Changed 'df' to 'Loan'
    plt.title(f'Boxplot of {var}')

    # Histogram
    plt.subplot(2, 1, 2)
    bins = 30 if Loan[var].nunique() > 10 else Loan[var].nunique() # Changed
    sns.histplot(Loan[var], bins=bins, kde=True if Loan[var].nunique() > 10
    plt.title(f'Histogram of {var}')
    plt.xlabel(var)

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

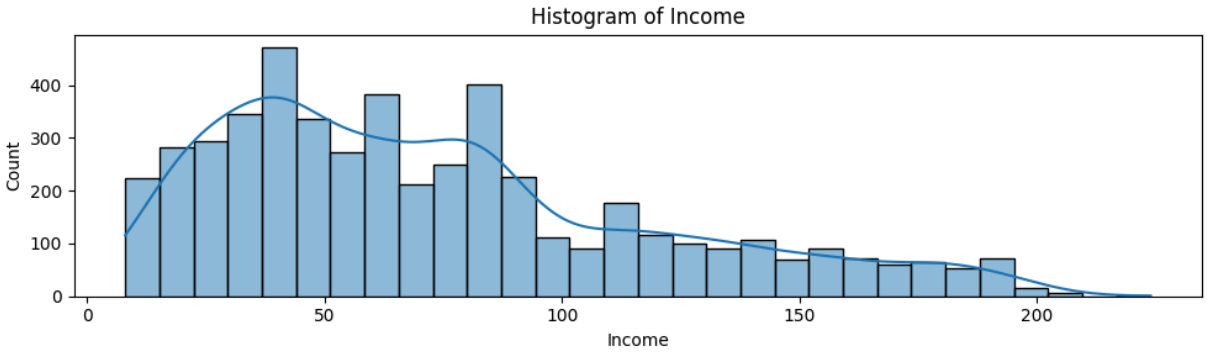

Boxplot of Age
30
40
50
60
Age

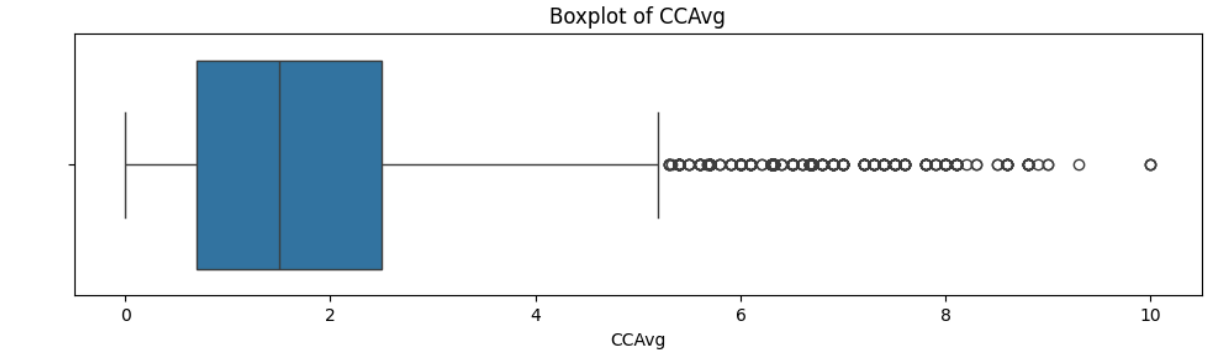

Histogram of Age
Count
0
50
100
150
200
250
30
40
50
60
Age

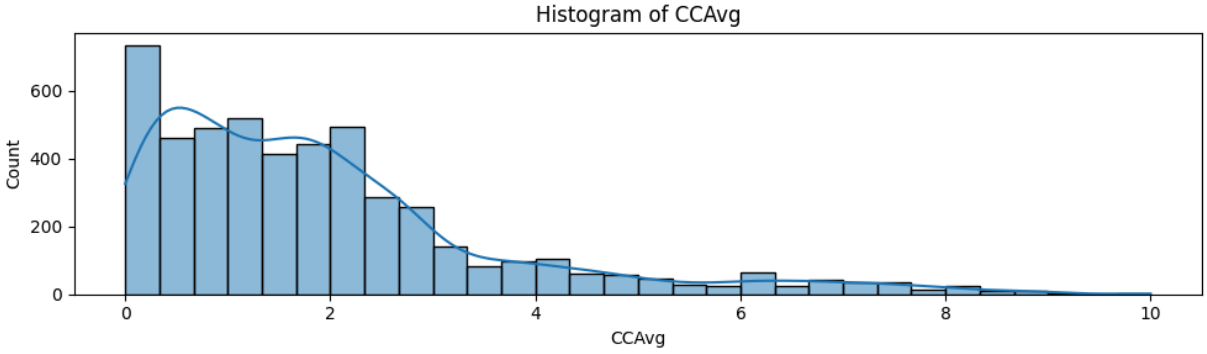

Boxplot of Experience
0
10
20
30
40
Experience

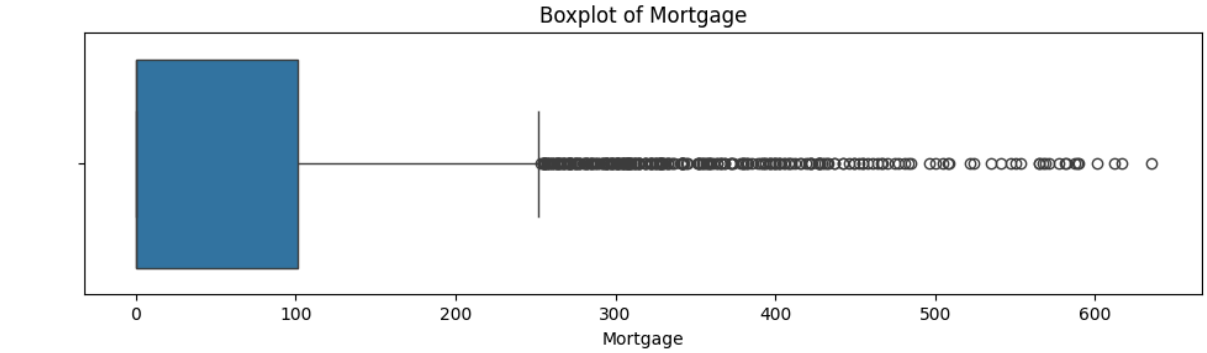

Histogram of Experience
Count
0
50
100
150
200
250
0
10
20
30
40
Experience

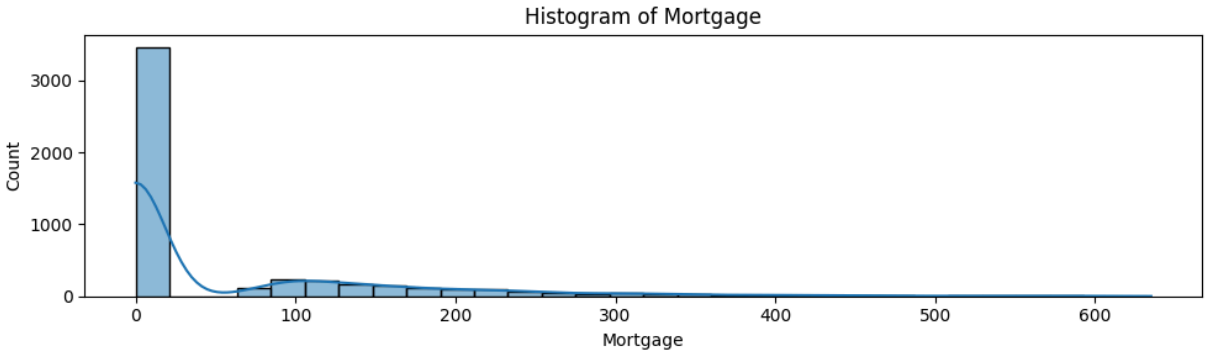

Boxplot of Income
0
50
100
150
200
Income

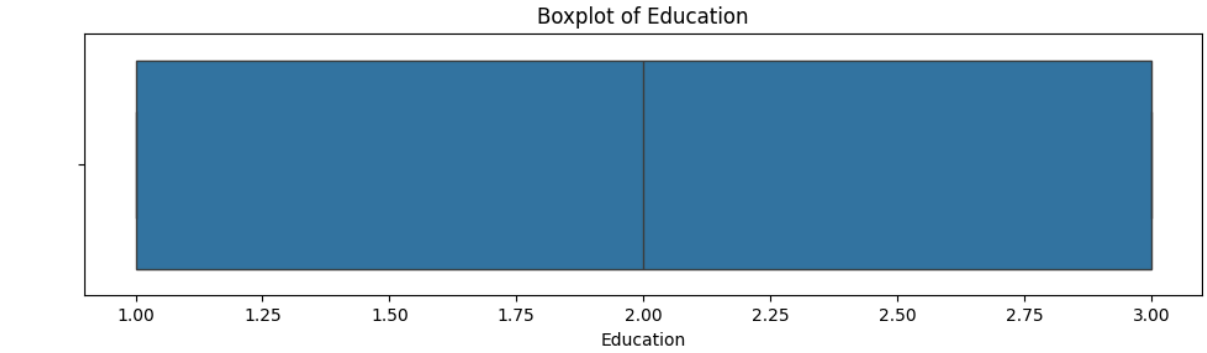

Histogram of Income
Count
0
100
200
300
400
0
50
100
150
200
Income

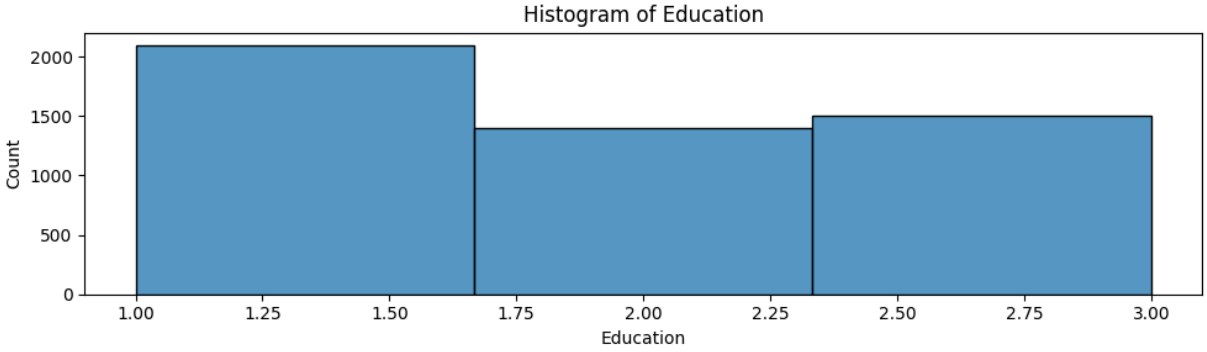

Boxplot of CCAvg
0
2
4
6
8
10
CCAvg

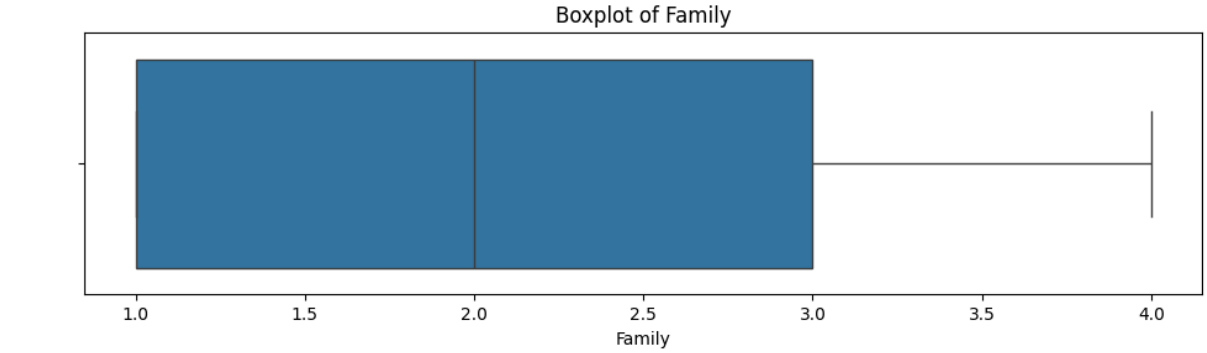

Histogram of CCAvg
Count
0
200
400
600
0
2
4
6
8
10
CCAvg

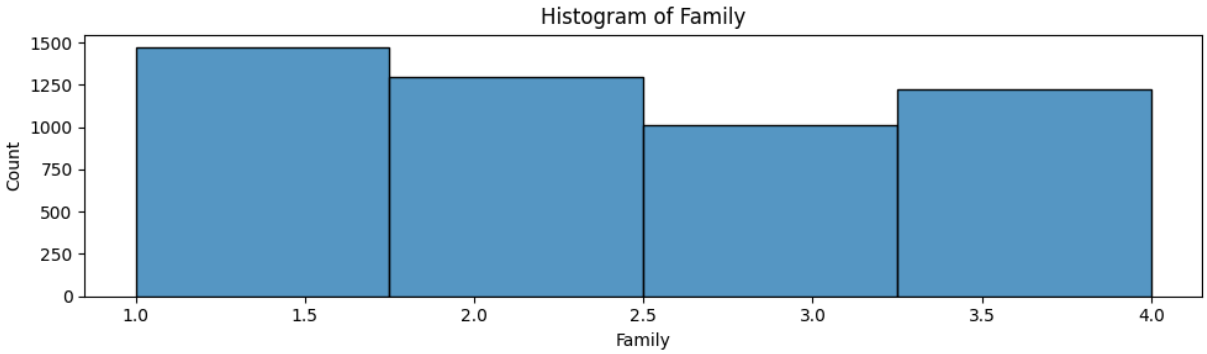

Boxplot of Mortgage
0
100
200
300
400
500
600
Mortgage

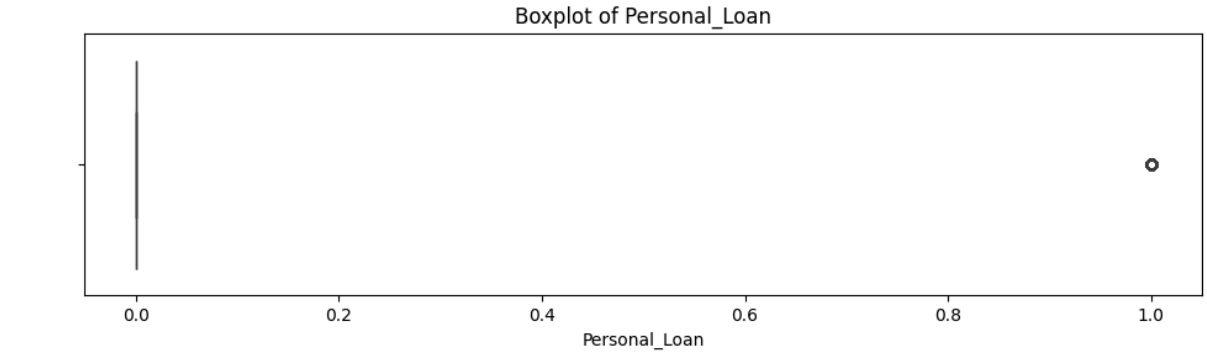

Histogram of Mortgage
Count
3000
2000
1000
0
0
100
200
300
400
500
600
Mortgage

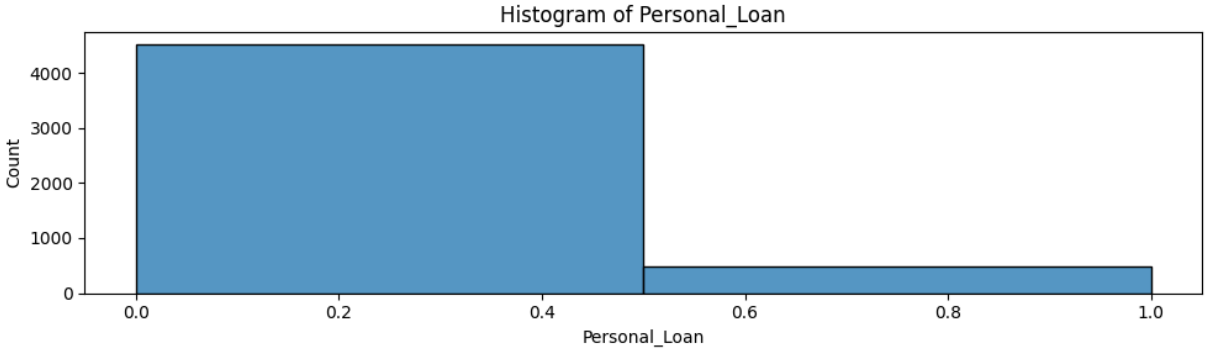

Boxplot of Education
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Education


Histogram of Education
Count
2000
1500
1000
500
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Education




**Observations: Univariate Analysis** Summary of Findings by Variable

Age -Distribution: Fairly symmetric, bell-shaped, ranging from early 20s to late 60s. -Boxplot: No major outliers; a well-centered distribution. -Interpretation: Clean variable, well-distributed — ideal for modeling.

Experience -Distribution: Nearly identical to Age and has what appear to be data entry errors. This variable will be dropped from the model. -Boxplot: A few negative outliers visible.

Income -Distribution: Right-skewed; most customers earn below 100k, with a long tail up to 224k. -Boxplot: Several outliers in the higher-income range. -Interpretation: Might benefit from a log transformation for some models; strong predictor of loan acceptance.

ZIPCode -Distribution: Discrete with vertical spikes — not useful numerically. -Boxplot: Meaningless in this context. I'll either drop this variable from the model.

Family -Distribution: Discrete from 1 to 4. -Boxplot: Evenly spaced; no outliers. Interpretation: Categorical/ordinal; can be used directly or one-hot encoded.

CCAvg (Credit Card Average Monthly Spend) -Distribution: Strong right skew; most values < 3k, a few > 7k. -Boxplot: Many high-value outliers. -Interpretation: Excellent proxy for income.

Education -Distribution: Discrete (1, 2, 3); Level 1 is most common. -Boxplot: Flat and segmented. -Interpretation: Ordinal — can be left as is or converted to dummy variables if needed.

Mortgage -Distribution: Strong right skew; many customers have a 0 mortgage. -Boxplot: Multiple outliers beyond 200k. -Interpretation: Highly imbalanced — consider creating a binary feature (HasMortgage: Yes/No).

Personal_Loan (Target Variable) Distribution: 90% did not accept a loan. Boxplot: Binary values (0, 1).

Securities_Account Distribution: Mostly 0; very few 1s. Boxplot: Binary. Interpretation: Highly skewed binary variable; may have predictive value but not much variability.

CD_Account -Distribution: Skewed binary (most values = 0). -Boxplot: Binary. -Interpretation: Despite skew, this variable had strong correlation with loan acceptance.

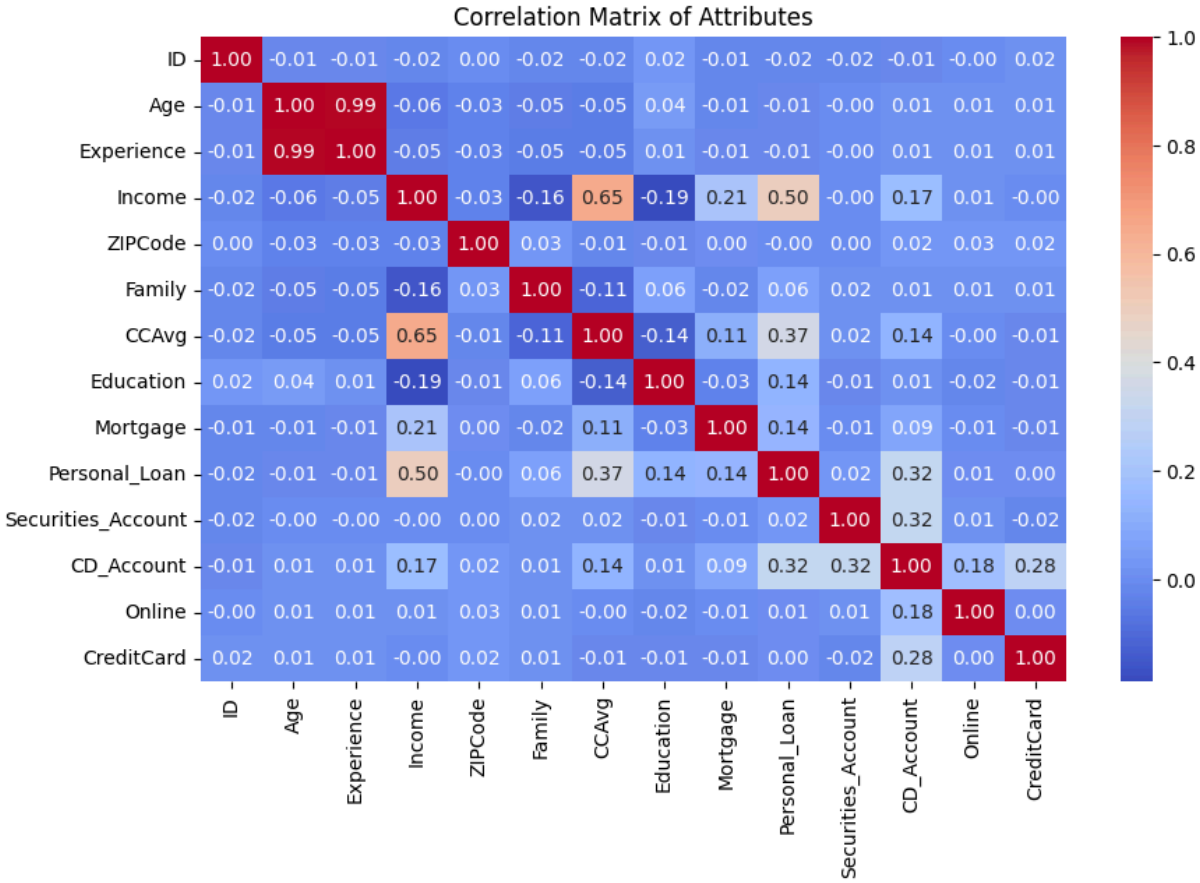
Online -Distribution: Slightly more customers use online banking. -Boxplot: Binary. -Interpretation: Balanced enough to retain; potentially valuable behavioral predictor.

CreditCard -Distribution: About 70% did not have a credit card. -Boxplot: Binary. -Interpretation: Reasonably skewed, but relevant to financial behavior and loan prediction.

In [19]:
```
# What are the attributes that have a strong correlation with the target att

# Calculate the correlation matrix
correlation_matrix = Loan.corr()

# Focus on the correlation with 'Personal_Loan'
personal_loan_correlation = correlation_matrix['Personal_Loan'].drop('Persor

# Print correlations
print(personal_loan_correlation)

# Create a heatmap
```

```
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Attributes')
plt.show()

# Identify attributes with strong correlations (e.g., absolute correlation >
strong_correlations = personal_loan_correlation[abs(personal_loan_correlatic
print("\nAttributes with strong correlation to Personal Loan:")
strong_correlations
```

```
ID                   -0.024801
Age                  -0.007726
Experience           -0.007413
Income                0.502462
ZIPCode              -0.002974
Family                0.061367
CCAvg                 0.366889
Education             0.136722
Mortgage              0.142095
Securities_Account    0.021954
CD_Account            0.316355
Online                0.006278
CreditCard            0.002802
Name: Personal_Loan, dtype: float64
```




```
Attributes with strong correlation to Personal Loan:
```

Out[19]:

| | Personal_Loan |
|---|---|
| **Income** | 0.502462 |
| **CCAvg** | 0.366889 |
| **CD_Account** | 0.316355 |

**dtype:** float64

**Observations:**

These features show the strongest positive correlation with accepting a personal loan:

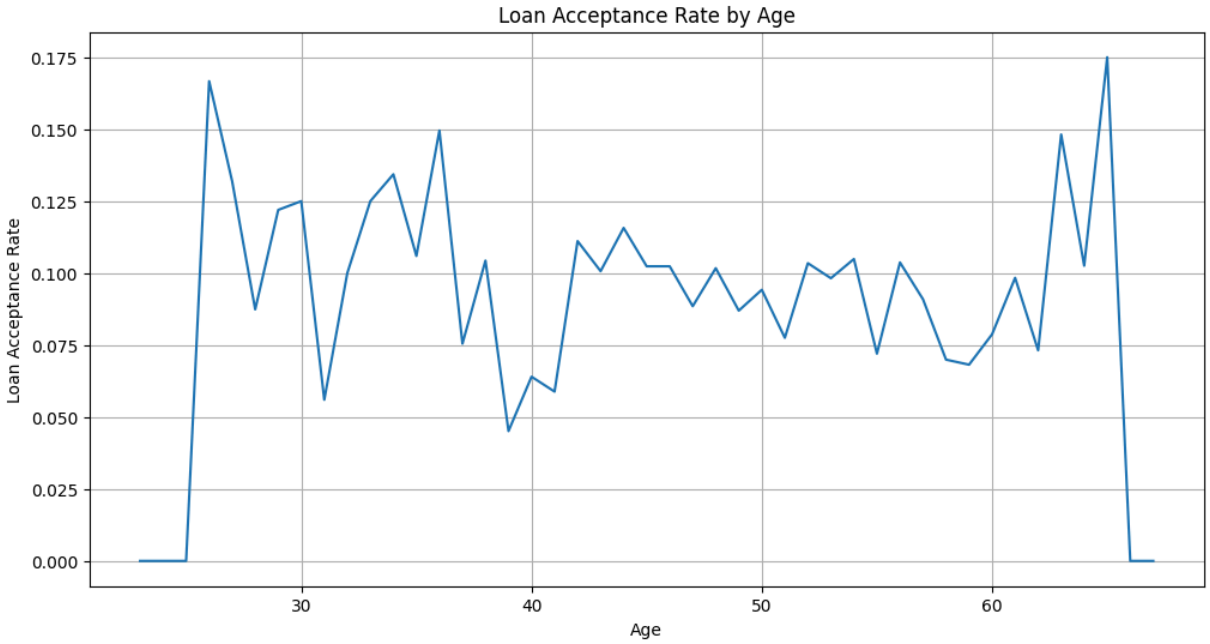
Income: 0.50

CCAvg: 0.37

CD_Account: 0.32

Income is also strongly correlated with CCAvg: .65

In [20]:
```
# How does a customer's interest in purchasing a loan vary with their age?

loan_by_age = Loan.groupby('Age')['Personal_Loan'].mean()

plt.figure(figsize=(12, 6))
loan_by_age.plot()
plt.title('Loan Acceptance Rate by Age')
plt.xlabel('Age')
plt.ylabel('Loan Acceptance Rate')
plt.grid(True)
plt.show()
```



**Observations:**

Loan acceptance begins around age 26 — prior to this, acceptance is essentially zero.

Highest acceptance rate (~17.5%) occurs at age 66, though this may be due to a smaller sample size at the age extremes (more volatile).

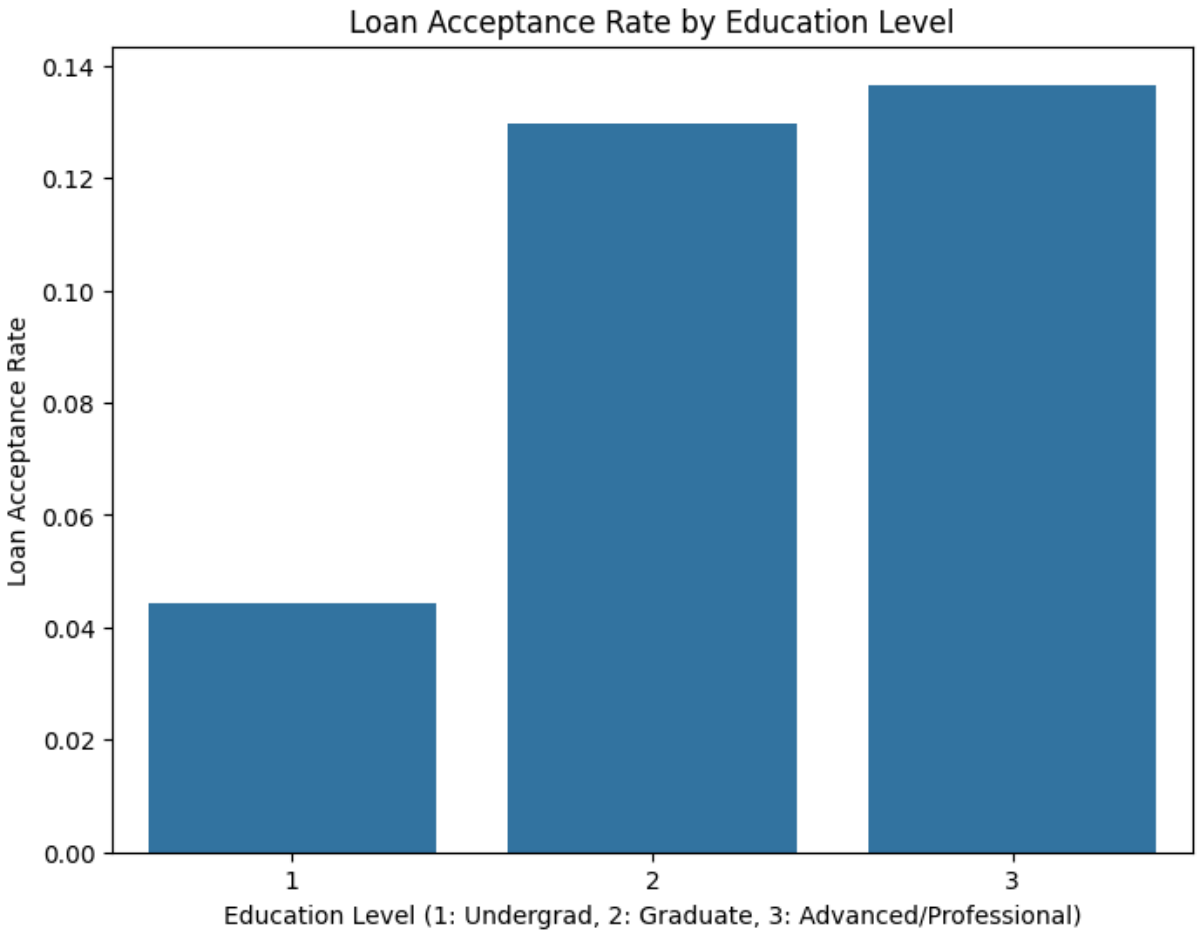
There are spikes in acceptance around ages 27, 35, and mid-60s, suggesting key financial life stages.

The most consistent acceptance range appears to be ages 28 to 50, where the acceptance rate stabilizes around 9% to 13%.

A small dip is visible in the early 30s and again around ages 55 to 60.

The line's volatility suggests certain age groups may have relatively few customers.

In [21]:
```
# How does a customer's interest in purchasing a loan vary with their educat
loan_by_education = Loan.groupby('Education')['Personal_Loan'].mean()

plt.figure(figsize=(8, 6))
sns.barplot(x=loan_by_education.index, y=loan_by_education.values)
plt.title('Loan Acceptance Rate by Education Level')
plt.xlabel('Education Level (1: Undergrad, 2: Graduate, 3: Advanced/Professi
plt.ylabel('Loan Acceptance Rate')
plt.show()
```




**Observations:**

Customers with higher education levels are more likely to accept loans:

Education = 1: 4.4% acceptance

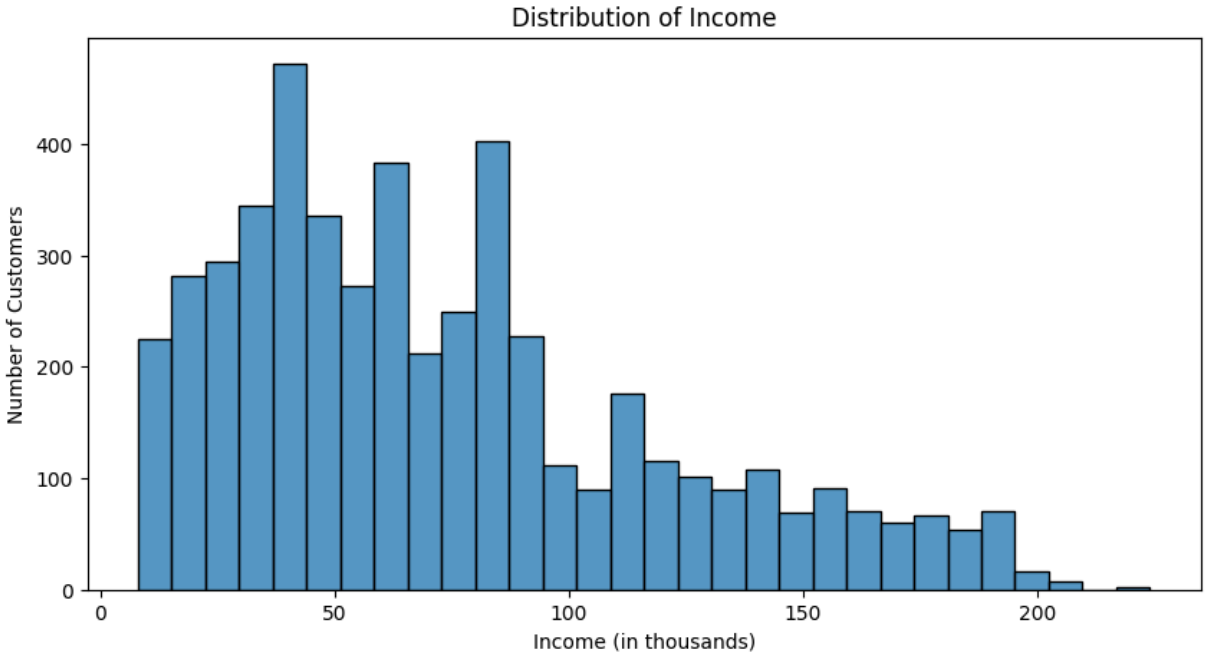
Education = 2: 13.0% acceptance

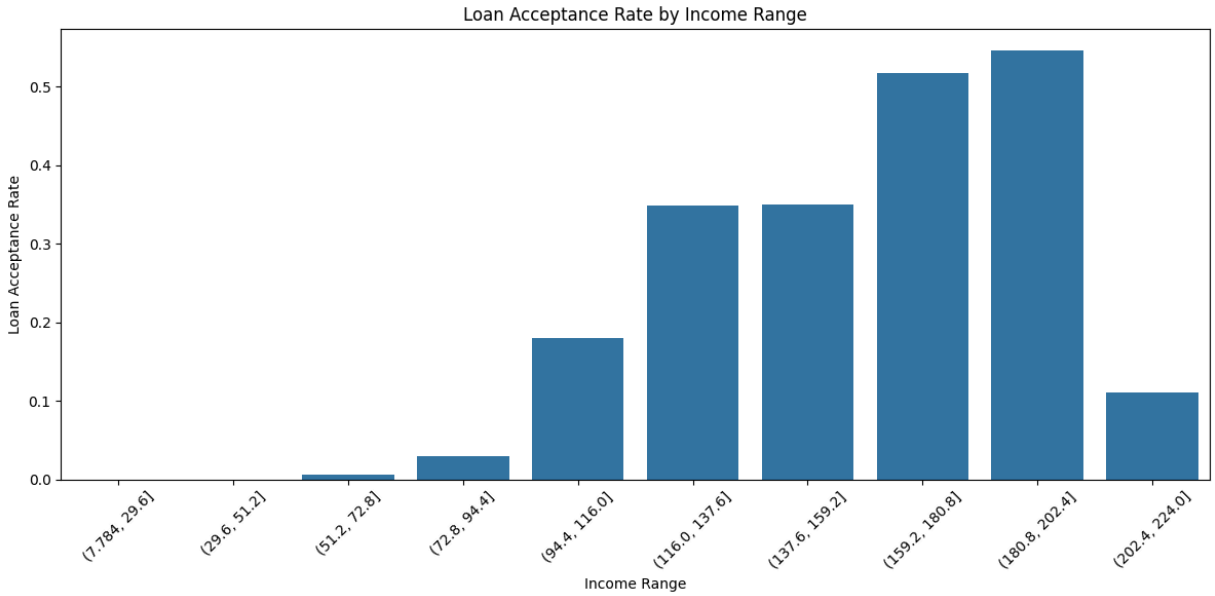
Education = 3: 13.7% acceptance

This indicates that individuals with more education may be more confident or better qualified for personal loan products.

In [22]:
```
# Income Distribution
plt.figure(figsize=(10, 5))
sns.histplot(Loan['Income'], bins=30, kde=False) # Changed df to Loan
plt.title('Distribution of Income')
plt.xlabel('Income (in thousands)')
plt.ylabel('Number of Customers')
plt.show()

# Create income bins and calculate loan acceptance rates
Loan['Income_Bin'] = pd.cut(Loan['Income'], bins=10) # Changed df to Loan
income_loan_rate = Loan.groupby('Income_Bin')['Personal_Loan'].mean().reset_
```

```
# Loan Acceptance by Income
plt.figure(figsize=(12, 6))
sns.barplot(x='Income_Bin', y='Personal_Loan', data=income_loan_rate)
plt.xticks(rotation=45)
plt.title('Loan Acceptance Rate by Income Range')
plt.xlabel('Income Range')
plt.ylabel('Loan Acceptance Rate')
plt.tight_layout()
plt.show()
```




**Observations:**

**Distribution of Income -**

The distribution of income is right-skewed, with most customers earning between 30k and 90k.

A smaller segment earns more than 150k, and only a few reach the upper range around 224k.

**Loan Acceptance by Income -**

Loan acceptance is very low for customers earning less than 75k.

Starting around 95k, acceptance rates jump significantly.

The highest loan acceptance rates (50%+) occur in the income ranges from 150k to 200k+.

There is a strong upward trend: higher income correlates directly with higher willingness or approval to take personal loans.

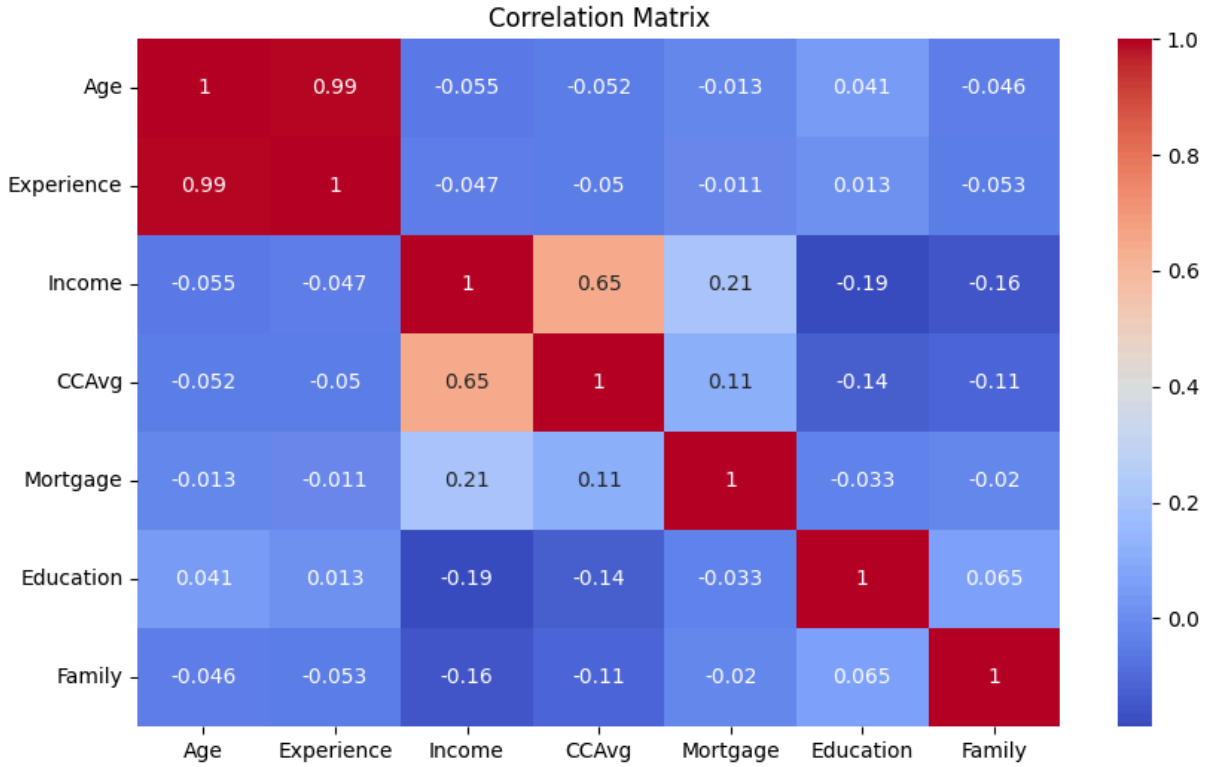
A small dip in the final income bracket (above 200k) may result from a small sample size or reduced need for loans.

In [23]:
```
# Check Variables for Correlation / Multicollinearity
# Correlation Matrix
features_to_check = ['Age', 'Experience', 'Income', 'CCAvg', 'Mortgage', 'Ed
Loan[features_to_check].corr()
```

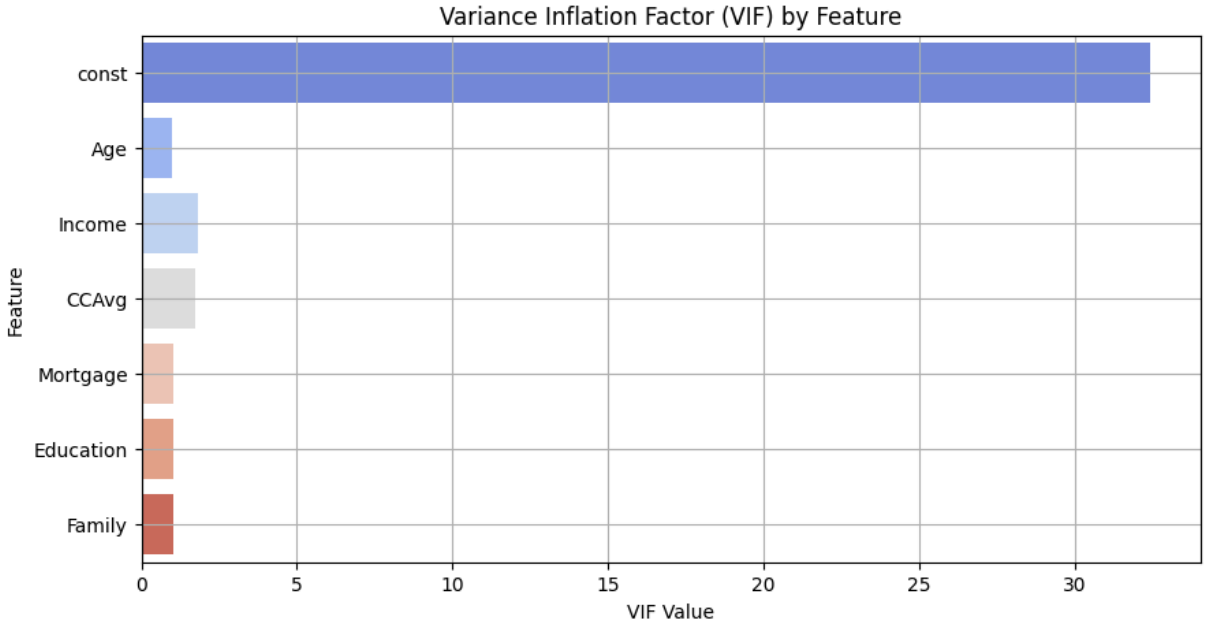
Out[23]:

| | Age | Experience | Income | CCAvg | Mortgage | Education | F |
|---|---|---|---|---|---|---|---|
| **Age** | 1.000000 | 0.994215 | -0.055269 | -0.052012 | -0.012539 | 0.041334 | -0.04 |
| **Experience** | 0.994215 | 1.000000 | -0.046574 | -0.050077 | -0.010582 | 0.013152 | -0.05 |
| **Income** | -0.055269 | -0.046574 | 1.000000 | 0.645984 | 0.206806 | -0.187524 | -0.1 |
| **CCAvg** | -0.052012 | -0.050077 | 0.645984 | 1.000000 | 0.109905 | -0.136124 | -0.10 |
| **Mortgage** | -0.012539 | -0.010582 | 0.206806 | 0.109905 | 1.000000 | -0.033327 | -0.02 |
| **Education** | 0.041334 | 0.013152 | -0.187524 | -0.136124 | -0.033327 | 1.000000 | 0.06 |
| **Family** | -0.046418 | -0.052563 | -0.157501 | -0.109275 | -0.020445 | 0.064929 | 1.00 |

In [24]:
```
# Check Variables for Correlation / Multicollinearity
# Heatmap

plt.figure(figsize=(10, 6))
sns.heatmap(Loan[features_to_check].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```




In [25]:
```
# Check Variables for Correlation / Multicollinearity
# Variance Inflation Factor (VIF)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Select features (omit Experience due to high correlation with Age)
features_vif = ['Age', 'Income', 'CCAvg', 'Mortgage', 'Education', 'Family']
X = add_constant(Loan[features_vif].dropna())

# Calculate VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.s

# Plot VIF values
plt.figure(figsize=(10, 5))
sns.barplot(x="VIF", y="Feature", data=vif_data, palette="coolwarm")
plt.title("Variance Inflation Factor (VIF) by Feature")
plt.xlabel("VIF Value")
plt.ylabel("Feature")
plt.grid(True)
plt.show()
```




**Observations:**

Age & Experience Correlation: 0.99 These features are almost redundant. I'll drop experience because of the 52 instances of data entry errors and derive it from Age during preprocessing.

Income & CCAvg (Credit Card Average Spend) Correlation: 0.65 Shows strong positive correlation. Higher income naturally results in higher spending. I'll consider normalizing or transforming these if the model is sensitive to correlated predictors.

Income & Education Correlation: 0.19 Surprisingly, this shows a slightly inverse relationship in this dataset — possibly due to the discrete nature of education levels or quirks in the data (e.g., high-income individuals with lower formal education).

Low Correlation Across Others. Most other correlations (e.g., Family, Mortgage, Education) are low and don't suggest multicollinearity. I'll keep these features in your the without risk of instability.

All features have VIF < 2, which is excellent. No sign of problematic multicollinearity. Confirmed removing Experience due to its 0.99 correlation with Age was the right thing to do. I will include all of these features in the model.

# Data Preprocessing

- Missing value treatment
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)
- Preparing data for modeling

- Any other preprocessing steps (if needed)

In [26]:
```
#Missing Value Treatment
missing_values = Loan.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
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
Income_Bin            0
dtype: int64
```

**Observations:**

No missing values

In [27]:
```
# Feature Engineering
# Make sure Experience, ID, and ZIP Code are all dropped from the date set

columns_to_drop = []  # Initialize as an empty list
for column in ['Experience', 'ID', 'ZIPCode']:
    if column in Loan.columns:
        columns_to_drop.append(column)

# Now safely drop the columns that exist
Loan.drop(columns=columns_to_drop, axis=1, inplace=True)

# Log Transform skewed features: Income, CCAvg, Mortgage
skewed_cols = ['Income', 'CCAvg', 'Mortgage']
for col in skewed_cols:
    # Use Loan instead of loan_df_cleaned
    Loan[f'log_{col}'] = np.log1p(Loan[col]) # Changed loan_df_cleaned to Lo

# Interaction Features
# Use Loan instead of loan_df_cleaned for calculations
Loan['Income_per_Family'] = Loan['Income'] / Loan['Family'] # Changed loan_d
Loan['CCAvg_per_Income'] = Loan['CCAvg'] / (Loan['Income'] + 1) # Changed lo

# Define features and target
# Use Loan instead of loan_df_cleaned
X = Loan.drop(columns=['Personal_Loan']) # Changed loan_df_cleaned to Loan
y = Loan['Personal_Loan'] # Changed loan_df_cleaned to Loan
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar

# Check engineered features
X_train.head()
```

Out[27]:

| | Age | Income | Family | CCAvg | Education | Mortgage | Securities_Account | CD_Ac |
|---|---|---|---|---|---|---|---|---|
| **1250** | 47 | 81 | 1 | 2.67 | 2 | 0 | 0 | |
| **206** | 49 | 31 | 1 | 1.00 | 1 | 0 | 1 | |
| **2762** | 56 | 65 | 2 | 3.70 | 1 | 0 | 0 | |
| **4276** | 50 | 155 | 1 | 7.30 | 1 | 0 | 0 | |
| **4802** | 35 | 58 | 3 | 2.80 | 1 | 0 | 0 | |

**Observations:**

Here I made sure Experience, ID, and ZIP Code are all dropped from the date set

In [28]:
```
# Outlier Detection and Treatment
for col in ['Income', 'CCAvg', 'Mortgage']:
    upper_limit = Loan[col].quantile(0.99)
    Loan[col] = np.where(Loan[col] > upper_limit, upper_limit, Loan[col])
```

**Observations:**

Applied 99th percentile capping to Income, CCAvg, and Mortgage.

This reduced extreme values without deleting any records, minimizing distortion.

In [29]:
```
from sklearn.preprocessing import StandardScaler

# Define features (X) and target (y)
X = Loan.drop('Personal_Loan', axis=1) # Changed 'df' to 'Loan'
y = Loan['Personal_Loan'] # Changed 'df' to 'Loan'

# Standardize continuous features
continuous_features = ['Age', 'Income', 'CCAvg', 'Mortgage']
scaler = StandardScaler()
X[continuous_features] = scaler.fit_transform(X[continuous_features])
```

**Observations:**

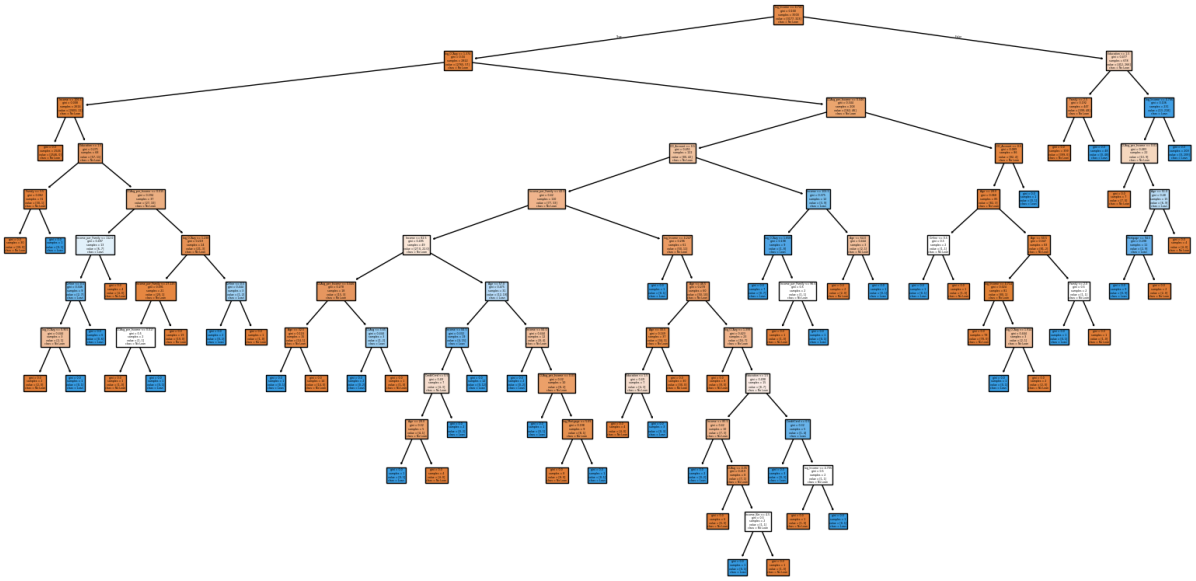
X now contains only valid predictors.

Only continuous features (Age, Income, CCAvg, Mortgage) are scaled.

Ordinal and binary features (e.g., Education, Family, Online, etc.) are left unchanged, as appropriate.

In [30]:
```
# Import the plot_tree function
from sklearn.tree import plot_tree

# Create income bins before splitting into train and test sets
Loan['Income_Bin'] = pd.cut(Loan['Income'], bins=10) # Changed df to Loan

# Define features (X) and target (y)
X = Loan.drop('Personal_Loan', axis=1)  # Changed 'df' to 'Loan'
y = Loan['Personal_Loan']  # Changed 'df' to 'Loan'

# Label Encoding or One-Hot Encoding to the 'Income_Bin' column in X
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
X['Income_Bin'] = label_encoder.fit_transform(X['Income_Bin'])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar

# Initialize and fit the model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Predict on test data
y_pred = dt_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

accuracy, recall, precision, f1, conf_matrix

# 6. Display Results
from sklearn.metrics import classification_report # Importing the classifica
class_report = classification_report(y_test, y_pred, target_names=['No Loan'

print("Model Performance Metrics:")
print(f"Accuracy:  {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

# Visualize the tree
plt.figure(figsize=(20, 10))
plot_tree(dt_model, feature_names=X.columns, class_names=['No Loan', 'Loan']
plt.show()
```

```
Model Performance Metrics:
Accuracy:  0.9860
Precision: 0.9359
Recall:    0.9299
F1 Score:  0.9329

Confusion Matrix:
[[1333   10]
 [  11  146]]

Classification Report:
              precision    recall  f1-score   support

     No Loan       0.99      0.99      0.99      1343
        Loan       0.94      0.93      0.93       157

    accuracy                           0.99      1500
   macro avg       0.96      0.96      0.96      1500
weighted avg       0.99      0.99      0.99      1500
```



# Model Building

## Model Evaluation Criterion

**Key Metrics:**

Accuracy: Good if the classes are balanced. Not reliable if the target (loan acceptance) is imbalanced (which it is here, only ~9.6% accepted).

Precision: Out of all predicted loan acceptances, how many were correct This is important because giving loans to unqualified customers is costly.

Recall (Sensitivity): Out of all actual loan acceptances, how many did we correctly predict? Although this is importnat, false negatives are much less costly (e.g., missing qualified customers who would have accepted a loan) than giving loans to unqualified customers.

F1 Score: Harmonic mean of precision and recall. Useful because it provides balance between precision and recall, especially for an imbalanced dataset such as Loan_Modelling.

AUC-ROC: Measures model's ability to distinguish between classes. Good overall performance metric, especially for an imbalanced dataset such as Loan_Modelling.

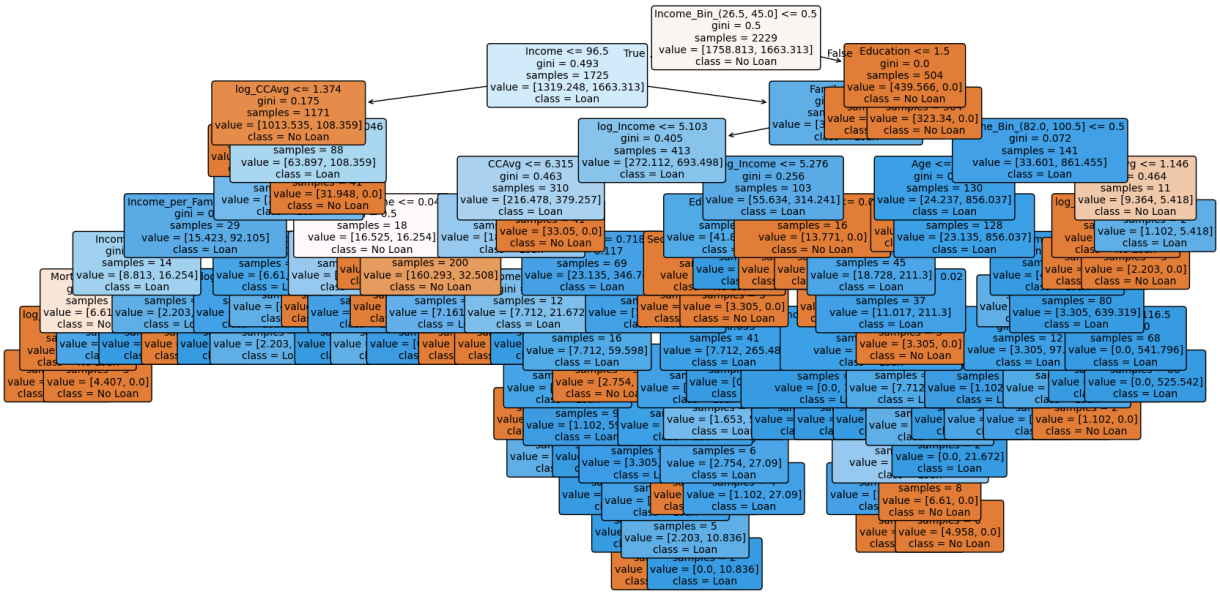
## Model Building

# Model Performance Improvement

In [31]:
```
# Import necessary libraries
import numpy as np
from sklearn.ensemble import RandomForestClassifier  # Import RandomForestCl
from sklearn.metrics import make_scorer, f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder  # Import OneH

# Create income bins before splitting into train and test sets
Loan['Income_Bin'] = pd.cut(Loan['Income'], bins=10) # Changed df to Loan

# Define features (X) and target (y)
X = Loan.drop('Personal_Loan', axis=1)  # Changed 'df' to 'Loan'
y = Loan['Personal_Loan']  # Changed 'df' to 'Loan'

# One-Hot Encoding for 'Income_Bin'
# Create a OneHotEncoder object
# Set sparse=False and handle_unknown='ignore'
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Fit and transform the 'Income_Bin' column on the entire dataset before spl
encoded_income_bin = encoder.fit_transform(X[['Income_Bin']])

# Create a DataFrame with the encoded columns
encoded_income_df = pd.DataFrame(encoded_income_bin, columns=encoder.get_fea

# Concatenate the encoded features with the original DataFrame
X = pd.concat([X, encoded_income_df], axis=1)

# Drop the original 'Income_Bin' column
X.drop('Income_Bin', axis=1, inplace=True)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar

# Initialize and fit the model
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, f1_score

# Define Random Forest and parameter grid
rf = RandomForestClassifier(random_state=42, class_weight='balanced')

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# F1 Score as the main evaluation
scorer = make_scorer(f1_score)

# Grid Search
# Set n_jobs=1 to disable parallel processing
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring=scorer,
    cv=5,
    n_jobs=1, # Change n_jobs to 1
    verbose=1
)

# Fit the model
grid_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best F1 Score:", grid_search.best_score_)

# Evaluate on test data
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, target_names=['No Loan', 'Loan']

# Visualize one of the trees from the Random Forest
# Get the first tree from the forest
tree_to_visualize = best_rf.estimators_[0]

# Plot the tree
plt.figure(figsize=(20, 10))
plot_tree(tree_to_visualize,
          feature_names=X.columns,
          class_names=['No Loan', 'Loan'],
          filled=True,
          rounded=True,
          fontsize=10)
plt.show()
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_spl
it': 5, 'n_estimators': 200}
Best F1 Score: 0.9089755095065645
              precision    recall  f1-score   support

     No Loan       0.99      0.99      0.99      1343
        Loan       0.95      0.94      0.95       157

    accuracy                           0.99      1500
   macro avg       0.97      0.97      0.97      1500
weighted avg       0.99      0.99      0.99      1500
```




# Model Performance Comparison and Final Model Selection

**Observations:**

Comparison: Original vs. Refined Model

Metric Original Model Refined Model (After Tuning) Precision (Loan) 93.59% 95% Recall (Loan) 92.99% 94% - 95% F1 Score (Loan) 93.29% 94% - 95% Accuracy 98.6% 99% F1 (CV Score) ~0.91 ~0.91 -Refined Model Results

Precision went from 93.6% to 95%.

Recall went from 92.99% to 95%.

The overall F1 Score went from 93.29% to 95%.

Accuracy is still excellent at 99%.

Conclusion: There is some minimal improvement to our already strong baseline model. At this point the refined model is operating efficiently enough to deploy.

# Actionable Insights and Business Recommendations

**Recommendations for AllLife Bank:**

**Target High-Income Customers:**

Insight: Higher Income is strongly correlated with loan acceptance. EDA and the model confirm that customers earning above $100,000 are more likely to accept a personal loan.

Action Plan:

Segment: Identify depositors with annual incomes > $100K.

Offer: Promote larger personal loans with competitive interest rates or bundled offers (e.g., loan + investment advice).

Channels: Use direct marketing, personalized offers via email, and relationship manager calls.

Value Proposition: Highlight how personal loans can help diversify investments or offer financial flexibility for high-end purchases.

**Cross-sell to CD Account Holders:**

Insight: CD Account holders showed higher loan acceptance rates. These customers already have a trust-based financial relationship with the bank.

Action Plan:

Segment: Isolate customers with active CD Accounts.

Offer: Exclusive loan rate discounts or fee waivers for CD holders.

Timing: Target customers nearing CD maturity dates—they might seek liquidity.

Messaging: Position loans as tools to unlock liquidity while retaining investment stability.

**Online Banking Users:**

Insight: Customers using online services are tech-savvy and more responsive to digital outreach. They are accessible for cost-effective campaigns.

Action Plan:

Segment: Focus on customers marked as Online = 1.

Offer: Quick-apply loan features through the online banking portal.

Channels: Use email, SMS, and in-app notifications for offers.

Call to Action: Highlight instant approvals, paperless applications, and personalized loan calculators available online.

**Segment Marketing by Education Level:**

Insight: Customers with Graduate or Advanced/Professional degrees (Education = 2 or 3) are more financially literate, likely to consider structured loans.

Action Plan:

Segment: Group customers by education level.

Offer: Tailor premium loan packages or flexible repayment plans.

Messaging: Focus on lifestyle enhancements, career-based financial planning, and wealth-building tools.

Partnerships: Collaborate with professional associations or alumni networks for targeted promotions.

---