DSC550-T301 Data Mining

Term Project: Term Project Milestone 1: Data Selection and EDA

Saravanan Janarthanan

Project Milestone 1 It is time to start using what you have learned throughout the first half of this course by developing an original data mining project. This week you will develop the project idea and do some data exploration/graphical analysis. You will continue working on and updating this project for the remainder of the term. The following link includes a sample project completed in Jupyter Notebook: Titanic Model Building Project Sample.

- The first step is coming up with an idea – arguably one of the hardest steps! Identify an original business problem for your project that can be solved with an appropriate model. By a business problem, it is meant that you should work on a problem where there is a good reason to solve it. There should be some organization or company that would find the solution to the problem useful. There are lots of ideas available online through Kaggle and other sources, but your idea should have a unique spin on it. The second step is locating your data. This can come from a variety of sources, e.g., Kaggle, your job, a website, API, etc. Feel free to reach out to your instructor if you are not sure if your idea and data are suitable. You may need to adjust your idea on the availability of data.
- Begin Milestone 1 with a 250-500-word narrative describing your original idea for the analysis/model building business problem. Clearly identify the problem you will address and the target for your model. Then, do a graphical analysis creating a minimum of four graphs. Label your graphs appropriately and explain/analyze the information provided by each graph. Your analysis should begin to answer the question(s) you are addressing. Write a short overview/conclusion of the insights gained from your graphical analysis.
- As a reminder, Teams is a great place to discuss your project with your peers. Feel free to solicit feedback/input (without creating a group project!) and collaborate on your projects with your peers. Each milestone will build on top of each other, so make sure you do not fall behind.

Lending and Credit Risk

Lending involves providing funds, typically as a loan, to individuals, businesses, or other entities, with the expectation that the borrowed capital will be repaid with interest or according to the terms of a loan agreement. This financial practice is widespread and fundamental to the economy.

Loan default occurs when a borrower fails to comply with the specified terms and conditions of the loan agreement. This often includes not making timely payments or breaching other contractual commitments. The persistent risk of a borrower failing to make payments necessitates accurate credit risk assessment, which is crucial.

Credit risk analytics transform both historical and projected data into actionable insights, enabling financial institutions to evaluate risk and determine lending and account management strategies. One approach organizations use is integrating credit risk modeling into their decision-making processes.

Significance of credit risk to finanical institiutions

Credit risk refers to the possibility that a borrower or debtor may not fulfill their financial commitments. For financial institutions, credit risk poses a threat to their financial stability, profitability, and overall operational health.

Evaluating the creditworthiness of potential borrowers

Assessing the creditworthiness of potential borrowers by financial institutions is a complex procedure that encompasses a thorough examination of a range of financial and personal elements.Some of them are

- Credit History and its length
- Credit Score– Debt to Income ratio
- Payment History– Credit Utilization
- Employment or income
- Collateral (Vehcile, House , stock and securities etc)

Personal Elements that contribute to credit risk are detailed below

- **Credit History**: An individual's credit history is one of the most critical factors contributing to credit risk.

- **Credit Score**: Credit scores, like FICO or VantageScore, provide a numerical representation of an individual's creditworthiness

- **Debt Levels**: The amount of debt an individual carries, including credit card balances, personal loans, and mortgages, relative to their income

- **Payment History**: Consistently making on-time payments on previous loans and credit accounts reflects positively on an individual's credit risk profile

- **Credit Utilization**: Lenders evaluate the ratio of credit card balances to credit limits. A lower credit utilization rate is seen as more favorable, as it indicates responsible credit use

- **Credit Inquiries**: Frequent applications for new credit can be viewed negatively, as they may indicate financial distress or an increased risk of taking on additional debt

Problem Statement

How do financial institutions evaluate the creditworthiness of potential borrowers by integrating credit risk modeling into their decision-making processes?

To address the problem statement, credit risk analytics will be developed by collecting data, performing Exploratory Data Analysis (EDA), and utilizing predictive classification models. Machine learning will be employed to evaluate the creditworthiness of borrowers.

Based on the approach detailed above, utilize datasets identified that includes the most of the personal elements listed above

- It involves the analysis and selective integration of this data after thorough cleansing and transformation through Exploratory Data Analysis and feature engineering.
- Subsequently, a conceptual model will be developed to detail the use of featues / predictors and response variables.
- Programmatically building and evaluating the predictive model
- The collected data will be divided into training and testing data sets- Build the model using python libraries.- Machine learning algorithms or ensemble techniques will be chosen formodel training using the training data set.
- Finally, the model's validity will be assessed using the test data set.

Dataset Identified

Datasets from the Kaggle are identified Credit Risk Dataset : This is a simulated data based on credit bureau data https://www.kaggle.com/datasets/laotse/credit-risk-dataset/data

Feature Name	Description
person_age	Age of the person
person_income	Annual Income of the person

person_home_ownership| Home ownership - whether person owns a homeor mortage or rented| person_emp_length | Employment length (in years)| |loan_intent | Loan intent - intent of the loan| |loan_grade | Loan grade - sclae based on credit worthiness| |loan_amnt | Loan amount| |loan_int_rate | Interest rate for the loan| |loan_status | Loan status - credit approval | |loan_percent_income | Percent income - % of income represented by loan| |cb_person_default_on_file| Historical default (previous default history from Bureau)| |cb_preson_cred_hist_length| Credit history length|

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
credit_data_df = pd.read_csv("credit_risk_dataset.csv")
```

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

Out[3]:

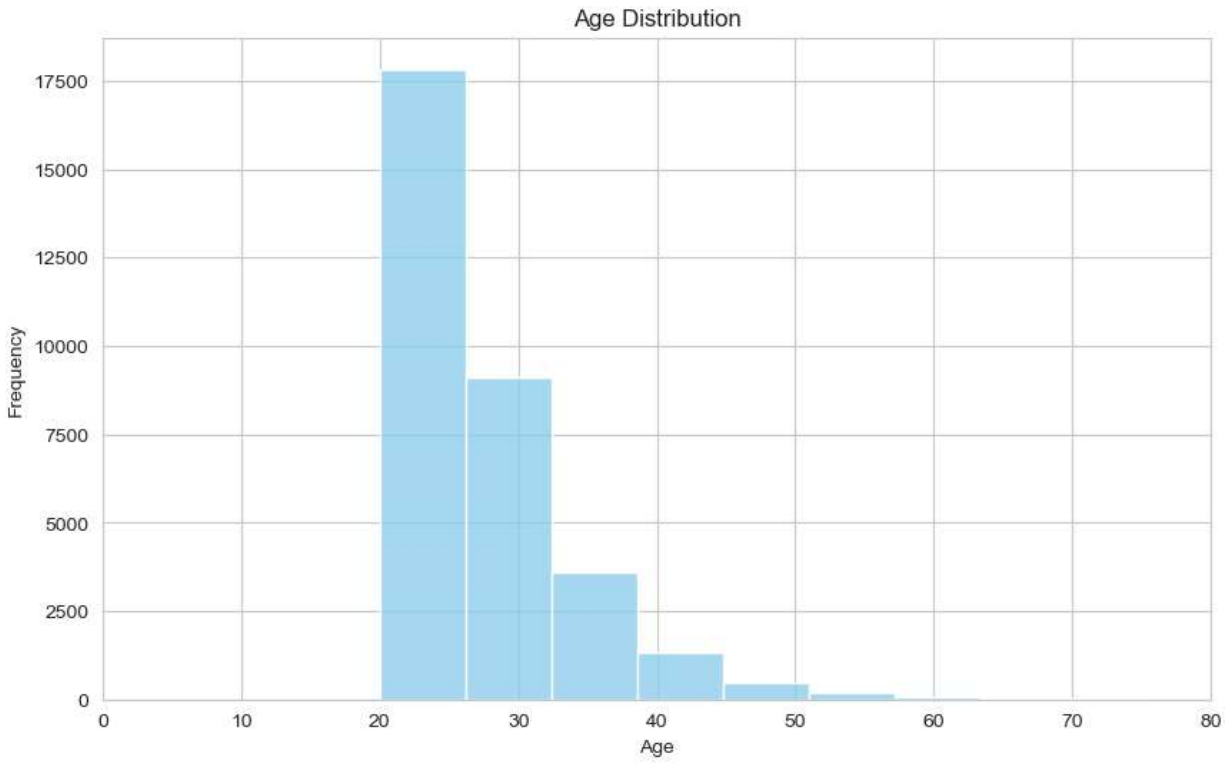
	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade
0	22	59000	RENT	123.0	PERSONAL	D
1	21	9600	OWN	5.0	EDUCATION	B
2	25	9600	MORTGAGE	1.0	MEDICAL	C
3	23	65500	RENT	4.0	MEDICAL	C
4	24	54400	RENT	8.0	MEDICAL	C

In [4]:
```
#Print the shape of the dataset
print("Shape of the dataset : ", credit_data_df.shape )
```

```
Shape of the dataset :  (32581, 12)
```

Find the frequency of age of the applicants

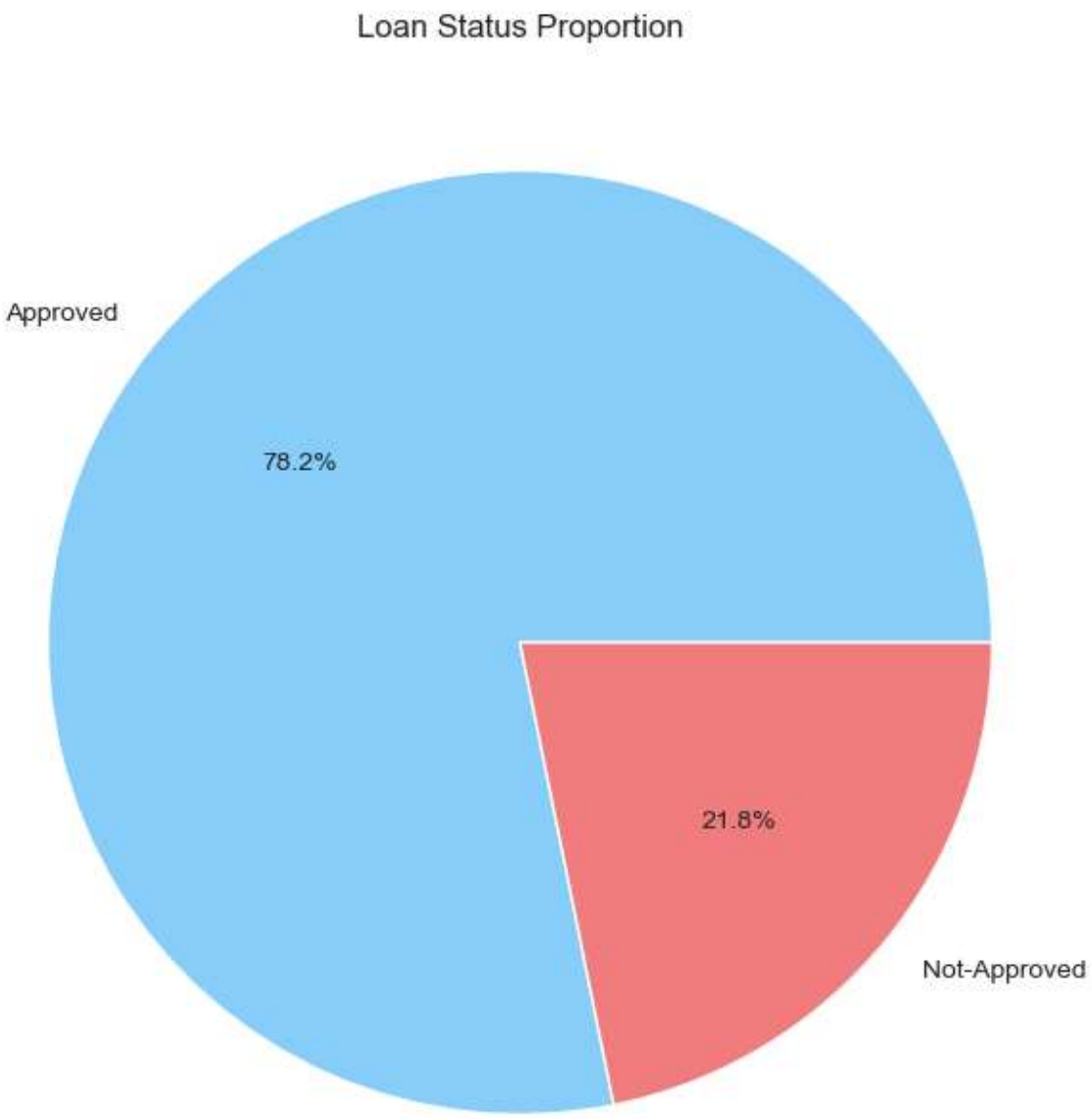
In [5]:
```
# Use the person_age feature or column values and use a histogram to display the frequ
# Set the aesthetic style of the plots
sns.set_style("whitegrid")

# Age Distribution Histogram
plt.figure(figsize=(10, 6))
sns.histplot(credit_data_df['person_age'], bins=20, color='skyblue')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.xlim(0, 80)
plt.ylabel('Frequency')
plt.show()
```



From the above 55% of the applicants are in the twenties and 27% are in the thirties. This age range reflects most of the appicants are young generation

Display the target variable data 'Balanced' status

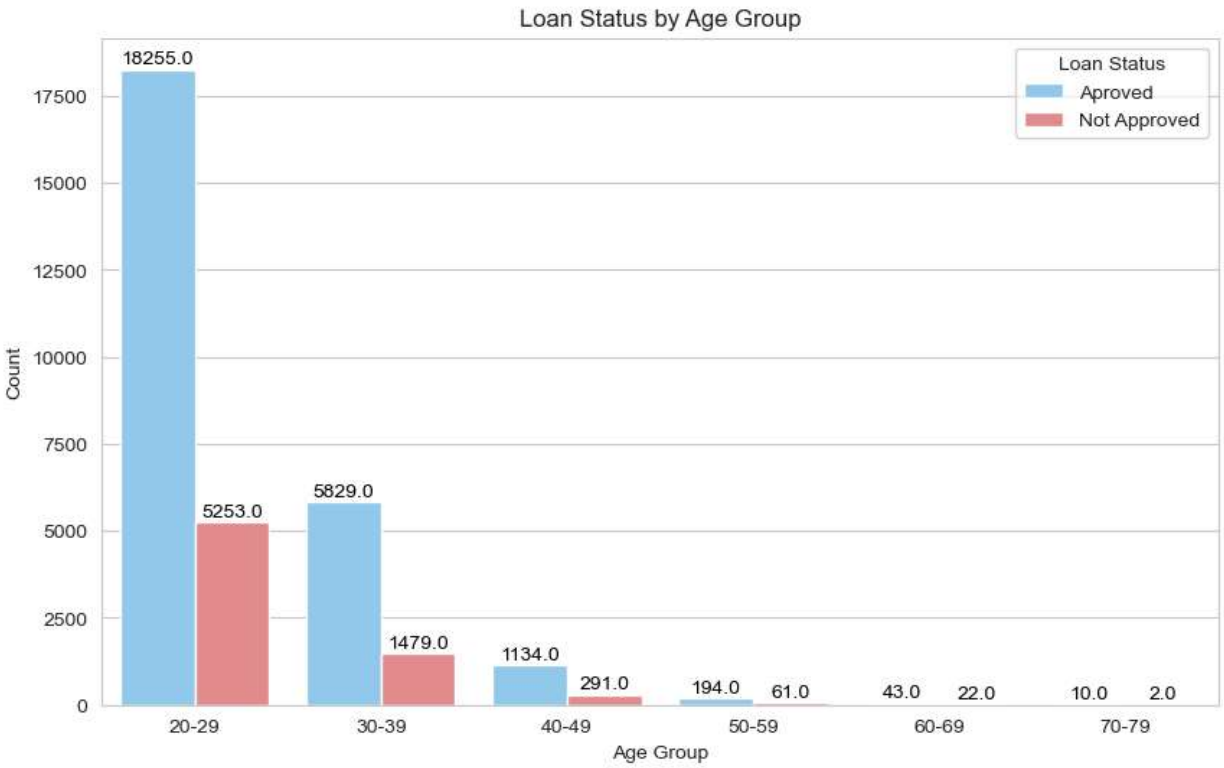
In [6]:
```
# Since loan_status is identified as the target variable,  use a pie chart to plot the
# Loan Status Proportion
plt.figure(figsize=(8, 8))
loan_status_counts = credit_data_df['loan_status'].value_counts()
plt.pie(loan_status_counts, labels=['Approved', 'Not-Approved'], autopct='%1.1f%%', cc
plt.title('Loan Status Proportion')
plt.show()
```



From the above the dataset is not balanced and need to be balanced to avoid overfitting

In [7]:
```
# Define the bins and labels
bins = [20, 30, 40, 50, 60, 70, 80]
labels = ['20-29', '30-39', '40-49', '50-59', '60-69', '70-79']

# Bin the 'person_age' into age groups
credit_data_df['age_group'] = pd.cut(credit_data_df['person_age'], bins=bins, labels=l

# Create a count plot of 'loan_status' against 'age_group'
plt.figure(figsize=(10, 6))
ax = sns.countplot(data=credit_data_df, x='age_group', hue='loan_status', palette=['li
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2., height), ha='center', va
plt.title('Loan Status by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Count')
```

```
plt.legend(title='Loan Status', labels=['Aproved', 'Not Approved'])
plt.show()
```



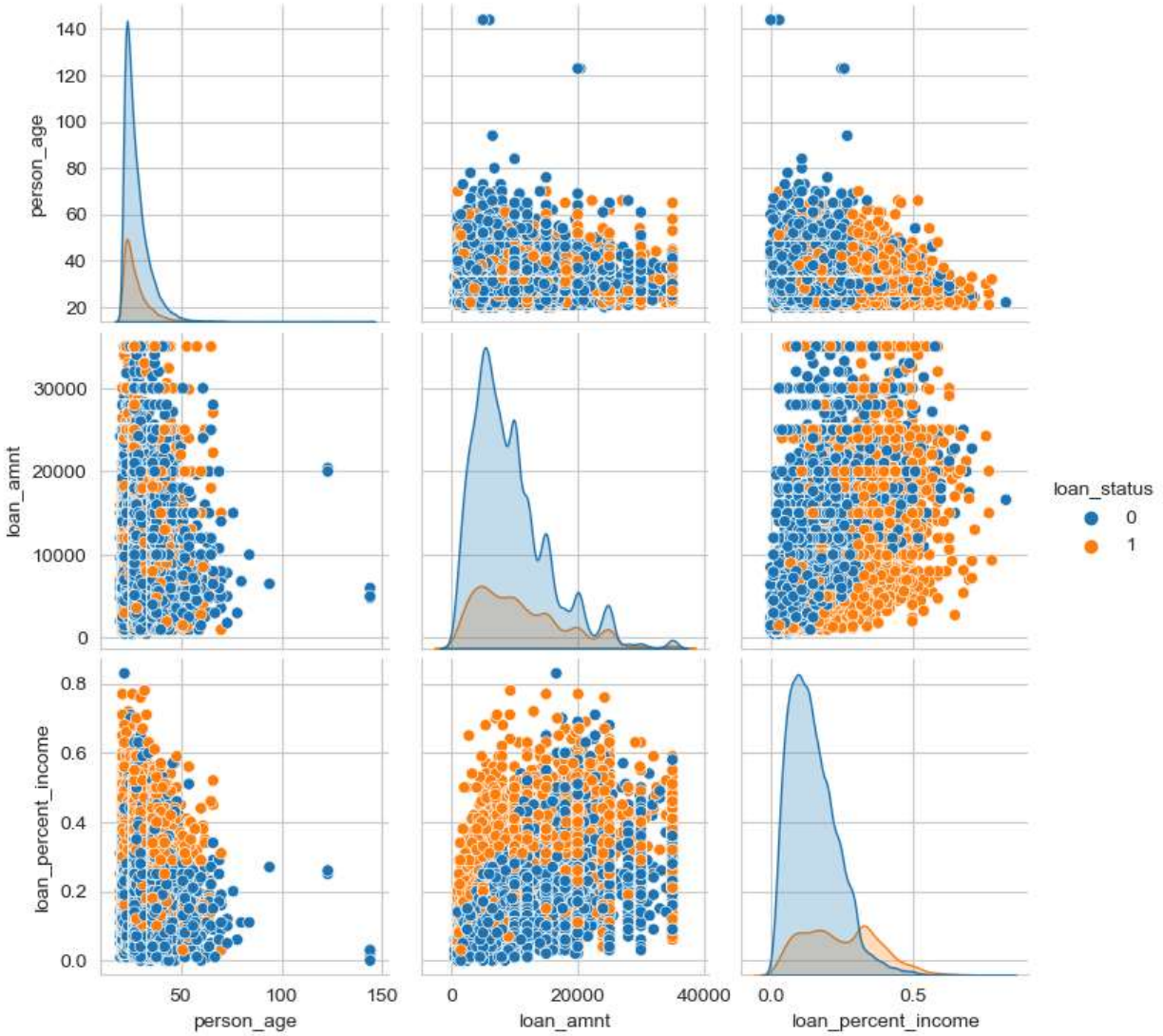
From the above graph close to 25% to 30% are not approved

Find the correlation between Age, Loan amount and Loan percent Income

Use a pair plot to find the correlation between between Age, Loan amount, Loan percent Income. Use the Loan status as a scatter trend to identify approved and not approved status

In [8]:
```
pirplot_lst = ['person_age', 'loan_amnt', 'loan_percent_income', 'loan_status']
sns.pairplot(credit_data_df[pirplot_lst], hue='loan_status')
plt.show()
```

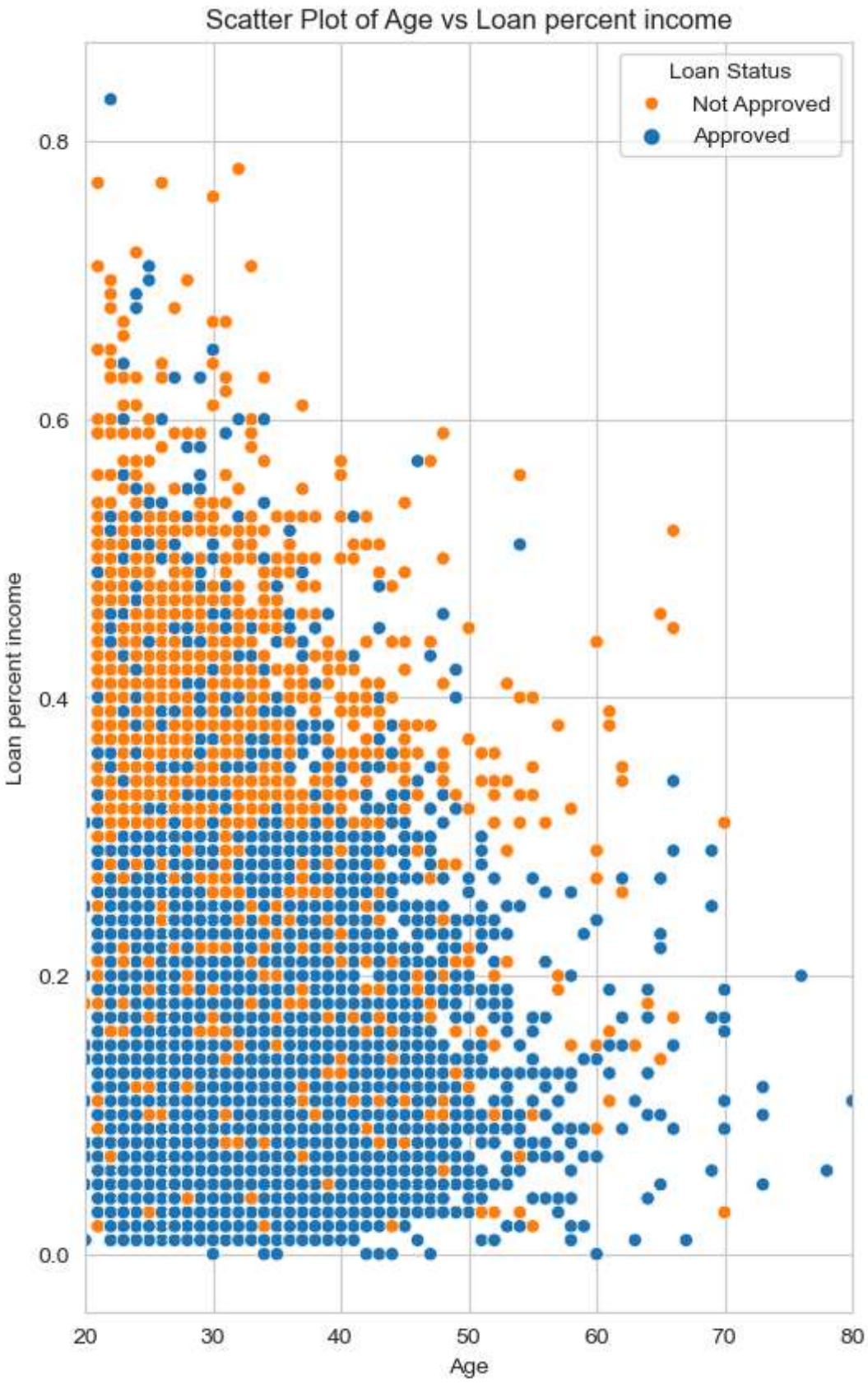
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Use a Scatter plot to analyze individual data except person age vs loan amount as the approved status is mixed and spread through out. Hence plot Age vs Loan percent income and Loan percent income vs loan amount

In [9]:
```
#default_on_file_mapping = {'N': 0, 'Y': 1}
#credit_data_df['cb_person_default_on_file_num'] = credit_data_df['cb_person_default_c
```

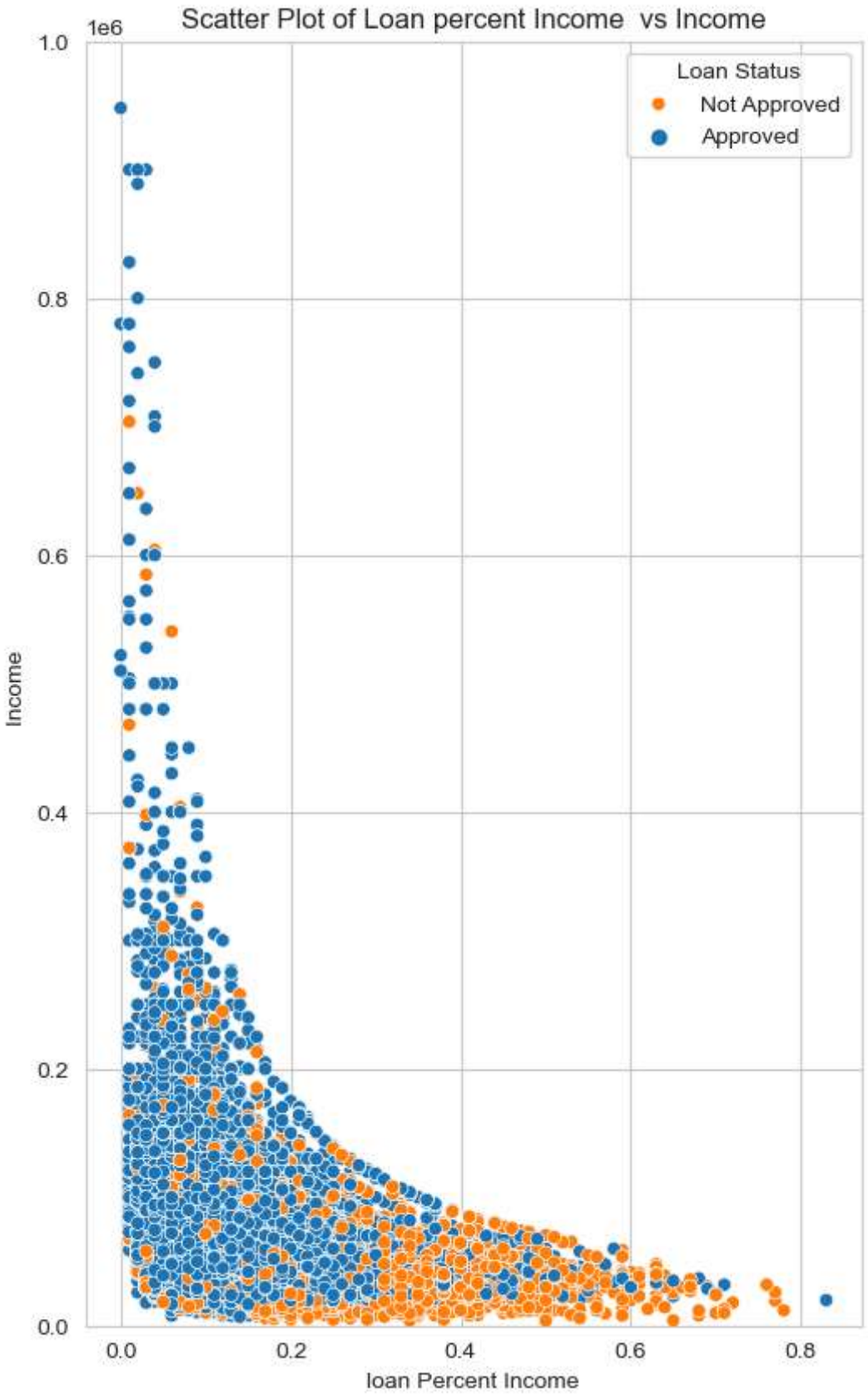
In [10]:
```
# Scatter plot for person_age vs loan_amnt with loan_status as hue
plt.figure(figsize=(6, 10))
sns.scatterplot(data=credit_data_df, x='person_age', y='loan_percent_income', hue='loa
plt.title('Scatter Plot of Age vs Loan percent income')
plt.xlabel('Age')
plt.ylabel('Loan percent income')
plt.xlim(20, 80)
plt.legend(title='Loan Status', labels=['Not Approved', 'Approved'])
plt.show()
```



The above scatter shows a loan percent income above 0.3 ratio has a low approval chance

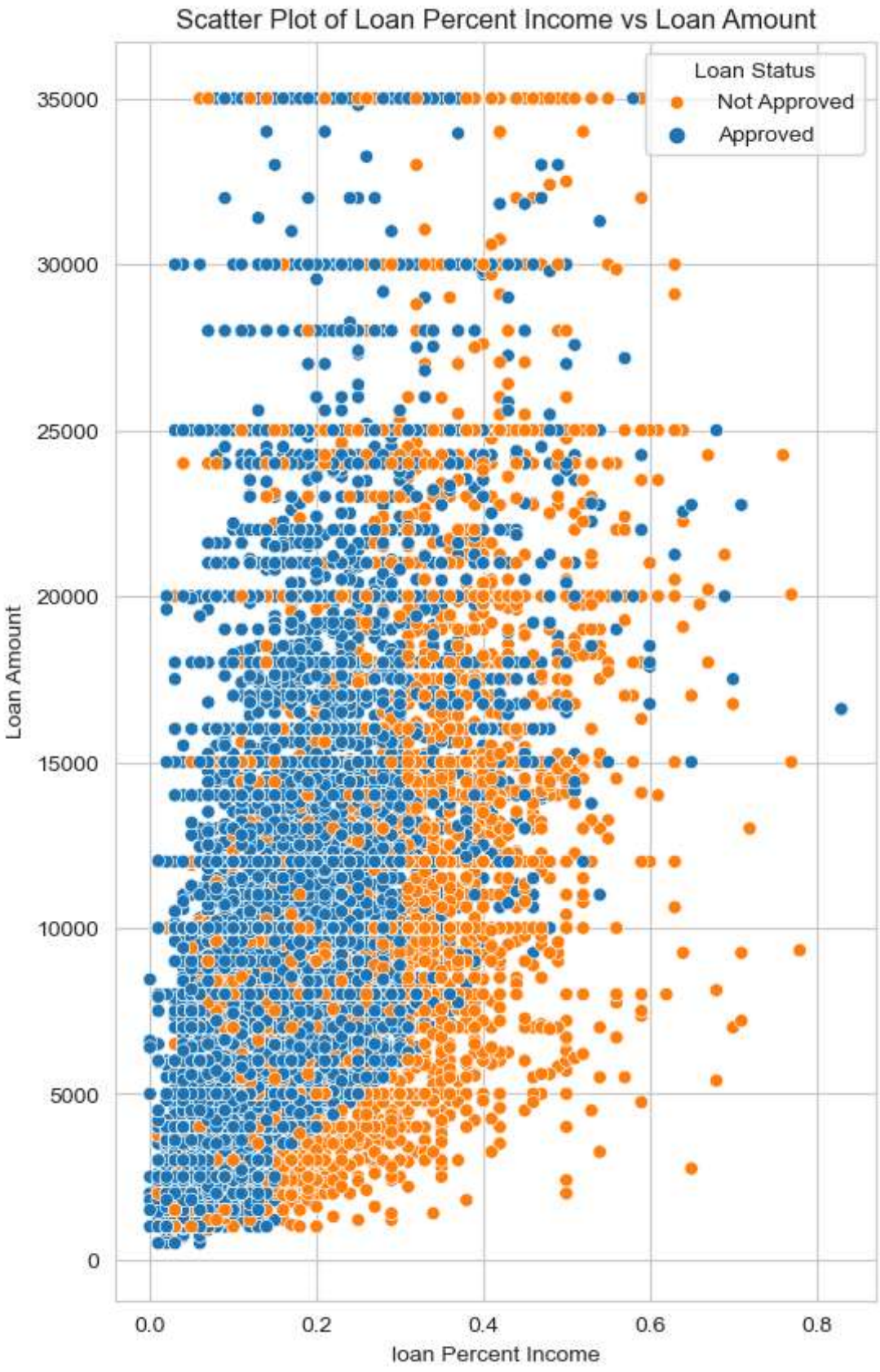
In [11]:
```
# Scatter plot for person_age vs loan_amnt with loan_status as hue
plt.figure(figsize=(6, 10))
sns.scatterplot(data=credit_data_df, x='loan_percent_income', y='person_income', hue='
plt.title('Scatter Plot of Loan percent Income  vs Income')
```

```
plt.xlabel('loan Percent Income')
plt.ylabel('Income ')
plt.ylim(0, 1000000)
plt.legend(title='Loan Status', labels=['Not Approved', 'Approved'])
plt.show()
```



In [12]:

```
# Scatter plot for person_age vs loan_amnt with loan_status as hue
plt.figure(figsize=(6, 10))
sns.scatterplot(data=credit_data_df, x='loan_percent_income', y='loan_amnt', hue='loar
plt.title('Scatter Plot of Loan Percent Income vs Loan Amount')
plt.xlabel('loan Percent Income')
plt.ylabel('Loan Amount')
plt.legend(title='Loan Status', labels=['Not Approved', 'Approved'])
plt.show()
```



The scatter plot illustrates that the loan amount and the loan percent ratio exhibit a positive correlation for approved loans. Conversely, even for lower loan amounts, if the loan percent income ratio exceeds 0.2, the loans are not approved

---

Overview

- From the graphical charts it is evident that age, loan amount, loan percent income will be determental in aiding the decisons for credit risk and loan approval.
- From the scatter plots there are outliers that are evident visually like age above 100 , income upto 6 million , loan percent income above 80% and still approved.
- Target variable loan_status has unbalanced data that is visible in the pie chart
- There are other cateogorical variables that might infleunce the credit risk and loan approvals and can be analyzed during feature engineering and model building.
- Most of the applicants age range betwen 20 to 40, but that may be fact considering the risk profile

---

Term Project: Term Project Milestone 2: Data Preperation

Saravanan Janarthanan

Drop any features that are not useful for your model building and explain why they are not useful.

Perform any data extraction/selection steps.

Transform features if necessary.

Engineer new useful features.

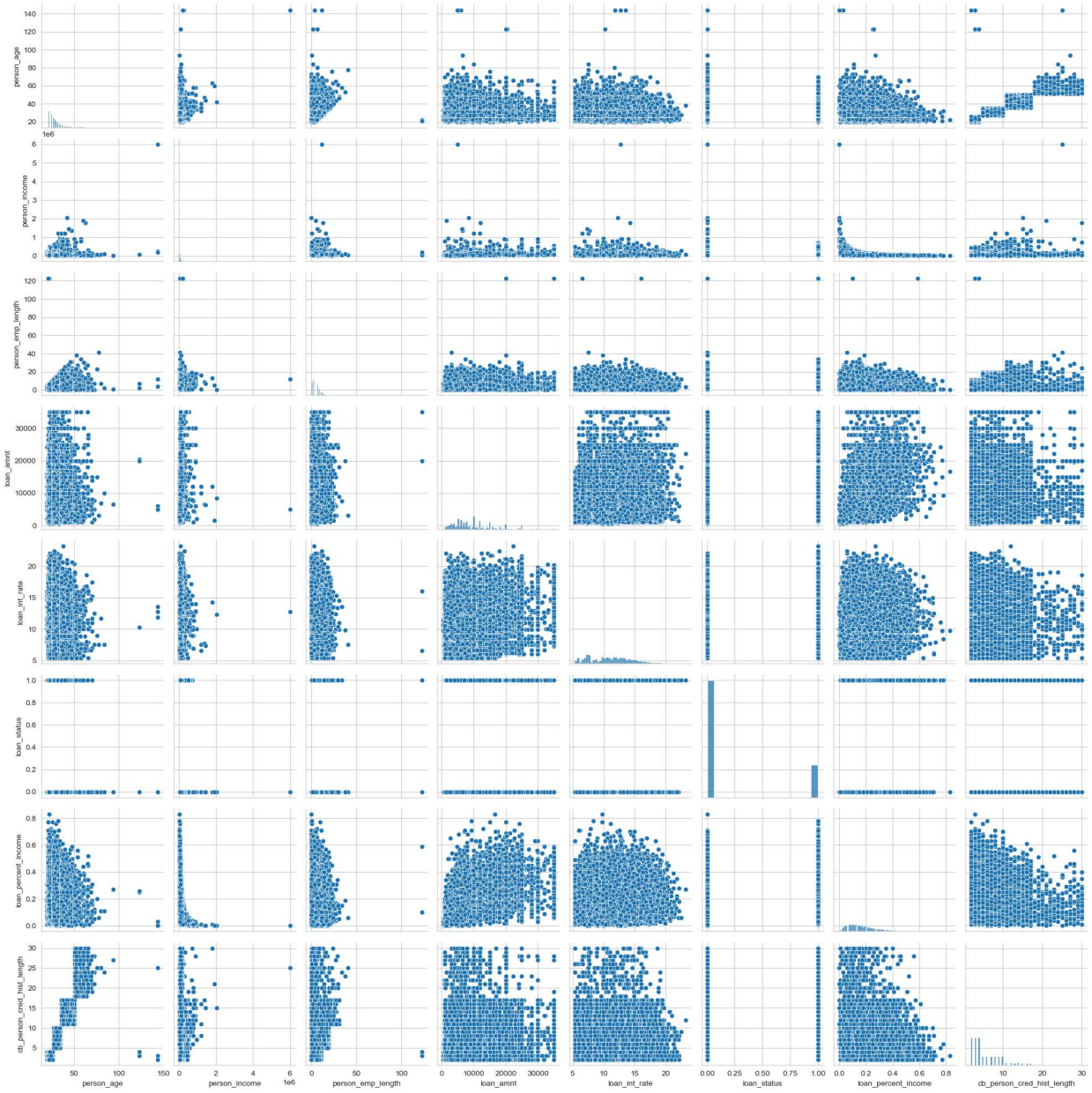
Deal with missing data (do not just drop rows or columns without justifying this).

Create dummy variables if necessary.

MLS2 - Q1. Drop any features that are not useful for your model building and explain why they are not useful.

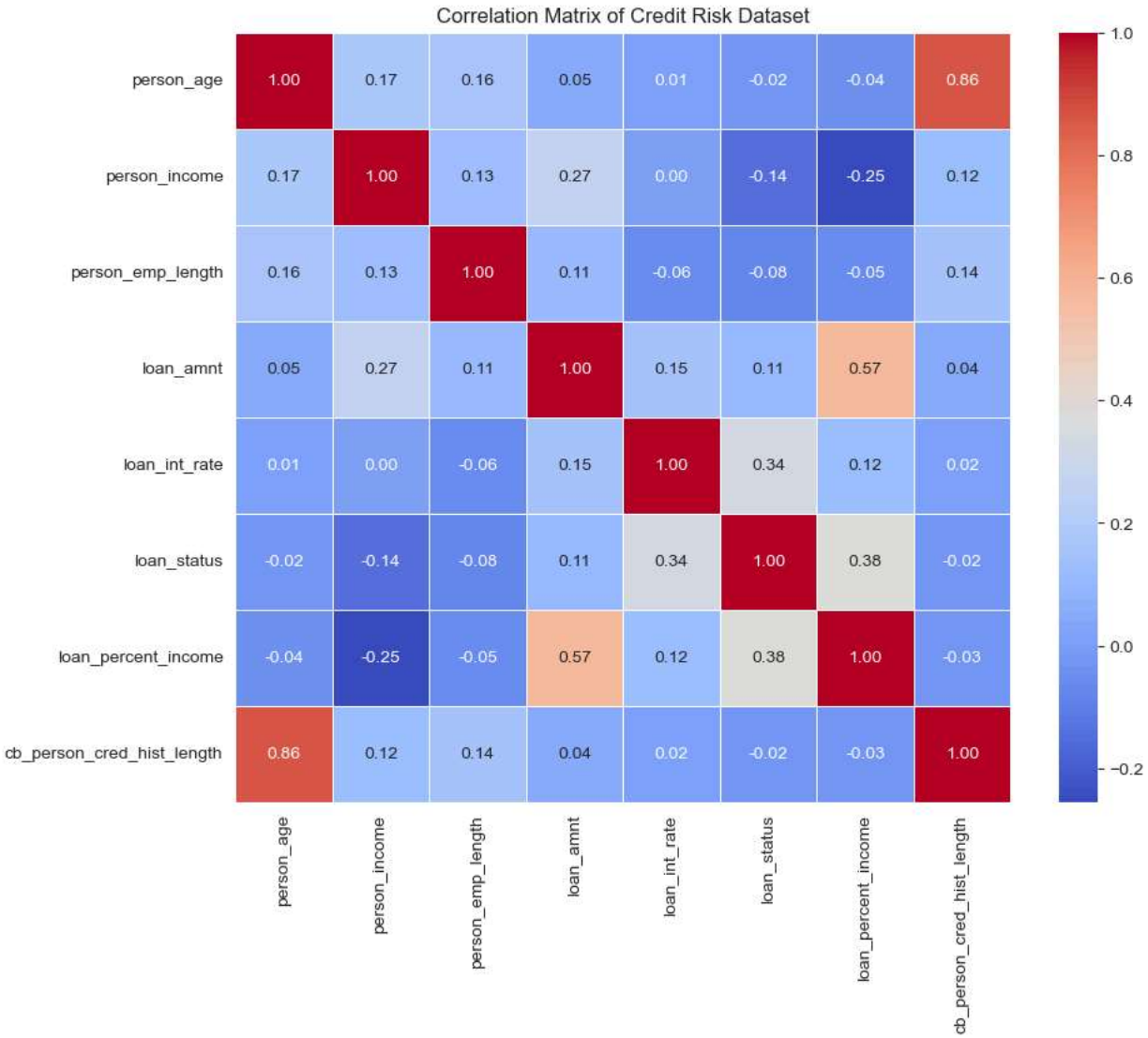
In [13]:
```
sns.pairplot(credit_data_df)
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [14]:

```
numerical_cols = credit_data_df.select_dtypes(include='number')

# Compute the correlation matrix
correlation_matrix = numerical_cols.corr()

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5
plt.title('Correlation Matrix of Credit Risk Dataset')
plt.show()
```

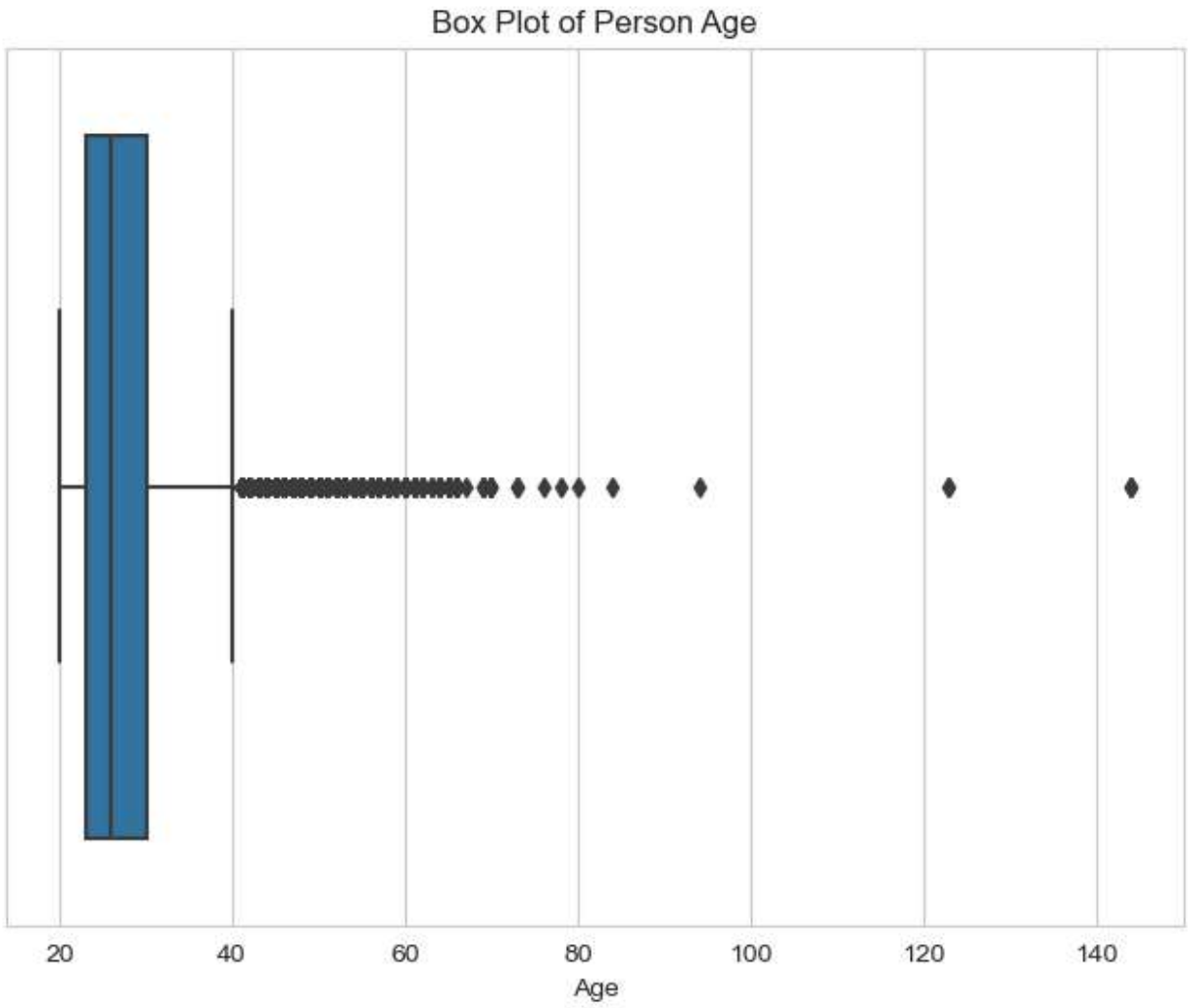


From the above pair plot and correlation matrix , person income and person employment length are not corelated with any of the features and can be dropped

In [15]:
```
credit_data_df_ref1 = credit_data_df.drop(columns=["person_emp_length"])
```

MLS2 - Q2. Perform any data extraction/selection steps.

In [16]:
```
# Generate the box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x=credit_data_df_ref1['person_age'])
plt.title('Box Plot of Person Age')
plt.xlabel('Age')
plt.show()
```



In [17]:
```
credit_data_df_ref1['person_age'].describe()
```

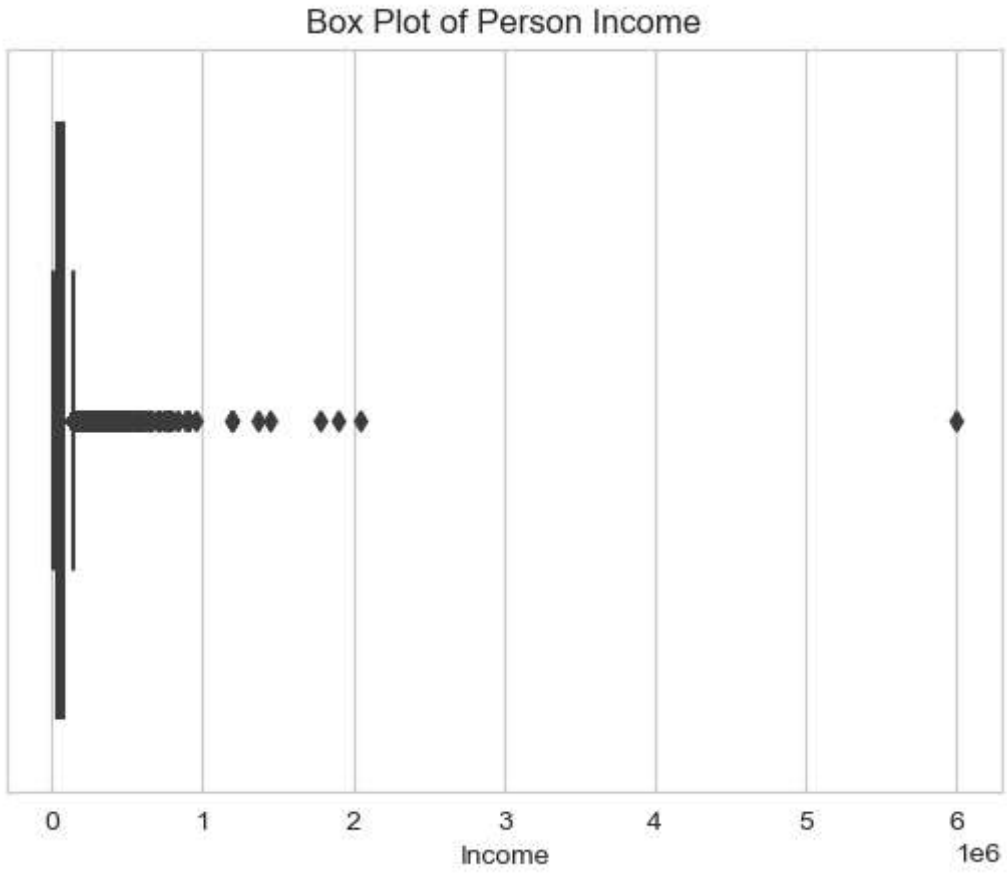
Out[17]:
```
count    32581.000000
mean        27.734600
std          6.348078
min         20.000000
25%         23.000000
50%         26.000000
75%         30.000000
max        144.000000
Name: person_age, dtype: float64
```

From the box plot there are outliers in person's age that may be a typo error. Hence we filter records with age less than

In [18]:
```
credit_data_df_ref2 = credit_data_df_ref1[credit_data_df_ref1['person_age'] < 80]
print("Old dataframe before filter " , credit_data_df_ref1.shape)
print("Old dataframe after filter " , credit_data_df_ref2.shape)
```

```
Old dataframe before filter  (32581, 12)
Old dataframe after filter  (32573, 12)
```

In [19]:
```
sns.boxplot(x=credit_data_df_ref1['person_income'])
plt.title('Box Plot of Person Income')
plt.xlabel('Income')
plt.show()
```



In [20]:
```
credit_data_df_ref2['person_income'].describe()
```

Out[20]:
```
count    3.257300e+04
mean     6.588260e+04
std      5.253665e+04
min      4.000000e+03
25%      3.850000e+04
50%      5.500000e+04
75%      7.920000e+04
max      2.039784e+06
Name: person_income, dtype: float64
```

In [21]:
```
credit_data_df_ref3 = credit_data_df_ref2[credit_data_df_ref2['person_income'] < 80000
print("Old dataframe before filter " , credit_data_df_ref2.shape)
print("Old dataframe after filter " , credit_data_df_ref3.shape)
```

```
Old dataframe before filter  (32573, 12)
Old dataframe after filter  (32556, 12)
```

MLS2 - Q3. Transform features if necessary.

The following features
person_age, person_income, loan_amnt, loan_percent_income and cb_person_cred_hist_length are numerical columns.
But there ranges and magnitudes are different and the high range ones might influence or overflow other low range numerical values. Hence to maintain the same variance but level them with in the same scale, **min max scaler can be used to transform the values to values between 0 to 1.**

In [22]:
```
from sklearn.preprocessing import MinMaxScaler

cat_targt_cols = ["person_home_ownership", "loan_intent", "loan_grade", "loan_status",



numerical_cols = credit_data_df_ref3.select_dtypes(include=['number']).columns.tolist(
numerical_cols.remove("loan_status")
numerical_col_df = credit_data_df_ref3[numerical_cols]
```

In [23]:
```
# standardize the data using min max scaler
scaler = MinMaxScaler()
numerical_df_scaled = scaler.fit_transform(numerical_col_df)
min_max_scaled_df = pd.DataFrame(numerical_df_scaled, columns=numerical_cols)
```

In [24]:
```
# Combine scaled numerical columns with original categorical columns
credit_data_df_scaled = pd.concat([min_max_scaled_df, credit_data_df_ref3[ cat_targt_c
```

In [25]:
```
credit_data_df_scaled
```

Out[25]:

	person_age	person_income	loan_amnt	loan_int_rate	loan_percent_income	cb_person_cred_hist
0	0.034483	0.070876	1.000000	0.595506	0.710843	0
1	0.017241	0.007216	0.014493	0.321348	0.120482	0
2	0.086207	0.007216	0.144928	0.418539	0.686747	0
3	0.051724	0.079253	1.000000	0.551124	0.638554	0
4	0.068966	0.064948	1.000000	0.497191	0.662651	0
...	...	...	...	...	...	
32551	0.637931	0.063144	0.153623	0.434831	0.132530	1
32552	0.586207	0.149485	0.496377	0.116292	0.180723	0
32553	0.775862	0.092784	1.000000	0.312921	0.554217	0
32554	0.620690	0.188144	0.420290	0.340449	0.120482	0
32555	0.793103	0.048969	0.173188	0.256742	0.180723	1

32556 rows × 11 columns

MLS2 - Q4.Engineer new useful features.

In [26]:
```
def add_loan_rate_category(input_df):
    loan_rates = input_df['loan_int_rate']*23.22
    rate_bins = [0, 10, 15, 100]
    rate_labels = ['Low', 'Medium', 'High']
    input_df['loan_rate_category'] = pd.cut(input_df['loan_int_rate'], bins=rate_bins,


add_loan_rate_category(credit_data_df_scaled)
```

In [27]:
```
rate_bins = [0, 10, 15, 100]
rate_labels = ['Low', 'Medium', 'High']
credit_data_df_scaled['loan_rate_category'] = pd.cut(credit_data_df_scaled['loan_int_r
```

In [28]:
```
credit_data_df_scaled
```

Out[28]:

	person_age	person_income	loan_amnt	loan_int_rate	loan_percent_income	cb_person_cred_hist
0	0.034483	0.070876	1.000000	0.595506	0.710843	
1	0.017241	0.007216	0.014493	0.321348	0.120482	
2	0.086207	0.007216	0.144928	0.418539	0.686747	
3	0.051724	0.079253	1.000000	0.551124	0.638554	
4	0.068966	0.064948	1.000000	0.497191	0.662651	
...	...	...	...	...	...	
32551	0.637931	0.063144	0.153623	0.434831	0.132530	
32552	0.586207	0.149485	0.496377	0.116292	0.180723	
32553	0.775862	0.092784	1.000000	0.312921	0.554217	
32554	0.620690	0.188144	0.420290	0.340449	0.120482	
32555	0.793103	0.048969	0.173188	0.256742	0.180723	

32556 rows × 12 columns

MLS2 - Q5. Deal with missing data (do not just drop rows or columns without justifying this).

In [29]:
```
credit_data_df_scaled.isnull().sum()
```

Out[29]:
```
person_age                     0
person_income                  0
loan_amnt                      0
loan_int_rate               3108
loan_percent_income            0
cb_person_cred_hist_length     0
person_home_ownership          0
loan_intent                    0
loan_grade                     0
loan_status                    0
cb_person_default_on_file      0
loan_rate_category          3108
dtype: int64
```

To fill the missng values in loan_int_rate use the median value of Grade value

In [30]:
```
median_loan_int_rate_by_grade = credit_data_df_scaled.groupby('loan_grade')['loan_int_
median_loan_int_rate_by_grade
```

Out[30]:
```
loan_grade
A    0.116292
B    0.312921
C    0.452809
D    0.555618
E    0.640449
F    0.736798
G    0.828090
Name: loan_int_rate, dtype: float64
```

In [31]:
```
def fill_missing_loan_int_rate(row):
    if pd.isnull(row['loan_int_rate']):
        return median_loan_int_rate_by_grade[row['loan_grade']]
    return row['loan_int_rate']

credit_data_df_scaled['loan_int_rate'] = credit_data_df_scaled.apply(fill_missing_loar
```

In [32]:
```
add_loan_rate_category(credit_data_df_scaled)
```

In [33]:
```
credit_data_df_scaled.isnull().sum()
```

Out[33]:
```
person_age                    0
person_income                 0
loan_amnt                     0
loan_int_rate                 0
loan_percent_income           0
cb_person_cred_hist_length    0
person_home_ownership         0
loan_intent                   0
loan_grade                    0
loan_status                   0
cb_person_default_on_file     0
loan_rate_category            0
dtype: int64
```

MLS2 - Q6. Create dummy variables if necessary..

In [34]:
```
#  use one hot encoding to convert categorical values to individual columns
cat_var_list = credit_data_df_ref3.select_dtypes(include=['object']).columns.tolist()
cat_var_list.append("loan_rate_category")
credit_data_df_scaled = pd.get_dummies(credit_data_df_scaled, columns=cat_var_list, dt
```

In [35]:
```
credit_data_df_scaled
```

Out[35]:

	person_age	person_income	loan_amnt	loan_int_rate	loan_percent_income	cb_person_cred_his
0	0.034483	0.070876	1.000000	0.595506	0.710843	
1	0.017241	0.007216	0.014493	0.321348	0.120482	
2	0.086207	0.007216	0.144928	0.418539	0.686747	
3	0.051724	0.079253	1.000000	0.551124	0.638554	
4	0.068966	0.064948	1.000000	0.497191	0.662651	
...	...	...	...	...	...	
32551	0.637931	0.063144	0.153623	0.434831	0.132530	
32552	0.586207	0.149485	0.496377	0.116292	0.180723	
32553	0.775862	0.092784	1.000000	0.312921	0.554217	
32554	0.620690	0.188144	0.420290	0.340449	0.120482	
32555	0.793103	0.048969	0.173188	0.256742	0.180723	

32556 rows × 29 columns

In [36]:
```
credit_data_df_scaled.isnull().sum()
```

Out[36]:
```
person_age                      0
person_income                   0
loan_amnt                       0
loan_int_rate                   0
loan_percent_income             0
cb_person_cred_hist_length      0
loan_status                     0
person_home_ownership_MORTGAGE  0
person_home_ownership_OTHER     0
person_home_ownership_OWN       0
person_home_ownership_RENT      0
loan_intent_DEBTCONSOLIDATION   0
loan_intent_EDUCATION           0
loan_intent_HOMEIMPROVEMENT     0
loan_intent_MEDICAL             0
loan_intent_PERSONAL            0
loan_intent_VENTURE             0
loan_grade_A                    0
loan_grade_B                    0
loan_grade_C                    0
loan_grade_D                    0
loan_grade_E                    0
loan_grade_F                    0
loan_grade_G                    0
cb_person_default_on_file_N     0
cb_person_default_on_file_Y     0
loan_rate_category_Low          0
loan_rate_category_Medium       0
loan_rate_category_High         0
dtype: int64
```

Term Project Milestone 3

Saravanan Janarthanan

In Milestone 3, you will begin the process of selecting, building, and evaluating a model. You are required to train and evaluate at least one model in this milestone. Write step-by-step for performing each of these steps. You can use any methods/tools you think are most appropriate, but you should explain/justify why you are selecting the model(s) and evaluation metric(s) you choose. It is important to think about what type of model and metric makes sense for your problem. Again, do what makes the most sense for your project. Write a short overview/conclusion of the insights gained from your model building/evaluation.

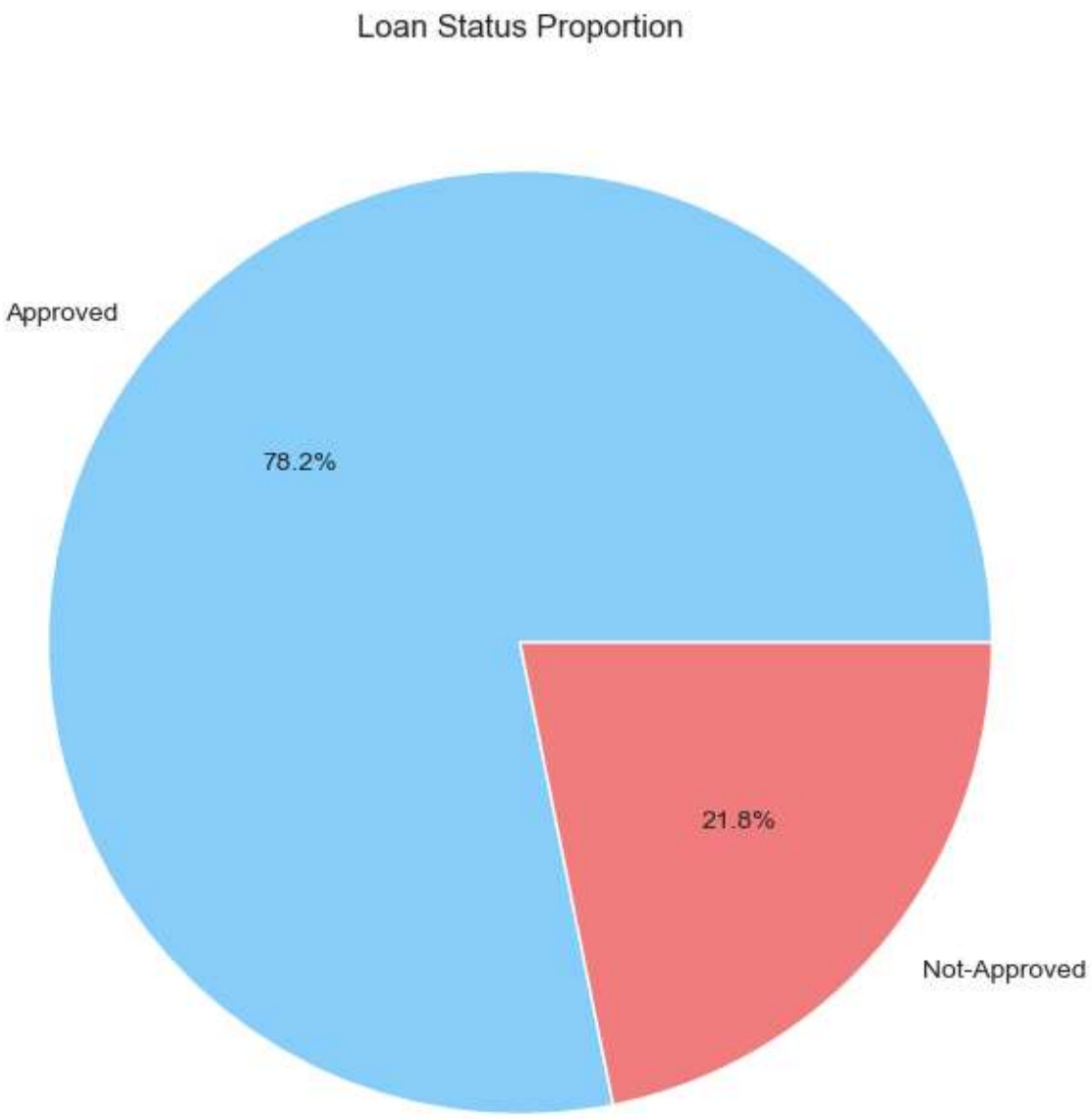
It is important to note that these milestones are meant to keep you on track for the final project submission. At any point, you can pivot or modify your project as needed based on what you discover. These milestones are not final versions; they are drafts of the many steps you need to complete along the way.

As a reminder, Teams is a great place to discuss your project with your peers. Feel free to solicit feedback/input (without creating a group project!) and collaborate on your projects with your peers.

Each milestone will build on top of each other, so make sure you do not fall behind. Submit Milestones 1-3 together. I recommend building your project milestones in a Jupyter Notebook, building upon one another. However, make sure it is clear where each milestone begins and ends.

Balance the imbalanced data

In [37]:
```
# Check the current target data propotion
plt.figure(figsize=(8, 8))
loan_status_counts = credit_data_df_scaled['loan_status'].value_counts()
plt.pie(loan_status_counts, labels=['Approved', 'Not-Approved'], autopct='%1.1f%%', cc
plt.title('Loan Status Proportion')
plt.show()
```



The loan status target variable is unbalanced. Hence need to balance the same. hence use SMOTE to balance the data SMOTE - Synthetic Minority Over-sampling Technique, SMOTE generates synthetic samples for the minority class to achieve a balanced dataset.

In [38]:
```
#from imblearn.utils._validation import _check_X
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Splitting the data into features (X) and target (y)
X = credit_data_df_scaled.drop('loan_status', axis=1)
y = credit_data_df_scaled['loan_status']

# Splitting into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

print(y_train.value_counts())
```

```
loan_status
0    20393
1     5651
Name: count, dtype: int64
```

In [39]:
```
# Apply SMOTE to balance the training data
smote = SMOTE(random_state=500)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)
```

In [40]:
```
print(y_train_balanced.value_counts())
```

```
loan_status
0    20393
1    20393
Name: count, dtype: int64
```

In [41]:
```
print("X_train balanced Length : ", len(X_train_balanced) , "   y_train balanced lengt
```

```
X_train balanced Length :  40786    y_train balanced length :  40786
```

The data is balanced now

In [42]:
```
X_train_balanced
```

Out[42]:

	person_age	person_income	loan_amnt	loan_int_rate	loan_percent_income	cb_person_cred_his
0	0.137931	0.048969	0.263768	0.139326	0.277108	
1	0.051724	0.052835	0.171014	0.555618	0.168675	
2	0.086207	0.051546	0.333333	0.512360	0.325301	
3	0.034483	0.021134	0.079710	0.496629	0.192771	
4	0.051724	0.095361	0.118841	0.125843	0.072289	
...	...	...	...	...	...	
40781	0.134808	0.099766	0.565217	0.314295	0.310026	
40782	0.049450	0.033361	0.206044	0.595146	0.317356	
40783	0.034483	0.030488	0.180984	0.556186	0.290280	
40784	0.078574	0.047787	0.352351	0.475199	0.365398	
40785	0.163471	0.043325	0.064950	0.297231	0.095048	

40786 rows × 28 columns

Train and Evaluate the model

Steps in training and evaluating the model

Train the model

- Split the data for training and test
- Choose a Alogorithm to train the model

- Fit the model
- Evaluate the model and find the metrics
- Use hyper tuning parameters to fine tune and identify best optimal model
- Use GridSearchCV to find the optimal model using hyper tuning model paramter grid
- Find the evlauation metrics from the optinal model
- summarize and find the best model

Split the data for training and test Since the split was already done while standardizing and adding dummy columns this step is already completed

Choose a Alogorithm to train the model The below algorithms will be used

- Logistic Regression: This algorithm is ideal for binary classification problems. Given that the target variable, "loan_status," is binary, logistic regression is a well-suited choice.
- Random Forest Classifier: This model uses multiple decision trees and averages their outputs, which helps in achieving high accuracy and mitigating overfitting. It is a robust choice for this classification task.
- Gradient Boosting - XGB Classifier: XGBoost is known for its high accuracy and ability to handle complex datasets effectively. It is selected for its superior performance in various classification tasks.
- Support Vector Machine (SVM): Although SVMs are particularly effective in text classification, they are included in this analysis to evaluate their metrics and determine their suitability for the given dataset.

Logisitc Regression

In [43]:
```
# Load the modules for logisitc algorithm and evaluation metrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c
```

In [44]:
```
model_results = {}
model_names = []
```

In [45]:
```
# Create and train the logistic regression model
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_balanced, y_train_balanced)
```

Out[45]:
```
LogisticRegression(max_iter=1000)
```

In [46]:
```
# Define a common resuable function to calculate Acuracy score, Precision score,  Reca
# Results are returned as a dictionary
def evaluvateModel(model, x_test_data, y_test_data):
    #  prediction the model
    y_pred_mdl = model.predict(x_test_data)

    # Evaluate the model and get metrics
```

```
    accuracy = accuracy_score(y_test_data, y_pred_mdl)
    precision = precision_score(y_test_data, y_pred_mdl)
    recall = recall_score(y_test_data, y_pred_mdl)
    f1 = f1_score(y_test_data, y_pred_mdl)
    conf_matrix = confusion_matrix(y_test_data, y_pred_mdl)

    return {"Prediction Values" : y_pred_mdl,
            "Accuracy Score" : accuracy,
            "Precision Score": precision,
            "Recall Score" : recall,
            "F1 Score": f1,
            "Confusion Matrix": conf_matrix
           }
```

In [47]:
```
# Evaluate and print the score metrics for logisitic regression
Log_model_results = evaluvateModel(log_model, X_test,  y_test)
Log_model_results
```

Out[47]:
```
{'Prediction Values': array([1, 1, 0, ..., 1, 0, 0], dtype=int64),
 'Accuracy Score': 0.8181818181818182,
 'Precision Score': 0.5688350983358548,
 'Recall Score': 0.7741935483870968,
 'F1 Score': 0.6558139534883721,
 'Confusion Matrix': array([[4200,  855],
        [ 329, 1128]], dtype=int64)}
```

Following Hyper tuning parameters can be used to find the optimal model for accuracy for Logistic regression

- C : The value of C helps prevent overfitting. Lower values of C correspond to higher regularization, which means the model will penalize more complex models.
- solver: This parameter specifies the algorithm to use for optimization. Different solvers have different characteristics and may perform differently depending on the dataset.
- penalty : This parameter specifies the norm used in the penalization (regularization). NOT used as there were erros thrown during gridsearch cv
- class_weight: This parameter adjusts the weights associated with classes to handle class imbalance. Since SMOTE was used to balance the data this was NOT used to further tune if needed

In [48]:
```
# define the hyper parameters for Logistic Regression
search_space_log_reg = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
}
```

In [49]:
```
# Load Grid Search CV module
from sklearn.model_selection import GridSearchCV

# execute the model through Grid Search CV and fit the same to find the optimal model
grid_search_log_reg = GridSearchCV(log_model, search_space_log_reg, cv=5, n_jobs=-1, s
grid_search_log_reg.fit(X_train_balanced, y_train_balanced)
```

Out[49]:

```
GridSearchCV
best_estimator_: LogisticRegression
LogisticRegression
```

In [50]:
```
grid_search_log_reg.best_estimator_
```

Out[50]:

```
LogisticRegression
LogisticRegression(C=10, max_iter=1000)
```

In [51]:
```
# Use the best optimized Logistic Regression model and evaluate the metrics
Log_model_results_cv = evaluvateModel(grid_search_log_reg.best_estimator_, X_test,  y_
Log_model_results_cv
```

Out[51]:
```
{'Prediction Values': array([1, 1, 0, ..., 1, 0, 0], dtype=int64),
 'Accuracy Score': 0.8177211302211302,
 'Precision Score': 0.56797583081571,
 'Recall Score': 0.7741935483870968,
 'F1 Score': 0.6552425210572176,
 'Confusion Matrix': array([[4197,  858],
        [ 329, 1128]], dtype=int64)}
```

In [52]:
```
# Add the Optimal Logistic Regression model results results to a dictionary
model_results["Logsitic Regression"] = Log_model_results_cv
model_names.append("Logsitic Regression")
```

Support Vector Machine

In [53]:
```
from sklearn.svm import SVC

# Create and train the SVM model
model_svc = SVC(kernel='linear', C=1, random_state=500)
model_svc.fit(X_train_balanced, y_train_balanced)
```

Out[53]:

```
SVC
SVC(C=1, kernel='linear', random_state=500)
```

In [54]:
```
# Evaluate and print the score metrics for Support vector machine
svc_model_results = evaluvateModel(model_svc, X_test,  y_test)
svc_model_results
```

Out[54]:
```
{'Prediction Values': array([1, 1, 0, ..., 1, 0, 0], dtype=int64),
 'Accuracy Score': 0.8286240786240786,
 'Precision Score': 0.5905469994689325,
 'Recall Score': 0.7632120796156486,
 'F1 Score': 0.665868263473054,
 'Confusion Matrix': array([[4284,  771],
        [ 345, 1112]], dtype=int64)}
```

Following Hyper tuning parameters can be used to find the optimal model for accuracy for SVM Support Vecotr Machine

- C : Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.
- kernel : Specifies the kernel type to be used in the algorithm. The kernel transforms the data into a higher-dimensional space.
- gamma : Kernel coefficient used based on the kernel chosen

In [55]:
```
# define the hyper parameters for Support Vector Machine
search_space_svc_reg = {'kernel': ['poly', 'rbf' ]}
```

In [56]:
```
# execute the model through Grid Search CV and fit the same to find the optimal model
grid_search_svc_reg = GridSearchCV(model_svc, search_space_svc_reg, cv=5, n_jobs=-1, s
grid_search_svc_reg.fit(X_train_balanced, y_train_balanced)
```

Out[56]:

GridSearchCV

best_estimator_: SVC

SVC

In [57]:
```
grid_search_svc_reg.best_estimator_
```

Out[57]:

SVC

```
SVC(C=1, kernel='poly', random_state=500)
```

In [58]:
```
# Use the best optimized Support Vector Machine model and evaluate the metrics
svc_model_results_cv = evaluvateModel(grid_search_svc_reg.best_estimator_, X_test,  y_
svc_model_results_cv
```

Out[58]:
```
{'Prediction Values': array([0, 0, 0, ..., 1, 0, 0], dtype=int64),
 'Accuracy Score': 0.8679361179361179,
 'Precision Score': 0.6907348242811502,
 'Recall Score': 0.7419354838709677,
 'F1 Score': 0.7154202514890801,
 'Confusion Matrix': array([[4571,  484],
        [ 376, 1081]], dtype=int64)}
```

In [59]:
```
# Add the Optimal support Vector Machine model results results to a dictionary
model_results["Support Vector Machine"] = svc_model_results_cv
model_names.append("Support Vector Machine")
```

Random Forest Classifier

In [60]:
```
from sklearn.ensemble import RandomForestClassifier
```

```
# Train the RandomForestClassifier
model_rf = RandomForestClassifier(n_estimators=100, random_state=500)
model_rf.fit(X_train_balanced, y_train_balanced)
```

Out[60]:

RandomForestClassifier

RandomForestClassifier(random_state=500)

In [61]:
```
# Evaluate and print the score metrics for Random Forest Classifier
rf_model_results = evaluvateModel(model_rf, X_test,  y_test)
rf_model_results
```

Out[61]:
```
{'Prediction Values': array([0, 0, 0, ..., 0, 0, 0], dtype=int64),
 'Accuracy Score': 0.9113943488943489,
 'Precision Score': 0.8389830508474576,
 'Recall Score': 0.7474262182566919,
 'F1 Score': 0.790562613430127,
 'Confusion Matrix': array([[4846,  209],
        [ 368, 1089]], dtype=int64)}
```

Following Hyper tuning parameters can be used to find the optimal model for accuracy for Random Forest Classifier

- n_estimators: Number of trees in the forest. More trees can improve performance but will increase computational cost.
- max_depth: Maximum depth of the tree. Controls how deep the tree can grow.
- min_samples_leaf: Minimum number of samples required to be at a leaf node. Helps prevent overfitting.
- min_samples_split: Minimum number of samples required to split an internal node. Controls the complexity of the tree. NOT used in below GridsearchCV
- max_features: Number of features to consider when looking for the best split. Controls the randomness of each tree. NOT used in below GridsearchCV
- bootstrap: Whether bootstrap samples are used when building trees. Controls the sampling method. NOT used in below GridsearchCV

In [62]:
```
# define the hyper parameters for Random Forest Classifier
search_space_rf = {
    'n_estimators': [ 500, 750, 1000],
    'max_depth': [None, 10, 20],
    'min_samples_leaf': [1, 2, 4]
}
```

In [63]:
```
# execute the model through Grid Search CV and fit the same to find the optimal model
grid_search_rf = GridSearchCV(model_rf, search_space_rf, cv=5, n_jobs=-1, scoring='acc
grid_search_rf.fit(X_train_balanced, y_train_balanced)
```

Out[63]:

GridSearchCV

best_estimator_: RandomForestClassifier

RandomForestClassifier

In [64]:
```
grid_search_rf.best_estimator_
```

Out[64]:
```
RandomForestClassifier
RandomForestClassifier(n_estimators=1000, random_state=500)
```

In [65]:
```
# Use the best optimized Random Forest Classifier  model and evaluate the metrics
rf_model_results_cv = evaluvateModel(grid_search_rf.best_estimator_, X_test,  y_test)
rf_model_results_cv
```

Out[65]:
```
{'Prediction Values': array([0, 1, 0, ..., 0, 0, 0], dtype=int64),
 'Accuracy Score': 0.9123157248157249,
 'Precision Score': 0.8376524390243902,
 'Recall Score': 0.7542896362388469,
 'F1 Score': 0.7937883712531599,
 'Confusion Matrix': array([[4842,  213],
        [ 358, 1099]], dtype=int64)}
```

In [66]:
```
# Add the Optimal Random Forest Classifier  model results results to a dictionary
model_results["Random Forest Classifier"] = rf_model_results_cv
model_names.append("Random Forest Classifier")
```

XGBClassifier

In [67]:
```
from xgboost import XGBClassifier

# Train the XGBoost classifier
model_xgb= XGBClassifier( eval_metric='logloss')
model_xgb.fit(X_train_balanced, y_train_balanced)
```

Out[67]:
```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators
```

In [68]:
```
#!pip install xgboost
```

In [69]:
```
# Evaluate and print the score metrics for XGBoost classifier
xgb_model_results = evaluvateModel(model_xgb, X_test,  y_test)
xgb_model_results
```

Out[69]:
```
{'Prediction Values': array([0, 1, 0, ..., 0, 0, 0]),
 'Accuracy Score': 0.9216830466830467,
 'Precision Score': 0.8989048020219039,
 'Recall Score': 0.7323266986959506,
 'F1 Score': 0.8071104387291982,
 'Confusion Matrix': array([[4935,  120],
        [ 390, 1067]], dtype=int64)}
```

Following Hyper tuning parameters can be used to find the optimal model for accuracy for XGBoost Classifier

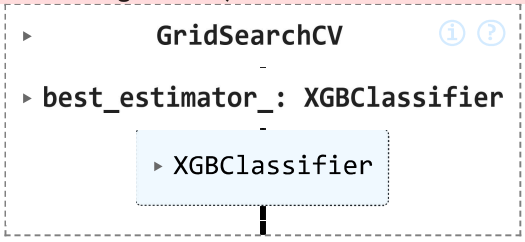
- n_estimators: Number of boosting rounds. More rounds can improve performance but will increase training time.
- learning_rate: Step size shrinkage used in updating to prevent overfitting.
- reg_alpha: L1 regularization term on weights. Helps prevent overfitting.
- reg_lambda: L2 regularization term on weights. Helps prevent overfitting.
- subsample: Subsample ratio of the training instances. Prevents overfitting by sampling a fraction of the data.
- gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree.

In [70]:
```
# define the hyper parameters for XGBooster Classifier
search_space_xgb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'reg_alpha': [0, 0.01, 0.1],
    'reg_lambda': [1, 1.5, 2],
    'subsample': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2]
}
```

In [71]:
```
# execute the model through Grid Search CV and fit the same to find the optimal model
grid_search_xgb = GridSearchCV(model_xgb, search_space_xgb, cv=5, n_jobs=-1, scoring='
grid_search_xgb.fit(X_train_balanced, y_train_balanced)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\joblib\externals\loky\process_executor.py:
700: UserWarning: A worker stopped while some jobs were given to the executor. This c
an be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
```

Out[71]:



In [72]:
```
grid_search_xgb.best_estimator_
```

Out[72]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=0, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.2, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators
```

In [73]:

```
# Use the best optimized XGBooster Classifier model and evaluate the metrics
xgb_model_results_cv = evaluvateModel(grid_search_xgb.best_estimator_, X_test,  y_test
xgb_model_results_cv
```

Out[73]:

```
{'Prediction Values': array([0, 0, 0, ..., 0, 0, 0]),
 'Accuracy Score': 0.9227579852579852,
 'Precision Score': 0.9015151515151515,
 'Recall Score': 0.7350720658888126,
 'F1 Score': 0.8098298676748582,
 'Confusion Matrix': array([[4938,  117],
        [ 386, 1071]], dtype=int64)}
```

In [74]:

```
# Add the Optimal XGBooster Classifier  model results results to a dictionary
model_results["XGBooster Classifier"] = xgb_model_results_cv
model_names.append("XGBooster Classifier")
```

Results and Summary

In [75]:

```
y_test.value_counts()
```

Out[75]:

```
loan_status
0    5055
1    1457
Name: count, dtype: int64
```

In [76]:

```
# Print the results from all the optimized models to compare
print("Based on training models the  findings from  the best optinal model among the 4
print("=" * 125)
print(f"* {'Model Name':^25} * {'Accuracy':^9} * {'Precision':^9} * {'Recall':^9} * {'
print("=" * 125)
for mdl_nm in model_names:
    res_dic = model_results.get(mdl_nm)
    acc_scr = str( res_dic.get('Accuracy Score') )[:4]
    prec_scr = str( res_dic.get('Precision Score') )[:4]
    recall_scr = str( res_dic.get('Recall Score') )[:4]
    f1_scr = str( res_dic.get('F1 Score') )[:4]
    conf_mtx = res_dic.get('Confusion Matrix')
    print(f"* {mdl_nm:<25} * {acc_scr:>9} * {prec_scr:>9} * {recall_scr:>9} * {f1_scr:
print("=" * 125, "\n")
```

```
Based on training models the  findings from  the best optinal model among the 4 is li
sted below

================================================================================
========================================
*        Model Name         * Accuracy  * Precision *  Recall   * F1 Score  * True Po
s  * True Neg  * False Pos * False Neg *
================================================================================
========================================
* Logsitic Regression       *      0.81 *      0.56 *      0.77 *      0.65 *      11
28 *      4197 *       329 *       858 *
* Support Vector Machine    *      0.86 *      0.69 *      0.74 *      0.71 *      10
81 *      4571 *       376 *       484 *
* Random Forest Classifier  *      0.91 *      0.83 *      0.75 *      0.79 *      10
99 *      4842 *       358 *       213 *
* XGBooster Classifier      *      0.92 *      0.90 *      0.73 *      0.80 *      10
71 *      4938 *       386 *       117 *
================================================================================
========================================
```

Evaluation Metrics

The following metrics are chosen for evaluating the performance of classification models due to their ability to provide comprehensive insights:

- Accuracy: Measures the ratio of correctly predicted instances to the total instances. This metric gives a quick snapshot of the model's overall performance across all classes.
- Precision: Represents the ratio of correctly predicted positive observations to the total predicted positives. Precision is crucial when the cost of false positives is high. For example, in spam detection, high precision ensures that fewer legitimate emails are incorrectly marked as spam.
- Recall: Indicates the ratio of correctly predicted positive observations to all observations in the actual positive class. Recall is essential when the cost of false negatives is high. For instance, in disease screening, high recall ensures that most actual positive cases are identified, reducing the risk of missing true positive cases.
- F1 Score: The harmonic mean of precision and recall. The F1 score balances both precision and recall, making it useful as a single metric for evaluating model performance, particularly in cases of class imbalance.
- Confusion Matrix: A table that describes the performance of a classification model by displaying the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The confusion matrix provides detailed insights into the model's performance, helping to diagnose specific issues, such as whether the model is more prone to false positives or false negatives.

These metrics collectively offer a well-rounded evaluation of a model's performance, highlighting different aspects and helping to identify areas for improvement.

Based on the evaluation metrics, the XGBoost Classifier excels in several aspects:

- **Accuracy**: The XGBoost Classifier has a high accuracy of 92%, indicating that it correctly predicted 92% of the test values. The Random Forest Classifier is close behind with 91% accuracy.
- **Precision**: With a precision of 90%, the XGBoost Classifier demonstrates that it correctly identified 90% of the positive predictions, effectively reducing the false positive rate..
- **Recall**: The recall score for XGBoost is 73%, which is lower than the optimal standard and slightly less than Logistic Regression's recall of 77%. This indicates that while XGBoost has a higher false negative rate, it is still competitive.
- **F1 Score**: The F1 score, balancing precision and recall, is 80% for XGBoost, higher than the other models, suggesting it better identifies positive cases.
- **Confusion Matrix**: The confusion matrix shows that the XGBoost Classifier has higher true positive and true negative predictions compared to other models, indicating better overall prediction capability.

In conclusion, the XGBoost Classifier outperforms other optimized models, demonstrating superior accuracy, precision, and balanced performance.

The comprehensive evaluation indicates that the XGBoost Classifier is the most optimized model for this classification task, providing the best balance among accuracy, precision, recall, and F1 score. Its ability to effectively handle class imbalances and minimize both false positives and false negatives makes it the preferred choice. However, the Random Forest Classifier is also a strong performer, with metrics only slightly lower than XGBoost. Support Vector Machine and Logistic Regression are suitable for specific scenarios where their particular strengths (higher recall for Logistic Regression and balanced performance for SVM) are advantageous.

Overall, selecting the right model depends on the specific needs of the application, whether prioritizing reducing false positives, minimizing false negatives, or balancing both effectively.

Project Learning

This project provides a comprehensive understanding of developing a business solution using data science and machine learning. It encompasses various critical stages, including:

- Data Cleaning: Preparing raw data by handling missing values, correcting inconsistencies, and removing noise.
- Data Transformation: Modifying data formats and types to suit analytical needs.
- Data Analysis: Exploring and interpreting data to identify patterns and insights.
- Feature Engineering: Creating new features or modifying existing ones to improve model performance.
- Standardizing the Data: Ensuring that data is on a comparable scale.
- Balancing the Data: Addressing class imbalances to prevent biased model outcomes.
- Building and Training the Model: Developing and fitting the model to the training data.
- Evaluating the Model: Assessing model performance using various metrics.

The project also delves into hyperparameter tuning, understanding computational constraints, and fine-tuning to ensure the model's performance is not significantly impacted by infrastructure limitations. This holistic approach enhances the ability to develop and deploy robust machine learning models in real-world business applications.

In [ ]: