# Predicting Loan Default on Multifamily Real Estate

Patricio Martinez, Logan Van Dine, and Victoria (Tori) Widjaja

Shiley-Marcos School of Engineering, University of San Diego

ADS 504 Machine Learning and Deep Learning for Data Science

## Abstract

Loan defaults on multifamily real estate pose significant financial risks to lenders and the broader housing market. In 2023, Fannie Mae’s multifamily loan acquisitions totaled an unpaid principal balance of $52.9 billion, with the serious delinquency rate peaking at 0.54%. This study seeks to address these risks by developing a predictive model for loan defaults, utilizing data from Fannie Mae’s public repository. The model focuses on identifying loans at risk of becoming 60+ days delinquent, a critical threshold indicating potential default. After extensive data preparation, including the removal of features prone to information leakage and addressing class imbalance through the Synthetic Minority Over-sampling Technique (SMOTE), three machine learning models—Logistic Regression, K-Nearest Neighbors (KNN), and Support Vector Machines (SVM)—were trained and evaluated. The KNN model outperformed others, demonstrating the highest precision in predicting loan defaults, albeit with some limitations due to the unbalanced dataset. Future research should explore more advanced models and incorporate macroeconomic variables to enhance predictive performance. This study provides a foundation for leveraging machine learning to mitigate financial risks associated with multifamily loan defaults, offering valuable insights for Fannie Mae and similar institutions.


*Keywords:* Loan Default Prediction, Multifamily Real Estate, Machine Learning, Fannie Mae, Synthetic Minority Over-sampling Technique (SMOTE)

# Predicting Loan Default on Multifamily Real Estate

Defaulting on a loan for multifamily real estate can pose a significant financial risk for lenders and the broader housing market. In 2023, Fannie Mae had a total Unpaid Principal Balance of $52.9 billion for multifamily loan acquisitions, which was part of their overall multifamily portfolio (Fannie Mae, 2024). While only a small percentage of loans typically result in defaults, the potential for significant losses remains due to the substantial dollar amounts associated with these loan types. As of September 30, 2023, the serious delinquency rate for multifamily loans reached its peak at 0.54%, but it decreased to 0.46% by the end of the year due to actions taken to minimize losses, such as foreclosure and loss mitigation activities (Fannie Mae, 2024). Tackling this challenge requires innovative methods to predict potential defaults early on and accurately. This study aims to develop a robust predictive model for loan defaults on multifamily real estate by using data from Fannie Mae's public Data Dynamics repository. The goal is to identify loans at risk of becoming 60+ days delinquent, an indicator if the loan was ever 60 days or greater delinquent, enabling proactive measures to reduce financial losses and stabilize the housing market (Fannie Mae, 2024).

## Methodology

The data obtained in this study is sourced from Fannie Mae’s public website, Data Dynamics, which houses historical information on housing acquisitions and loan performance. The dataset included numeric, categorical, and date data types, including details on the loan type, interest rates, amortization, and historical events like foreclosures, liquidations, and default information. The data was paired down from over 4,628,626 records to 56,643 after removing duplicates and loans acquired in 2020 or later since they may not have had enough time to potentially default. The reporting date ranged from January 2000 through December 2023. The target feature for the selected dataset is the binary field “Loan Ever 60+ Days Delinquent”. All columns with information that would not be available at the time of acquisition were removed to avoid any information leakage and ensure the validity of our predictive model. By excluding features that would not have been known or accessible at the time of loan acquisition, the aim is to build a model that accurately reflects real-world conditions and predicts loan delinquency based on information that would have been available when the loan was issued without introducing the unconscious bias of economic downfall periods.

## Exploratory Data Analysis

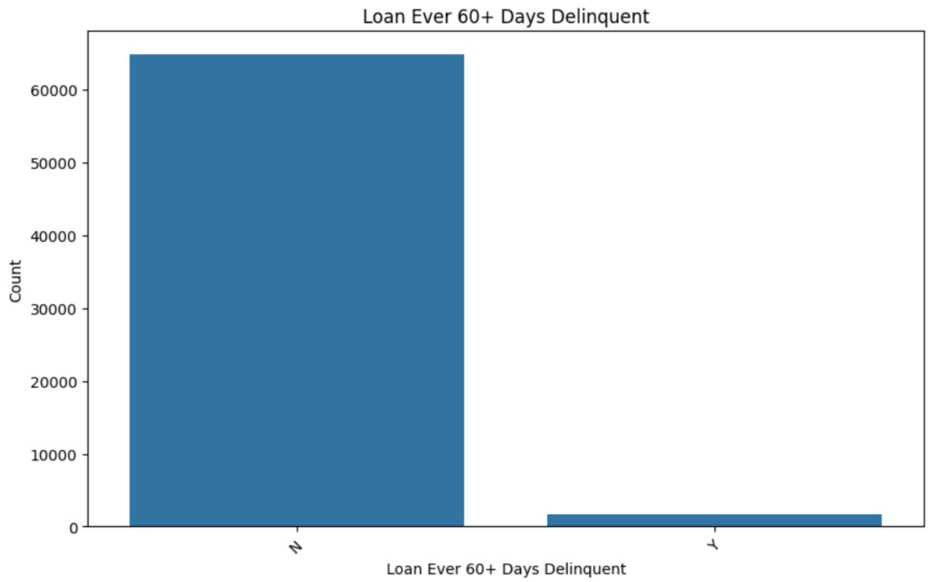
After completing the initial analysis of features to be included and those to be removed, the team explored the complete data further. At the time of data exploration, after transformations, the Fannie Mae data contains just over 66,000 records of 26 features. The features are separated into two data frames based on the data type, categorical or numeric.

### Target Distributions

To begin, the team delves deeper into the target variable “Loan Ever 60+ Days Delinquent”. Figure 1 represents the distribution of the binary target variable. It is clear that the target variable is highly unbalanced. An unbalanced target variable poses a severe risk of bias to the majority class, in this case, the negative class, in model performance (Marini Systems, 2022). This risk will have to be mitigated during the model preprocessing by utilizing the Synthetic Oversampling Minority Technique, or SMOTE.

**Figure 1**

*Unbalanced Distribution of Target Variable*

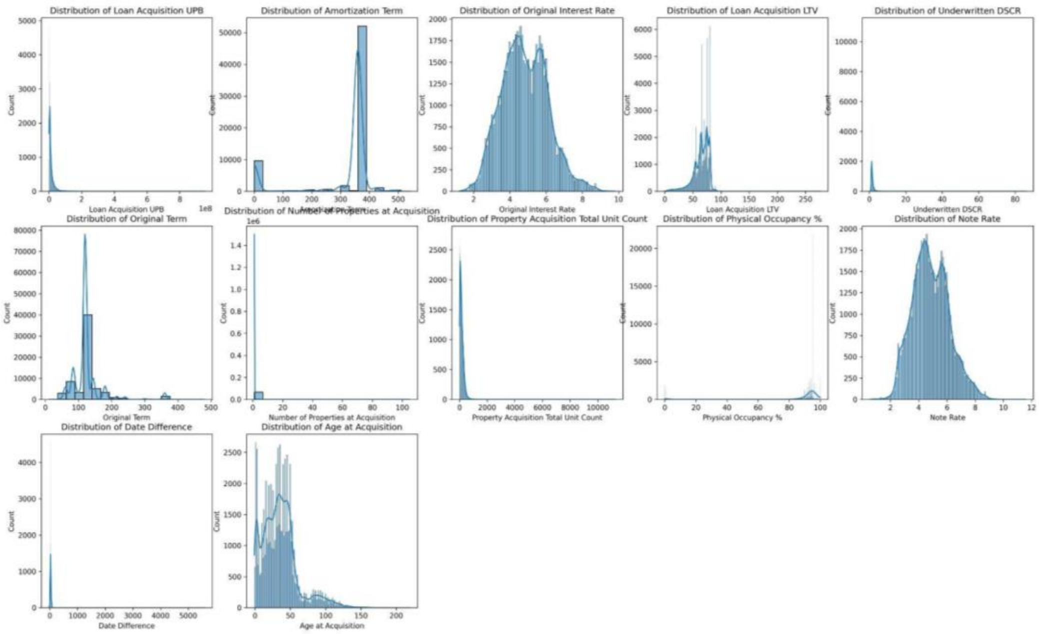



## Numeric Distributions and Correlations

In addition to the target variable distribution, it is also necessary to visualize the distribution of all numeric features to determine if further model preparation is needed in the pipeline to be created. Figure 2 represents the histograms of the twelve numeric features analyzed in the data. The histograms show that most numeric features are right-skewed or positively skewed. Only two variables appear to be regularly distributed: original interest note and note rate. The numeric features will require standardization before models are trained. In addition to the distributions, the team also analyzes the relationships between each numeric variable and others.

**Figure 2**

*Histograms of Numeric Features*




A correlation test is completed to determine if multicollinearity or variables too closely related are potential risks in model development. Figure 3 captures the Fannie Mae dataset's correlation matrix of numeric variables. The only relationship posing a risk of multicollinearity is that of the “Original Interest Rate” and “Note Rate,” as both variables contain financial interest information records. Table 1 presents the correlation coefficients of a biserial correlation test to determine the relationship between all numeric variables and the target variable, “Loan Ever 60+ Days Delinquent”. With its relation to the target variable and “Original Interest Rate,” the “Note Rate” feature is removed. As there are no notable strong relations to the target variable in the numeric variables, all other numeric variables are included in initial model training.

**Figure 3**

*Correlation Matrix of Numeric Variables*

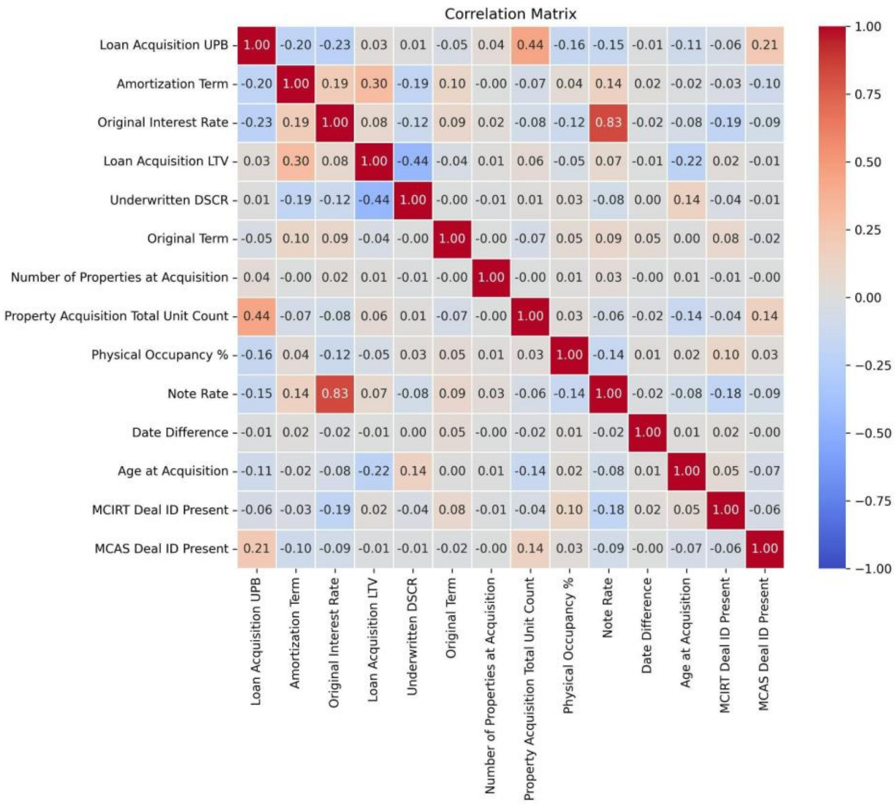



**Table 1**

*Biserial Correlation Coefficients: Target vs. Numeric*

| Feature | Correlation |
|---|---:|
| Loan Acquisition UPB | -0.02878946024 |
| Amortization Term | 0.05947423117 |
| Original Interest Rate | 0.1012866047 |
| Loan Acquisition LTV | 0.08954661332 |
| Underwritten DSCR | -0.03334909583 |
| Original Term | 0.007612170092 |
| Number of Properties at Acquisition | 0.001285869757 |
| Property Acquisition Total Unit Count | -0.02097665214 |
| Physical Occupancy % | -0.01026586182 |
| Note Rate | 0.09265011763 |
| Date Difference | 0.02883836116 |
| Age at Acquisition | 0.007437165594 |

In order to properly select features of every data type for modeling, the team runs a similar test to compare the relation between the target variable and categorical variables. Table 2 represents the correlation coefficients of the strength of the relationship between each categorical variable and the target by a Cramer's V Chi-Square test. Similar to the numeric variables, no categorical variables have significantly strong relationships to the target. Utilizing the same logic, the team will include all categorical variables in model testing.

**Table 2**

*Cramer's V Chi-Square Coefficients: Target vs. Categorical*

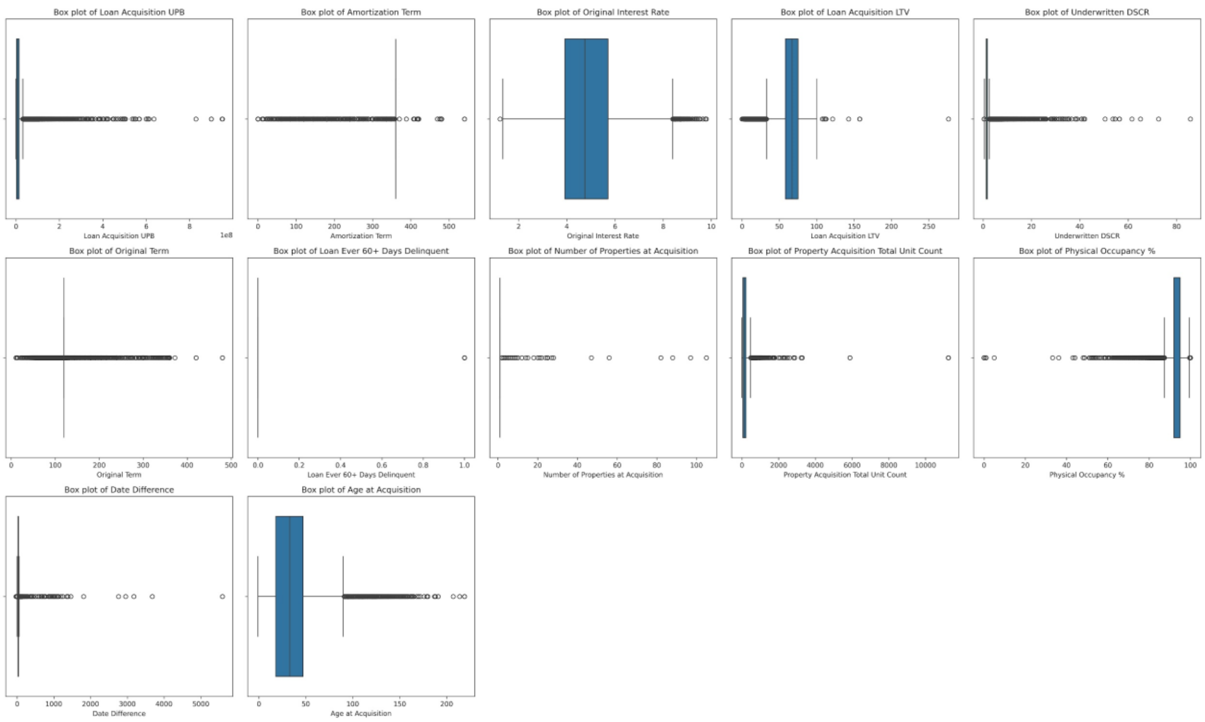
| Feature | Cramér's V Chi Square |
|---|---|
| Amortization Type | 0.06887541774 |
| Interest Type | 0.006410915218 |
| Loan Product Type | 0.01817016219 |
| Lien Position | 0.01407024632 |
| Underwritten DSCR Type | 0.09350729837 |
| Loss Sharing Type | 0.03306320366 |
| Specific Property Type | 0.0709858782 |
| MCIRT Deal ID | 0.03820287534 |
| MCAS Deal ID | 0.01474952907 |
| MCIRT Deal ID Present | 0.03421853938 |
| MCAS Deal ID Present | 0.01474334954 |
| Prepayment Provision Category | 0.03787858845 |

## Outliers

The final test completed to provide a thorough analysis of the data is to determine if outliers are present in the numeric data. Visually, all numeric features appear to have outliers outside of the outer whiskers in the box-and-whisker plots of Figure 4. Z-score detection was used with a common threshold of 3 to properly diagnose the outliers. This threshold was used to remove outliers above or below the value about the standard deviation. Given that the dataset is large, it is expected that there will still be points on the plots that represent "outliers" outside of the whiskers. However, removing all of these points would significantly affect the entire distribution of the dataset. After z-score removals, the reduced dataset recorded just over 58,000 records, and the datasets were finalized before model preparation and training.

**Figure 4**

*Box-and-Whisker Plots of Numeric Variables*




## Pre-Processing and Feature Engineering

Several features were identified as needing additional processing during the EDA process before moving into the modeling stage. First, new features were created to replace any dates in the dataset. Due to many historical and economic events over the last 23 years of the data, it was important to strip dates to prevent the model from learning the historical events versus the indicators of a loan defaulting. All dates were transformed into ages or month counts like the age of the build or time between notes and acquisitions. The three particular program indicators, Affordable Housing Type, Social Bond Indicator, and Green Bond Indicator, were combined into a feature called ‘Special Program’. These programs commonly ebb and flow over the years as government regulations and priorities change.

Many features with missing data were handled in multiple ways. Categorical values were generally filled with the mode, sometimes by subgroup. Numerical values were imputed with a median due to the skewed nature of most of the features as analyzed in the exploratory data analysis. Some features were removed due to their lack of information, containing less than 25% of the recorded data. The Fannie Mae Multifamily Loan Performance Data Attribute Glossary and File Layout guide did provide additional context on why specific fields were blank (Fannie Mae, 2023). However, these designated fields were determined to not provide equal context for older loans, and thus, were removed.

## Data Splitting

In order to account for the imbalanced nature of the data, a stratified split was used. This method guarantees that the proportions of each class in both the training and test datasets closely resemble those in the original dataset (Géron, 2022). By employing a stratified split, the model is exposed to a representative sample of each class, which helps prevent biased performance metrics and assesses the model's effectiveness in real-world scenarios. A 20/80 split was applied to ensure we had enough of the target feature to get an accurate representation for testing purposes.

## Model Strategies

In this study, we developed a machine learning pipeline to predict loan defaults using Fannie Mae's multifamily housing data, addressing the complexities of the dataset, including numeric and categorical variables and the imbalance in the target variable, "Loan Ever 60+ Days Delinquent." The pipeline began with standardizing numeric features to ensure that each contributed equally to the model, which was crucial for distance-based models like K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) (Géron, 2022). To mitigate the severe imbalance in the target variable, with a minority of loans defaulting, the Synthetic Minority Over-sampling Technique (SMOTE) was employed, generating synthetic examples of the minority class to prevent model bias toward the majority class (Marini Systems, 2022).

Three models were chosen for training: Logistic Regression, KNN, and SVM. Logistic Regression serves as the baseline due to its simplicity and interpretability, enabling us to determine key predictors of loan defaults and providing a benchmark for more complex models. KNN was selected for its non-parametric nature, allowing it to effectively capture local patterns in the data by classifying points based on the majority class among their nearest neighbors. SVM was chosen for its ability to handle high-dimensional spaces and effectively separate classes by finding the optimal hyperplane, which is particularly advantageous in complex datasets where the number of features may exceed the number of samples. Each model provides unique strengths, enabling a comprehensive assessment of predictive power using different algorithmic approaches.

## Model Evaluation and Selection

Given that the target variable was a binary feature, common practice would be to analyze accuracy to determine the best-performing model. However, further evaluation metrics are necessary given the nature of the unbalanced features, both predictors and targets. Significant metrics for the baseline Logistic Regression, K-nearest neighbors, and Support Vector Machine include the precision, recall, and F1 scores. Table 3 presents the evaluation metrics for all three trained models. In every significant metric except recall, the K-nearest neighbors model performs best. While the margin may not be large, the precision score of the KNN model suggests that this model is the best choice for Fannie Mae going forward. This shows that despite the data imbalance challenges, the KNN model remains the most reliable in predicting loan defaults.

In the context of this unbalanced dataset, it is vital that the true positive class, or the prediction that a loan will default, is accurately predicted, as measured by precision. While the precision score of 0.5433 is not as strong as the team would like to see in the future, it does provide the highest rate of true positive predictions among the models trained. Despite having the lowest recall value, the KNN model does record the highest F1 score or the harmonic mean of precision and recall. This score suggests that the K-nearest neighbor model most commonly predicts accurate positive instances, a significant accomplishment in a dataset heavily imbalanced by the negative class.

**Table 3**

*Evaluation Metrics for Three Trained Models*

Model Comparison Metrics:

| | Model | Accuracy | Precision | Recall | F1 Score | CV Mean F1 Score |
|---|---|---|---|---|---|---|
| 0 | Logistic Regression | 0.7076725246463781 | 0.5293464739697475 | 0.7437144849904185 | 0.47399089678081396 | 0.40369748858948373 |
| 1 | KNN | 0.8825546506643807 | 0.5433477664724456 | 0.6718854766948119 | 0.5522494986532505 | 0.49179508249090775 |
| 2 | SVM | 0.8162023146163737 | 0.5402012149372046 | 0.7439836347753568 | 0.5286682513315014 | 0.483589250530267 |

## Limitations

The dataset presented an additional challenge for the project due to the imbalance in data distribution. Despite best efforts to mitigate this issue using the Synthetic Minority Over-sampling Technique (SMOTE), an increased number of positive default records would have substantially improved the model's predictive accuracy. We took great care to eliminate any features that might result in information leakage. We removed all date-related information from the dataset to prevent the model from being influenced by historical economic events; however, incorporating a feature that reflects current economic conditions at the time of acquisition would be advantageous, particularly given its significant impact on the housing industry.

## Future Research

This study examined three fundamental models: Logistic Regression, K-Nearest Neighbors, and Support Vector Machine. While these are valuable, it is important to consider that alternative models may yield more acceptable predictive performance, particularly for the precision metric. Adding ensemble methods or advanced techniques could enhance the accuracy and robustness of the model. In the future, researchers should consider exploring additional models like neural networks or gradient-boosting machines. One approach is to include economic data available during acquisition to analyze how specific indicators might influence the likelihood of default. This could involve integrating macroeconomic variables such as employment, interest, and GDP growth, providing deeper insights into the factors driving loan defaults. While this study focuses on multifamily loans, there is potential to broaden its scope to include single-family loan types and other types of real estate financing. Doing so would enhance the generalizability of the findings and provide a more comprehensive understanding of default risks across different housing market segments.

## Conclusion

Fannie Mae possesses a significant amount of financial data related to multifamily real estate, as well as private real estate. It is vital to the success of Fannie Mae that mortgage assistance, regardless of real estate type, is provided to clients who will properly pay back the assistance in a timely manner. As previously mentioned, the delinquency rate for multifamily loans peaked at 0.54%. In order to significantly lower the risk of financial distress for Fannie Mae, machine learning methods can be utilized to understand clients’ behavior better. Throughout the creation of the machine learning model, the team’s goal was to properly classify if loans were at risk of default, which would result in significant financial loss for Fannie Mae.

Introducing a K-nearest neighbors model created a positive initial idea of how Fannie Mae could use the immense amount of loan data. This model predicted the positive cases or cases where the loan would default at a higher rate than other models. As the model becomes more familiar with the mortgage financial assistance corporation, it can be optimized by reducing features and parameters to provide even more precise results. Ultimately, the KNN model will provide Fannie Mae with more insight into dispersed loans than if the loans are left to chance.

**References**

https://colab.research.google.com/drive/18KVt1EWGB2n507qnsQV1mYX5CNaNZR0E?usp=sharing

Fannie Mae. (2024). *Financial Supplement - Q4 and Full Year 2023.* Fannie Mae. Retrieved from https://www.fanniemae.com/media/50326/display

Fannie Mae. (2023). *Multifamily Loan Performance Data Attribute Glossary and File Layout.* Fannie Mae. Retrieved from https://capitalmarkets.fanniemae.com/media/5986/display

Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media, Inc.

Marini Systems. (2022, February 3). *Unbalanced classes (machine learning)*. Unbalanced Classes (Machine Learning). Retrieved from https://marini.systems/en/glossary/unbalanced-classes-machine-learning/

# Appendix

file:///C:/Users/lvand/Downloads/Group3_FinalProject.html

# Group3_FinalProject

August 12, 2024

## 1 Appendix

[38]:
```
import pandas as pd
import numpy as np
import json
import os
import matplotlib.pyplot as plt
import seaborn as sns
import json
import scipy.stats as stats

from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder,␣
 ↪StandardScaler, Normalizer
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪ConfusionMatrixDisplay, classification_report
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import Perceptron
from IPython.display import display
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from tabulate import tabulate
from scipy.stats import pointbiserialr
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

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

Fannie Mae Multifamily Housing Data from: https://datadynamics.fanniemae.com/data-dynamics/#/downloadLoanData/Multifamily Data Definitions: https://capitalmarkets.fanniemae.com/media/598 General Multifamily info: https://capitalmarkets.fanniemae.com/credit-risk-transfer/multifamily-credit-risk-transfer/multifamily-loan-performance-data Using: Loan Performance Data Main File

[3]:
```
df = pd.read_csv('/content/drive/MyDrive/ADS 504 Group 3 Final Project/
 ↪FNMA_MF_Loan_Performance_Data_202312.csv')
df.head()
```

```
<ipython-input-3-7917c9ac8d36>:1: DtypeWarning: Columns
(12,26,29,34,35,36,37,38,39,40,46,49,51,53,54,55,56,61) have mixed types.
Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('/content/drive/MyDrive/ADS 504 Group 3 Final
Project/FNMA_MF_Loan_Performance_Data_202312.csv')
```

[3]:
```
   Loan Number Acquisition Date   Note Date Maturity Date at Acquisition  \
0       140296       2000-10-31  1985-07-16                   2001-08-10
1       140296       2000-10-31  1985-07-16                   2001-08-10
2       140296       2000-10-31  1985-07-16                   2001-08-10
3       140296       2000-10-31  1985-07-16                   2001-08-10
4       140297       2000-10-31  1985-07-18                   2001-08-10

  Loan Acquisition UPB Amortization Type Interest Type Loan Product Type  \
0           $82,501.71               NaN           ARM               DUS
1           $82,501.71               NaN           ARM               DUS
2           $82,501.71               NaN           ARM               DUS
3           $82,501.71               NaN           ARM               DUS
4          $548,872.98               NaN           ARM               DUS

  Original UPB  Amortization Term  …  Prepayment Provision  \
0   $82,501.71                NaN  …                   NaN
1   $82,501.71                NaN  …                   NaN
2   $82,501.71                NaN  …                   NaN
3   $82,501.71                NaN  …                   NaN
4  $548,872.98                NaN  …                   NaN

  Prepayment Provision End Date Affordable Housing Type MCIRT Deal ID  \
0                           NaN                     NaN           NaN
1                           NaN                     NaN           NaN
2                           NaN                     NaN           NaN
3                           NaN                     NaN           NaN
4                           NaN                     NaN           NaN

   MCAS Deal ID  DUS Prepayment Outcomes DUS Prepayment Segments  Loan Age  \
0           NaN                      NaN                     NaN       NaN
1           NaN                      NaN                     NaN       NaN
2           NaN                      NaN                     NaN       NaN
```

```
3          NaN                       NaN                       NaN       NaN
4          NaN                       NaN                       NaN       NaN

   Green Bond Indicator Social Bond Indicator
0                   NaN                   NaN
1                   NaN                   NaN
2                   NaN                   NaN
3                   NaN                   NaN
4                   NaN                   NaN

[5 rows x 62 columns]
```

### 1.0.1 Initial Data Exploration

[4]:
```python
#(rows, columns)
print(df.shape)

#column names and data types
print(df.info())
```

```
(4628626, 62)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4628626 entries, 0 to 4628625
Data columns (total 62 columns):
 #   Column                                   Dtype
---  ------                                   -----
 0   Loan Number                              int64
 1   Acquisition Date                         object
 2   Note Date                                object
 3   Maturity Date at Acquisition             object
 4   Loan Acquisition UPB                     object
 5   Amortization Type                        object
 6   Interest Type                            object
 7   Loan Product Type                        object
 8   Original UPB                             object
 9   Amortization Term                        float64
 10  Original Interest Rate                   float64
 11  Lien Position                            object
 12  Transaction ID                           object
 13  Issue Date                               object
 14  Loan Acquisition LTV                     float64
 15  Underwritten DSCR                        float64
 16  Underwritten DSCR Type                   object
 17  Original Term                            int64
 18  Original I/O Term                        float64
 19  I/O End Date                             object
 20  Loan Ever 60+ Days Delinquent            object
```

```
 21  Loss Sharing Type                     object
 22  Modified Loss Sharing Percentage      float64
 23  Number of Properties at Acquisition   float64
 24  Property Acquisition Total Unit Count float64
 25  Specific Property Type                object
 26  Year Built                            object
 27  Property City                         object
 28  Property State                        object
 29  Property Zip Code                     object
 30  Metropolitan Statistical Area         object
 31  Physical Occupancy %                  float64
 32  Liquidation/Prepayment Code           object
 33  Liquidation/Prepayment Date           object
 34  Foreclosure Date                      object
 35  Credit Event Date                     object
 36  Foreclosure Value                     object
 37  Lifetime Net Credit Loss Amount       object
 38  Sale Price                            object
 39  Default Amount                        object
 40  Credit Event Type                     object
 41  Reporting Period Date                 object
 42  Loan Active Property Count            float64
 43  Note Rate                             float64
 44  Maturity Date - Current               object
 45  UPB - Current                         object
 46  Delinquency UPB                       object
 47  Loan Payment Status                   object
 48  SDQ Indicator                         object
 49  Most Recent Modification Date         object
 50  Modification Indicator                object
 51  Defeasance Date                       object
 52  Prepayment Provision                  object
 53  Prepayment Provision End Date         object
 54  Affordable Housing Type               object
 55  MCIRT Deal ID                         object
 56  MCAS Deal ID                          object
 57  DUS Prepayment Outcomes               object
 58  DUS Prepayment Segments               object
 59  Loan Age                              float64
 60  Green Bond Indicator                  object
 61  Social Bond Indicator                 object
dtypes: float64(12), int64(2), object(48)
memory usage: 2.1+ GB
None
```

[5]:
```
### Checking for Duplicates
unique_loans = df['Loan Number'].unique()
```

```python
print(len(unique_loans))
```

```
66487
```

There are 66,487 unique loans in the data showing duplication of the records.

[6]:
```python
print("Reporting Max:", df['Reporting Period Date'].max())
print("Reporting Min:", df['Reporting Period Date'].min())
print("Acquisition Date Max:", df['Acquisition Date'].max())
print("Acquisition Date Min:", df['Acquisition Date'].min())
```

```
Reporting Max: 2023-12-01
Reporting Min: 2000-01-01
Acquisition Date Max: 2023-12-29
Acquisition Date Min: 2000-01-01
```

The reporting range of the report is month-end reporting from Jan 2000 through Dec 2023. With Properties that Fannie Mae acquired from beginning of Jan 2000 through the end of December 2023.

[7]:
```python
df['Reporting Period Date'] = pd.to_datetime(df['Reporting Period Date'])
df_unique = df.loc[df.groupby('Loan Number')['Reporting Period Date'].idxmax()]
print(df_unique.shape)
```

```
(66487, 62)
```

[8]:
```python
df_unique.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 66487 entries, 3 to 4628613
Data columns (total 62 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   Loan Number                          66487 non-null  int64
 1   Acquisition Date                     66487 non-null  object
 2   Note Date                            66487 non-null  object
 3   Maturity Date at Acquisition         66487 non-null  object
 4   Loan Acquisition UPB                 66487 non-null  object
 5   Amortization Type                    66485 non-null  object
 6   Interest Type                        66487 non-null  object
 7   Loan Product Type                    66487 non-null  object
 8   Original UPB                         66487 non-null  object
 9   Amortization Term                    65447 non-null  float64
 10  Original Interest Rate               66487 non-null  float64
 11  Lien Position                        66486 non-null  object
 12  Transaction ID                       60698 non-null  object
 13  Issue Date                           60669 non-null  object
 14  Loan Acquisition LTV                 66484 non-null  float64
 15  Underwritten DSCR                    66478 non-null  float64
```

```
 16  Underwritten DSCR Type                  66478 non-null  object
 17  Original Term                           66487 non-null  int64
 18  Original I/O Term                       32174 non-null  float64
 19  I/O End Date                            32123 non-null  object
 20  Loan Ever 60+ Days Delinquent           66487 non-null  object
 21  Loss Sharing Type                       63923 non-null  object
 22  Modified Loss Sharing Percentage        2879 non-null   float64
 23  Number of Properties at Acquisition     65165 non-null  float64
 24  Property Acquisition Total Unit Count  64707 non-null  float64
 25  Specific Property Type                  66487 non-null  object
 26  Year Built                              66485 non-null  object
 27  Property City                           66487 non-null  object
 28  Property State                          66487 non-null  object
 29  Property Zip Code                       66487 non-null  object
 30  Metropolitan Statistical Area           66487 non-null  object
 31  Physical Occupancy %                    64707 non-null  float64
 32  Liquidation/Prepayment Code             39390 non-null  object
 33  Liquidation/Prepayment Date             39390 non-null  object
 34  Foreclosure Date                        656 non-null    object
 35  Credit Event Date                       707 non-null    object
 36  Foreclosure Value                       744 non-null    object
 37  Lifetime Net Credit Loss Amount         707 non-null    object
 38  Sale Price                              585 non-null    object
 39  Default Amount                          677 non-null    object
 40  Credit Event Type                       704 non-null    object
 41  Reporting Period Date                   66487 non-null  datetime64[ns]
 42  Loan Active Property Count              65752 non-null  float64
 43  Note Rate                               66485 non-null  float64
 44  Maturity Date - Current                 66487 non-null  object
 45  UPB - Current                           66487 non-null  object
 46  Delinquency UPB                         1237 non-null   object
 47  Loan Payment Status                     66487 non-null  object
 48  SDQ Indicator                           66487 non-null  object
 49  Most Recent Modification Date           529 non-null    object
 50  Modification Indicator                  66487 non-null  object
 51  Defeasance Date                         336 non-null    object
 52  Prepayment Provision                    66293 non-null  object
 53  Prepayment Provision End Date           54092 non-null  object
 54  Affordable Housing Type                 5743 non-null   object
 55  MCIRT Deal ID                           12510 non-null  object
 56  MCAS Deal ID                            990 non-null    object
 57  DUS Prepayment Outcomes                 56361 non-null  object
 58  DUS Prepayment Segments                 56362 non-null  object
 59  Loan Age                                32104 non-null  float64
 60  Green Bond Indicator                    60630 non-null  object
 61  Social Bond Indicator                   10581 non-null  object
dtypes: datetime64[ns](1), float64(12), int64(2), object(47)
memory usage: 32.0+ MB
```

### 1.0.2 Transforming Data

**Find Unique values and Handle Different Data Types**

[9]:
```
#Dropping columns that should not be included based on domain knowledge
#They would not be avalable at the time of acquisitions
df_unique.drop(['Transaction ID ', 'Issue Date', 'I/O End Date', 'Liquidation/
↪Prepayment Code', 'Liquidation/Prepayment Date',
        'Foreclosure Date', 'Credit Event Date', 'Foreclosure Value',␣
↪'Lifetime Net Credit Loss Amount', 'Default Amount',
        'Credit Event Type', 'Loan Active Property Count', 'UPB - Current',␣
↪'Delinquency UPB', 'Loan Payment Status',
        'SDQ Indicator', 'Most Recent Modification Date', 'Modification␣
↪Indicator', 'Defeasance Date',
        'Prepayment Provision End Date', 'DUS Prepayment Outcomes', 'DUS␣
↪Prepayment Segments', 'Loan Age'], axis=1, inplace=True)

df_unique.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 66487 entries, 3 to 4628613
Data columns (total 39 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Loan Number                       66487 non-null  int64
 1   Acquisition Date                  66487 non-null  object
 2   Note Date                         66487 non-null  object
 3   Maturity Date at Acquisition      66487 non-null  object
 4   Loan Acquisition UPB              66487 non-null  object
 5   Amortization Type                 66485 non-null  object
 6   Interest Type                     66487 non-null  object
 7   Loan Product Type                 66487 non-null  object
 8   Original UPB                      66487 non-null  object
 9   Amortization Term                 65447 non-null  float64
 10  Original Interest Rate            66487 non-null  float64
 11  Lien Position                     66486 non-null  object
 12  Loan Acquisition LTV              66484 non-null  float64
 13  Underwritten DSCR                 66478 non-null  float64
 14  Underwritten DSCR Type            66478 non-null  object
 15  Original Term                     66487 non-null  int64
 16  Original I/O Term                 32174 non-null  float64
 17  Loan Ever 60+ Days Delinquent     66487 non-null  object
 18  Loss Sharing Type                 63923 non-null  object
 19  Modified Loss Sharing Percentage  2879 non-null   float64
 20  Number of Properties at Acquisition  65165 non-null  float64
 21  Property Acquisition Total Unit Count 64707 non-null  float64
 22  Specific Property Type            66487 non-null  object
 23  Year Built                        66485 non-null  object
```

```
 24  Property City                          66487 non-null  object
 25  Property State                         66487 non-null  object
 26  Property Zip Code                      66487 non-null  object
 27  Metropolitan Statistical Area          66487 non-null  object
 28  Physical Occupancy %                   64707 non-null  float64
 29  Sale Price                             585 non-null    object
 30  Reporting Period Date                  66487 non-null  datetime64[ns]
 31  Note Rate                              66485 non-null  float64
 32  Maturity Date - Current                66487 non-null  object
 33  Prepayment Provision                   66293 non-null  object
 34  Affordable Housing Type                5743 non-null   object
 35  MCIRT Deal ID                          12510 non-null  object
 36  MCAS Deal ID                           990 non-null    object
 37  Green Bond Indicator                   60630 non-null  object
 38  Social Bond Indicator                  10581 non-null  object
dtypes: datetime64[ns](1), float64(10), int64(2), object(26)
memory usage: 20.3+ MB
```

[10]:

```python
# Creating New column to remove years/dates
# want to avoid predicting Economic events like 2008

# Convert 'Acquisition Date' and 'Note Date' to datetime objects
df_unique['Acquisition Date'] = pd.to_datetime(df_unique['Acquisition Date'])
df_unique['Note Date'] = pd.to_datetime(df_unique['Note Date'])

# Calculate the difference between the two dates
df_unique['Date Difference'] = (df_unique['Acquisition Date'] - df_unique['Note␣
↪Date']).dt.days

# Display the result
print(df_unique[['Acquisition Date', 'Note Date', 'Date Difference']])
```

```
        Acquisition Date  Note Date  Date Difference
3             2000-10-31 1985-07-16             5586
7             2000-10-31 1985-07-18             5584
67            2000-01-01 1999-11-01               61
141           2000-01-01 1999-11-01               61
264           2000-01-01 1999-12-01               31
…                    …          …                …
4628407       2018-01-25 2017-12-22               34
4628488       2018-09-28 2018-08-23               36
4628510       2020-01-30 2019-12-27               34
4628553       2020-01-30 2019-12-27               34
4628613       2021-02-25 2021-02-25                0

[66487 rows x 3 columns]
```

[11]:
```python
### Year had NAN that were filled with MODE
# Convert 'Acquisition Date' to datetime format
df_unique['Acquisition Date'] = pd.to_datetime(df_unique['Acquisition Date'])

# Extract the year from 'Acquisition Date'
df_unique['Acquisition Year'] = df_unique['Acquisition Date'].dt.year

# Ensure 'Acquisition Year' is integer
df_unique['Acquisition Year'] = df_unique['Acquisition Year'].astype(int)

# Convert 'Year Built' to numeric, handling non-numeric values
df_unique['Year Built'] = pd.to_numeric(df_unique['Year Built'],␣
↪errors='coerce')

# Fill missing values with the mode of 'Year Built'
year_built_mode = df_unique['Year Built'].mode()[0]
df_unique['Year Built'].fillna(year_built_mode, inplace=True)

# Calculate 'Age at Acquisition'
df_unique['Age at Acquisition'] = df_unique['Acquisition Year'] -␣
↪df_unique['Year Built']

# Print the relevant columns
print(df_unique[['Acquisition Date', 'Year Built', 'Age at Acquisition']])
```

```
        Acquisition Date  Year Built  Age at Acquisition
3             2000-10-31      1972.0                28.0
7             2000-10-31      1972.0                28.0
67            2000-01-01      1985.0                15.0
141           2000-01-01      1985.0                15.0
264           2000-01-01      1964.0                36.0
…                    …         …                 …
4628407       2018-01-25      1979.0                39.0
4628488       2018-09-28      1965.0                53.0
4628510       2020-01-30      1974.0                46.0
4628553       2020-01-30      1974.0                46.0
4628613       2021-02-25      1925.0                96.0

[66487 rows x 3 columns]
```

[12]:
```python
# changing the MCIRT and MCAS deals to t/f

# Convert empty strings and None to NaN
df_unique['MCIRT Deal ID'].replace('', None, inplace=True)
df_unique['MCAS Deal ID'].replace('', None, inplace=True)

# convert columns to string
```

```
df_unique['MCIRT Deal ID'] = df_unique['MCIRT Deal ID'].astype(str)
df_unique['MCAS Deal ID'] = df_unique['MCAS Deal ID'].astype(str)

# boolean columns indicating the presence of values
df_unique['MCIRT Deal ID Present'] = df_unique['MCIRT Deal ID'].apply(lambda x:␣
↪x not in [None, 'nan', ''])
df_unique['MCAS Deal ID Present'] = df_unique['MCAS Deal ID'].apply(lambda x: x␣
↪not in [None, 'nan', ''])
```

[13]:
```
# Merging the special programs into one category since most loans are labeled␣
↪as N or NAN.
# This is a newer data field, and programs come and go frequently
def is_special_program(row):
    # Combine Affordable Housing Type into a special program indicator
    affordable = row['Affordable Housing Type'] not in [None, '']
    # Combine Green Bond and Social Bond indicators
    green_bond = row['Green Bond Indicator'] == 'Y'
    social_bond = row['Social Bond Indicator'] == 'Y'

    return 'Y' if affordable or green_bond or social_bond else 'N'

# Apply the function to create a new column
df_unique['Special Program'] = df_unique.apply(is_special_program, axis=1)

#drop original columns in next step
```

[14]:
```
def categorize_prepayment_provision(value):
  if isinstance(value, str):
    if 'L' in value:
        return 'L'
    elif 'YM' in value:
        return 'YM'
    elif '%' in value:
        return 'Percent'
    elif 'DEF' in value:
        return 'DEF'
    elif 'O*' in value:
        return 'Open3'
    elif 'O' in value:
        return 'Open'
    else:
        return 'Other'  # Handling any values that do not fit into the defined␣
↪categories
  else:
      return 'Other'  # Handling non-string values

# Apply the mapping function to the DataFrame
```

```
df_unique['Prepayment Provision Category'] = df_unique['Prepayment Provision'].
 ↪apply(categorize_prepayment_provision)

# Optional: Drop the original 'Prepayment Provision' column if no longer needed
# df_unique = df_unique.drop(columns=['Prepayment Provision'])
```

**Drop Features based on Domain Knowledge**

[15]:
```
#Drop dates, Loan #, and extra location info
#drop 'Original I/O Term' because half the records were missing values
#drop 'Modified Loss Sharing Percentage' because most of the records are␣
 ↪missing data
#drop sales price because most data was missing and get the same info from Loan␣
 ↪Acquisition UPB & Underwritten DSCR
#drop Original UPB bc highly correlated to Loan Acquisition UPB
df_unique.drop(['Acquisition Date', 'Note Date', 'Year Built', 'Reporting␣
 ↪Period Date', 'Loan Number',
                'Maturity Date at Acquisition', 'Acquisition Year', 'Property␣
 ↪City', 'Property State',
                'Property Zip Code', 'Maturity Date - Current', 'Original I/O␣
 ↪Term',
                'Modified Loss Sharing Percentage', 'Sale Price', 'Original␣
 ↪UPB','Affordable Housing Type',
                'Green Bond Indicator', 'Social Bond Indicator', 'Prepayment␣
 ↪Provision', 'Metropolitan Statistical Area'], axis=1, inplace=True)

df_unique.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 66487 entries, 3 to 4628613
Data columns (total 26 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   Loan Acquisition UPB                 66487 non-null  object
 1   Amortization Type                    66485 non-null  object
 2   Interest Type                        66487 non-null  object
 3   Loan Product Type                    66487 non-null  object
 4   Amortization Term                    65447 non-null  float64
 5   Original Interest Rate               66487 non-null  float64
 6   Lien Position                        66486 non-null  object
 7   Loan Acquisition LTV                 66484 non-null  float64
 8   Underwritten DSCR                    66478 non-null  float64
 9   Underwritten DSCR Type               66478 non-null  object
 10  Original Term                        66487 non-null  int64
 11  Loan Ever 60+ Days Delinquent        66487 non-null  object
 12  Loss Sharing Type                    63923 non-null  object
 13  Number of Properties at Acquisition  65165 non-null  float64
```

```
 14  Property Acquisition Total Unit Count  64707 non-null  float64
 15  Specific Property Type                 66487 non-null  object
 16  Physical Occupancy %                   64707 non-null  float64
 17  Note Rate                              66485 non-null  float64
 18  MCIRT Deal ID                          66487 non-null  object
 19  MCAS Deal ID                           66487 non-null  object
 20  Date Difference                        66487 non-null  int64
 21  Age at Acquisition                     66487 non-null  float64
 22  MCIRT Deal ID Present                  66487 non-null  bool
 23  MCAS Deal ID Present                   66487 non-null  bool
 24  Special Program                        66487 non-null  object
 25  Prepayment Provision Category          66487 non-null  object
dtypes: bool(2), float64(9), int64(2), object(13)
memory usage: 12.8+ MB
```

[16]:

```
# Remove currency symbols and commas
df_unique['Loan Acquisition UPB'] = df_unique['Loan Acquisition UPB'].
 ↪replace({'\$': '', ',': ''}, regex=True)

# Convert to float
df_unique['Loan Acquisition UPB'] = df_unique['Loan Acquisition UPB'].
 ↪astype(float)
```

[17]:

```
df_unique.head()
df_unique.isna().sum()
```

[17]:

```
Loan Acquisition UPB                      0
Amortization Type                         2
Interest Type                             0
Loan Product Type                         0
Amortization Term                      1040
Original Interest Rate                    0
Lien Position                             1
Loan Acquisition LTV                      3
Underwritten DSCR                         9
Underwritten DSCR Type                    9
Original Term                             0
Loan Ever 60+ Days Delinquent             0
Loss Sharing Type                      2564
Number of Properties at Acquisition    1322
Property Acquisition Total Unit Count  1780
Specific Property Type                    0
Physical Occupancy %                   1780
Note Rate                                 2
MCIRT Deal ID                             0
MCAS Deal ID                              0
Date Difference                           0
```

```
Age at Acquisition                          0
MCIRT Deal ID Present                       0
MCAS Deal ID Present                        0
Special Program                             0
Prepayment Provision Category               0
dtype: int64
```

**Handling Missing NAN Values**

[18]:
```
# Cleanup NAN Values

#Amortization Type has 2 NAN
def fill_mode(series):
    mode = series.mode()
    if not mode.empty:
        return series.fillna(mode[0])
    else:
        return series

def fill_median(series, global_median):
    # Calculate the median for the series
    median = series.median()
    # If median is valid, fill missing values with it
    if pd.notna(median):
        return series.fillna(median)
    else:
        # If median is not valid, fill with the global median
        return series.fillna(global_median)

# Apply the function to fill missing values in 'Amortization Type' grouped by␣
 ↪'Interest Type'
df_unique['Amortization Type'] = df_unique.groupby('Interest␣
 ↪Type')['Amortization Type'].transform(fill_mode)

#Amortization Term has 1040 NANs
# Apply the function to fill missing values in 'Amortization Term' grouped by␣
 ↪'Loan Product Type'
df_unique['Amortization Term'] = df_unique.groupby('Loan Product␣
 ↪Type')['Amortization Term'].transform(fill_mode)

#lien position has 1 NAN filling with Mode
df_unique['Lien Position'].fillna(df_unique['Lien Position'].mode()[0],␣
 ↪inplace=True)

#Loan Acquisition LTV has 3 NAN filling with Median due to heavy right skew
df_unique['Loan Acquisition LTV'].fillna(df_unique['Loan Acquisition LTV'].
 ↪median(), inplace=True)
```

```
# Underwritten DSCR & Underwritten DSCR Type are heavily skewed and returning␣
↪Medium and Mode
df_unique['Underwritten DSCR'].fillna(df_unique['Underwritten DSCR'].median(),␣
↪inplace=True)
df_unique['Underwritten DSCR Type'].fillna(df_unique['Underwritten DSCR Type'].
↪mode()[0], inplace=True)

# Loss Sharing Type has 2564 NAN filling with mode grouped by loan product type
def impute_mode(group):
    mode_value = group.mode()[0]  # Get the mode of the group
    return group.fillna(mode_value)

df_unique['Loss Sharing Type'] = df_unique.groupby('Loan Product Type')['Loss␣
↪Sharing Type'].transform(impute_mode)

# Number of Properties at Acquisition & Property Acquisition Total Unit Count &␣
↪Physical Occupancy %
# filling with median group by loan type
global_median = df_unique['Number of Properties at Acquisition'].median()
df_unique['Number of Properties at Acquisition'] = df_unique.groupby('Loan␣
↪Product Type')['Number of Properties at Acquisition'].transform(
    lambda x: fill_median(x, global_median))
df_unique['Property Acquisition Total Unit Count'] = df_unique.groupby('Loan␣
↪Product Type')['Property Acquisition Total Unit Count'].transform(
    lambda x: fill_median(x, global_median))
df_unique['Physical Occupancy %'] = df_unique.groupby('Loan Product␣
↪Type')['Physical Occupancy %'].transform(
    lambda x: fill_median(x, global_median))

# Note Rate has 2 NAN filling with mean grouped by Interest type
df_unique['Note Rate'] = df_unique.groupby('Interest Type')['Note Rate'].
↪transform(lambda x: x.fillna(x.mean()))
```

```
/usr/local/lib/python3.10/dist-packages/numpy/lib/nanfunctions.py:1215:
RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis, out=out, keepdims=keepdims)
/usr/local/lib/python3.10/dist-packages/numpy/lib/nanfunctions.py:1215:
RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis, out=out, keepdims=keepdims)
/usr/local/lib/python3.10/dist-packages/numpy/lib/nanfunctions.py:1215:
RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis, out=out, keepdims=keepdims)
```

[19]:
```
# Verify above logic
df_unique.isna().sum()
```

```
[19]: Loan Acquisition UPB                        0
      Amortization Type                           0
      Interest Type                               0
      Loan Product Type                           0
      Amortization Term                           0
      Original Interest Rate                      0
      Lien Position                               0
      Loan Acquisition LTV                        0
      Underwritten DSCR                           0
      Underwritten DSCR Type                      0
      Original Term                               0
      Loan Ever 60+ Days Delinquent               0
      Loss Sharing Type                           0
      Number of Properties at Acquisition         0
      Property Acquisition Total Unit Count       0
      Specific Property Type                      0
      Physical Occupancy %                        0
      Note Rate                                   0
      MCIRT Deal ID                               0
      MCAS Deal ID                                0
      Date Difference                             0
      Age at Acquisition                          0
      MCIRT Deal ID Present                       0
      MCAS Deal ID Present                        0
      Special Program                             0
      Prepayment Provision Category               0
      dtype: int64
```

### 1.0.3 Exploratory Data Analysis

```
[20]: df_unique.shape
```

```
[20]: (66487, 26)
```

```
[21]: # Summary Statistics of entire unique df
print("\nSummary statistics:")
summary_stats = df_unique.describe()
display(summary_stats)
```

```
Summary statistics:

       Loan Acquisition UPB  Amortization Term  Original Interest Rate  \
count          6.648700e+04       66487.000000            66487.000000
mean           1.253142e+07         304.238167                4.823674
std            2.377555e+07         128.689564                1.260500
min            1.144005e+04           0.000000                1.222000
25%            2.580000e+06         360.000000                3.920000
```

```
50%             6.071000e+06          360.000000                4.750000
75%             1.420000e+07          360.000000                5.710000
max             9.514950e+08          540.000000                9.800000

       Loan Acquisition LTV  Underwritten DSCR  Original Term  \
count          66487.000000       66487.000000   66487.000000
mean              64.673790           1.757384     125.544257
std               14.207045           1.522877      48.015747
min                0.000000           0.390000      11.000000
25%               58.100000           1.280000     120.000000
50%               66.900000           1.420000     120.000000
75%               75.000000           1.790000     120.000000
max              276.200000          85.720000     480.000000

       Number of Properties at Acquisition  \
count                         66487.000000
mean                              1.031344
std                               0.935104
min                               1.000000
25%                               1.000000
50%                               1.000000
75%                               1.000000
max                             105.000000

       Property Acquisition Total Unit Count  Physical Occupancy %  \
count                           66487.000000          66487.000000
mean                              160.271421             89.190800
std                               179.891703             20.431779
min                                 1.000000              0.000000
25%                                56.000000             92.000000
50%                               122.000000             95.000000
75%                               224.000000             95.000000
max                             11246.000000            100.000000

          Note Rate  Date Difference  Age at Acquisition
count  66487.000000     66487.000000        66487.000000
mean       4.867519        31.667213           35.137952
std        1.287385        47.432683           24.845559
min        0.552000       -30.000000           -1.000000
25%        3.950000        24.000000           18.000000
50%        4.765000        30.000000           33.000000
75%        5.740000        37.000000           47.000000
max       11.570000      5586.000000          219.000000
```

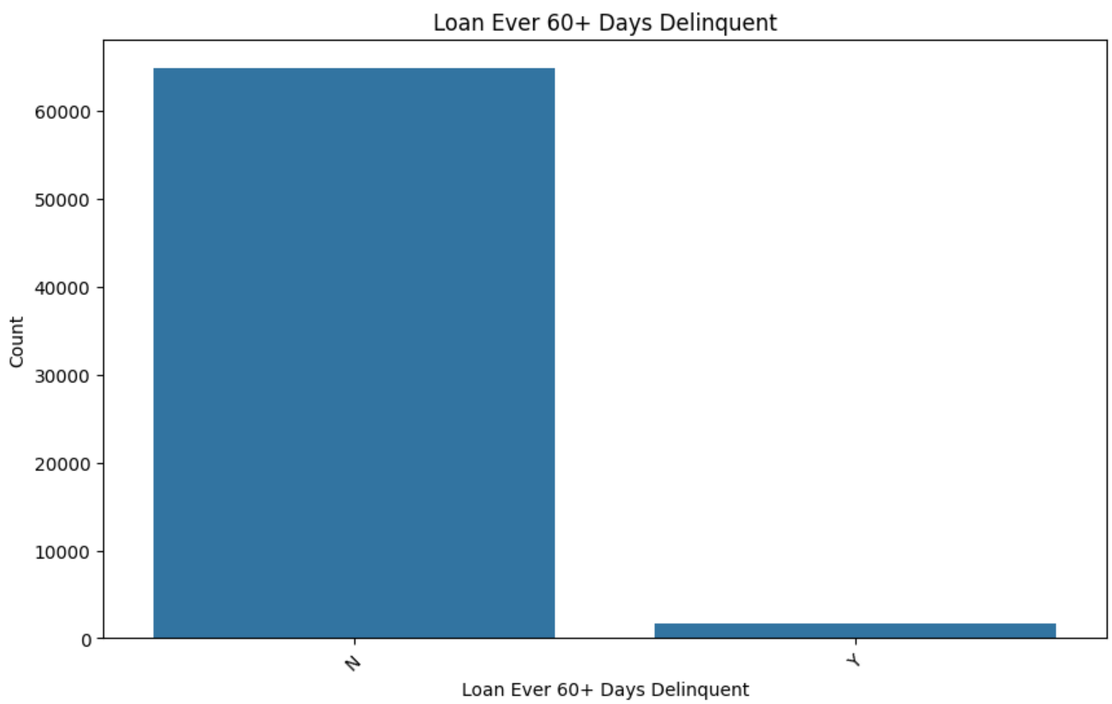
[22]:

```
# Separate categorical and numeric data for EDA
numeric_df = df_unique.select_dtypes(include=['float64', 'int64', 'bool'])
categorical_df = df_unique.select_dtypes(include=['object'])
```

[23]:
```python
# Check the distribution of the target variable
if 'Loan Ever 60+ Days Delinquent' in df.columns:
    print("\nDistribution of the target variable:")
    value_counts = df_unique['Loan Ever 60+ Days Delinquent'].value_counts()
    print(value_counts)
```

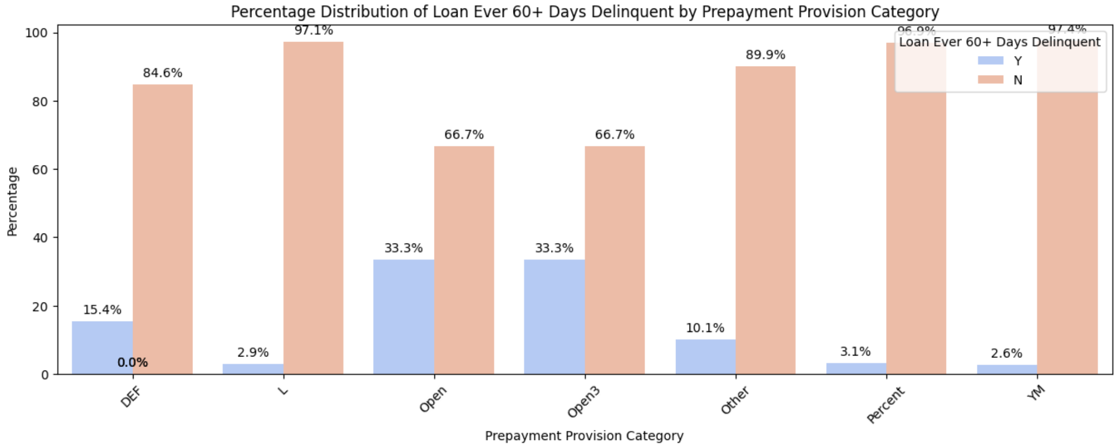
```
Distribution of the target variable:
Loan Ever 60+ Days Delinquent
N    64727
Y     1760
Name: count, dtype: int64
```

[24]:
```python
# Convert value_counts to DataFrame for better handling with Seaborn
target_counts_df = value_counts.reset_index()
target_counts_df.columns = ['Loan Ever 60+ Days Delinquent', 'count']

# Plotting the distribution
plt.figure(figsize=(10, 6))
sns.barplot(data=target_counts_df, x='Loan Ever 60+ Days Delinquent', y='count')
plt.title('Loan Ever 60+ Days Delinquent')
plt.xlabel('Loan Ever 60+ Days Delinquent')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



There is a significant inbalance in the target variable that will have to be balanced using SMOTE in the preprocessing stages of the pipeline further on in the analysis.

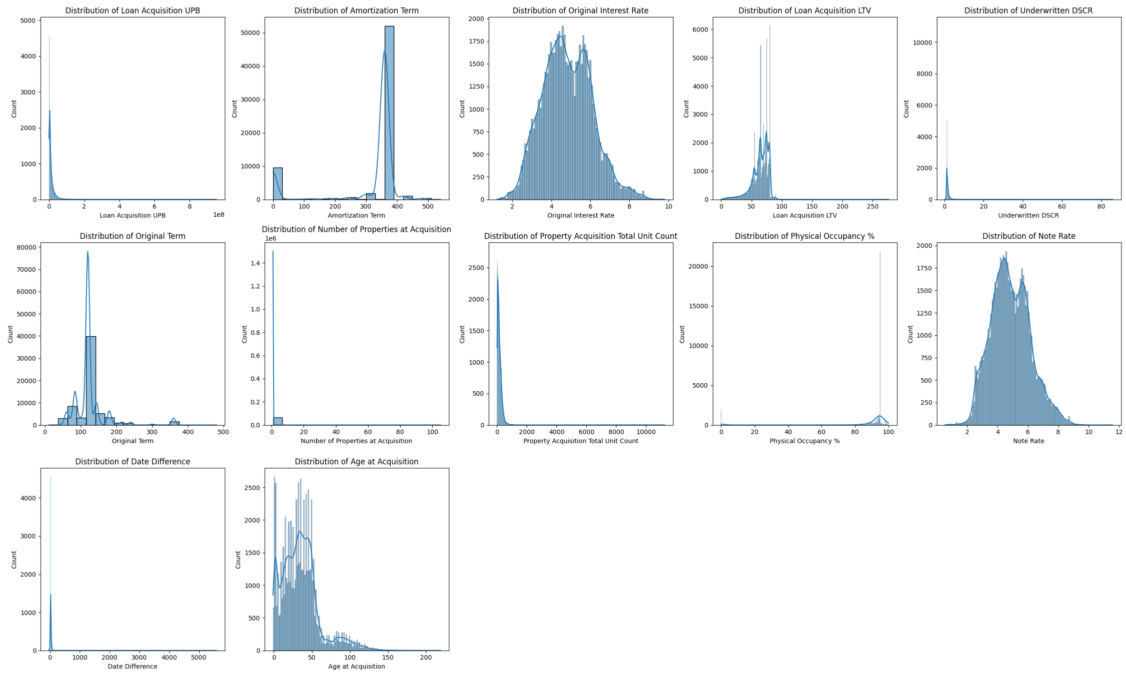
[25]:
```
# Distribution of target variable by prepayment provision category feature
plt.figure(figsize=(15, 5))
# Calculate counts
count_data = df_unique.groupby(['Prepayment Provision Category', 'Loan Ever 60+␣
↪Days Delinquent']).size().unstack().fillna(0)

# Calculate percentages
percentage_data = count_data.div(count_data.sum(axis=1), axis=0) * 100

# Reset index for plotting
percentage_data = percentage_data.reset_index()
# Melt the DataFrame for seaborn compatibility
melted_percentage_data = percentage_data.melt(id_vars='Prepayment Provision␣
↪Category',
                                              value_vars=['Y', 'N'],
                                              var_name='Loan Ever 60+ Days␣
↪Delinquent',
                                              value_name='Percentage')

# Create a bar plot
sns.barplot(data=melted_percentage_data, x='Prepayment Provision Category',␣
↪y='Percentage', hue='Loan Ever 60+ Days Delinquent', palette='coolwarm')

# Add labels and title
plt.title('Percentage Distribution of Loan Ever 60+ Days Delinquent by␣
↪Prepayment Provision Category')
plt.xlabel('Prepayment Provision Category')
plt.ylabel('Percentage')
plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.legend(title='Loan Ever 60+ Days Delinquent', loc='upper right')

# Add percentage labels on bars
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_height():.1f}%',
                       (p.get_x() + p.get_width() / 2., p.get_height()),
                       ha = 'center',
                       va = 'center',
                       xytext = (0, 9),
                       textcoords = 'offset points')

plt.show()
```




### Distribution of Features

[26]:
```
# Distribution (Historgram) of Numeric Features
print("\nHistograms for numerical features:")
numerical_features = df_unique.select_dtypes(include=[np.number]).columns.
 ↪tolist()

# Define the number of plots per row
plots_per_row = 5
num_features = len(numerical_features)
num_rows = (num_features // plots_per_row) + int(num_features % plots_per_row !
 ↪= 0)

fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(plots_per_row * 5,␣
 ↪num_rows * 5))

# Flatten axes array for easy iteration
axes = axes.flatten()

for i, feature in enumerate(numerical_features):
    sns.histplot(df_unique[feature], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {feature}')

# Remove any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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

```
Histograms for numerical features:
```




There are numeric variables that will need to be standardized in the preproccesing pipeline of the models.

[27]:
```python
# Get frequency counts for each categorical column
frequency_counts = categorical_df.apply(lambda x: x.value_counts()).stack().
 ↪reset_index()
frequency_counts.columns = ['variable', 'value', 'count']

# Convert 'count' to numeric if it's not already
frequency_counts['count'] = pd.to_numeric(frequency_counts['count'],␣
 ↪errors='coerce')

# Sort the frequency counts table
frequency_counts = frequency_counts.sort_values(by=['variable', 'count'],␣
 ↪ascending=[True, False])

# Print the frequency counts table using tabulate
print(tabulate(frequency_counts, headers='keys', tablefmt='psql'))
```

```
+----+-----------------------------------+-------------------------------+-------
--+
|    | variable                          | value                         |
count |
|----+-----------------------------------+-------------------------------+-------
--|
|  0 | ARM                               | Interest Type                 |
```

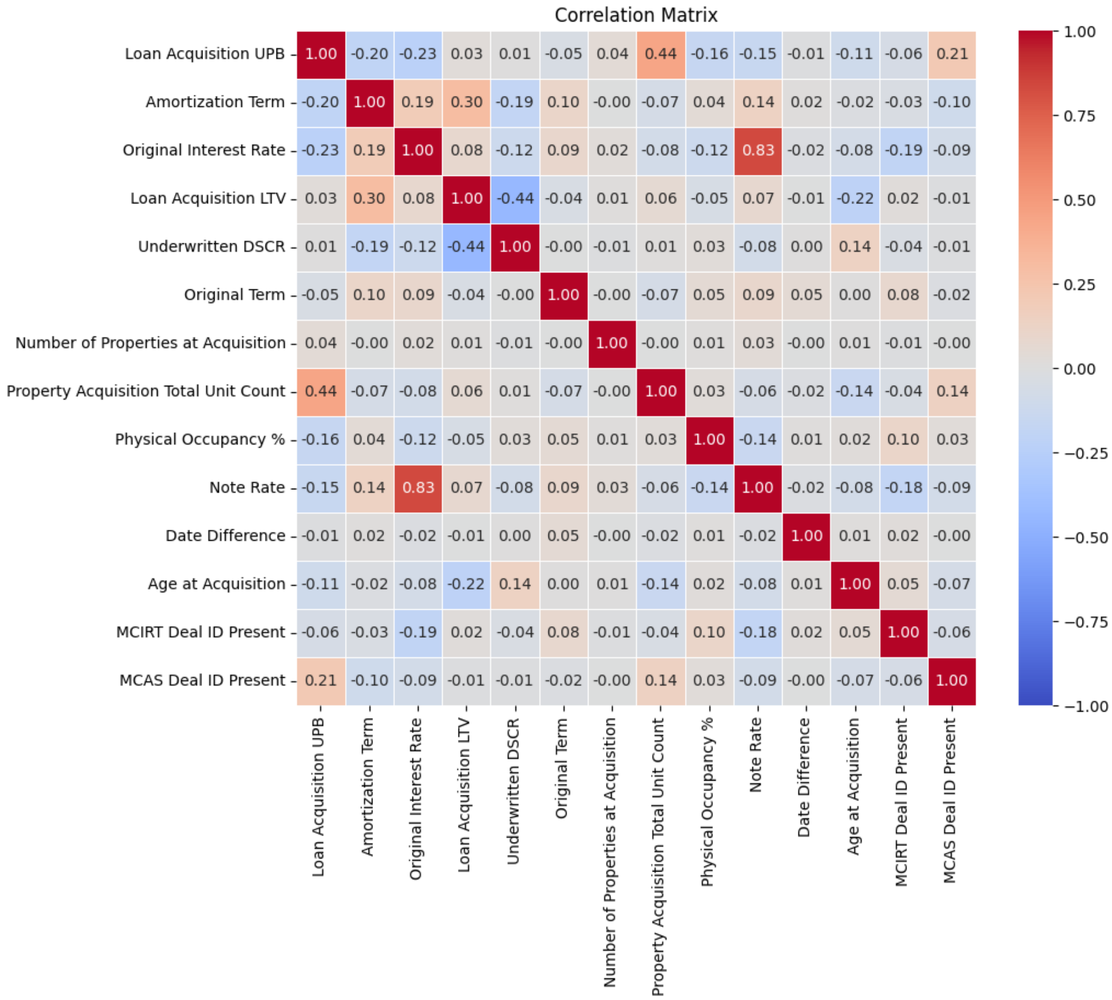
```
5862 |
|  1 | Amortizing Balloon                 | Amortization Type           |
31949 |
|  2 | Bulk Delivery                      | Loan Product Type           |
374 |
|  3 | Cooperative                        | Specific Property Type      |
1881 |
|  4 | Credit Facility                    | Loan Product Type           |
1322 |
|  5 | DEF                                | Prepayment Provision Category |
13 |
|  6 | DUS                                | Loan Product Type           |
62826 |
|  7 | Deal UW DSCR NCF                   | Underwritten DSCR Type      |
740 |
|  8 | Dedicated Student                  | Specific Property Type      |
1064 |
|  9 | First                              | Lien Position               |
60731 |
| 10 | Fixed                              | Interest Type               |
60625 |
| 11 | Fourth or More Subordinate         | Lien Position               |
46 |
| 12 | Fully Amortizing                   | Amortization Type           |
2429 |
| 13 | Interest Only/Amortizing/Balloon | Amortization Type           |
21626 |
| 14 | Interest Only/Balloon              | Amortization Type           |
10465 |
| 15 | Interest Only/Fully Amortizing   | Amortization Type           |
18 |
| 16 | L                                  | Prepayment Provision Category |
5122 |
| 17 | Lender UW DSCR                     | Underwritten DSCR Type      |
25209 |
| 18 | MCAS 2019-01                       | MCAS Deal ID                |
340 |
| 19 | MCAS 2020-01                       | MCAS Deal ID                |
218 |
| 20 | MCAS 2023-01                       | MCAS Deal ID                |
432 |
| 21 | MCIRT 2016-01                      | MCIRT Deal ID               |
782 |
| 22 | MCIRT 2017-01                      | MCIRT Deal ID               |
1273 |
| 23 | MCIRT 2018-01                      | MCIRT Deal ID               |
1103 |
| 24 | MCIRT 2018-02                      | MCIRT Deal ID               |
```

```
1085 |
| 25 | MCIRT 2019-01                    | MCIRT Deal ID                 |
1154 |
| 26 | MCIRT 2019-02                    | MCIRT Deal ID                 |
1031 |
| 27 | MCIRT 2019-03                    | MCIRT Deal ID                 |
1042 |
| 28 | MCIRT 2020-01                    | MCIRT Deal ID                 |
1017 |
| 29 | MCIRT 2021-01                    | MCIRT Deal ID                 |
1323 |
| 30 | MCIRT 2021-02                    | MCIRT Deal ID                 |
883 |
| 31 | MCIRT 2022-01                    | MCIRT Deal ID                 |
1262 |
| 32 | MCIRT 2023-01                    | MCIRT Deal ID                 |
555 |
| 33 | Manufactured Housing Community   | Specific Property Type        |
3062 |
| 34 | Military                         | Specific Property Type        |
296 |
| 35 | Multifamily                      | Specific Property Type        |
56601 |
| 36 | Multiple Properties              | Specific Property Type        |
1780 |
| 37 | N                                | Loan Ever 60+ Days Delinquent |
64727 |
| 38 | No Lender Loss Sharing           | Loss Sharing Type             |
1685 |
| 39 | Non-DUS                          | Loan Product Type             |
1965 |
| 40 | Open                             | Prepayment Provision Category |
3 |
| 41 | Open3                            | Prepayment Provision Category |
6 |
| 43 | Other                            | Prepayment Provision Category |
268 |
| 42 | Other                            | Specific Property Type        |
8 |
| 44 | Pari Passu                       | Loss Sharing Type             |
34283 |
| 45 | Percent                          | Prepayment Provision Category |
1147 |
| 46 | Second                           | Lien Position                 |
5358 |
| 47 | Seniors                          | Specific Property Type        |
1795 |
| 48 | Standard DUS                     | Loss Sharing Type             |
```

```
30519 |
| 49 | Third                                | Lien Position                  |
352 |
| 50 | UW Actual DSCR                       | Underwritten DSCR Type         |
17441 |
| 51 | UW DSCR NCF                          | Underwritten DSCR Type         |
23097 |
| 53 | Y                                    | Special Program                |
66487 |
| 52 | Y                                    | Loan Ever 60+ Days Delinquent |
1760 |
| 54 | YM                                   | Prepayment Provision Category |
59928 |
| 56 | nan                                  | MCAS Deal ID                   |
65497 |
| 55 | nan                                  | MCIRT Deal ID                  |
53977 |
+----+----------------------------------+--------------------------------+-------
--+
```

**Correlations**

[28]:
```
# Calculate the correlation matrix of numeric variables
corr_matrix = numeric_df.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.
 ↪5, vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```




[29]:
```python
# Correlation of numeric features to target

# Convert 'Y'/'N' in the target variable to 1/0
df_unique['Loan Ever 60+ Days Delinquent'] = df_unique['Loan Ever 60+ Days␣
↪Delinquent'].map({'Y': 1, 'N': 0})

# Drop rows where the target variable is NaN
df_unique = df_unique.dropna(subset=['Loan Ever 60+ Days Delinquent'])

# Ensure all numerical features are actually numeric
numerical_features = df_unique.select_dtypes(include=[np.number]).columns.
↪tolist()

# Additional check to ensure there are no non-numeric entries in numerical␣
↪features
for feature in numerical_features:
```

```
    if feature != 'Loan Ever 60+ Days Delinquent':
        df_unique[feature] = pd.to_numeric(df_unique[feature], errors='coerce')

# Drop rows where any numerical feature is NaN after conversion
df_unique = df_unique.dropna(subset=numerical_features)

# Calculate point biserial correlation for numerical features
correlations = {}
for feature in numerical_features:
    if feature != 'Loan Ever 60+ Days Delinquent':
        corr,_ = pointbiserialr(df_unique[feature], df_unique['Loan Ever 60+␣
↪Days Delinquent'])
        correlations[feature] = corr

# Convert the correlations to a DataFrame for better visualization
correlations_df = pd.DataFrame(correlations.items(), columns=['Feature',␣
↪'Correlation'])
print(correlations_df)
```

```
                              Feature  Correlation
0                 Loan Acquisition UPB    -0.028789
1                     Amortization Term     0.059474
2               Original Interest Rate     0.101287
3                  Loan Acquisition LTV     0.089547
4                    Underwritten DSCR    -0.033349
5                         Original Term     0.007612
6     Number of Properties at Acquisition     0.001286
7   Property Acquisition Total Unit Count    -0.020977
8                  Physical Occupancy %    -0.010266
9                             Note Rate     0.092650
10                      Date Difference     0.028838
11                   Age at Acquisition     0.007437
```

Highly correlated variables note rate and original interest rate have similar correlations to the target variable. Knowing this, we will remove the note rate feature in order to avoid multicollinearity and still represent the feature properly with original interest rate.

[30]:
```
# Remove Note Rate
df_unique = df_unique.drop(columns=['Note Rate'])
```

[31]:
```
# Correlation of categorical features to target

def cramers_v(confusion_matrix):
    chi2 = stats.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum()
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
```

```
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

categorical_features = df_unique.select_dtypes(include=['category', 'object',␣
 ↪'bool']).columns.tolist()

# Calculate Cramer's V (chi-square) for categorical features
correlations = {}
for feature in categorical_features:
    if feature != 'Loan Ever 60+ Days Delinquent':
        # Create a contingency table
        confusion_matrix = pd.crosstab(df_unique[feature], df['Loan Ever 60+␣
 ↪Days Delinquent'])
        # Calculate Cramer's V
        correlations[feature] = cramers_v(confusion_matrix.to_numpy())

# Convert the correlations to a DataFrame for better visualization
correlations_df = pd.DataFrame(correlations.items(), columns=['Feature',␣
 ↪'Cramér\'s V'])
print(correlations_df)
```

```
                          Feature  Cramér's V
0               Amortization Type    0.068875
1                   Interest Type    0.006411
2               Loan Product Type    0.018170
3                   Lien Position    0.014070
4          Underwritten DSCR Type    0.093507
5               Loss Sharing Type    0.033063
6          Specific Property Type    0.070986
7                   MCIRT Deal ID    0.038203
8                    MCAS Deal ID    0.014750
9           MCIRT Deal ID Present    0.034219
10           MCAS Deal ID Present    0.014743
11                Special Program         NaN
12  Prepayment Provision Category    0.037879
```

```
<ipython-input-31-cc25151ac21d>:11: RuntimeWarning: invalid value encountered in
scalar divide
  return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))
```

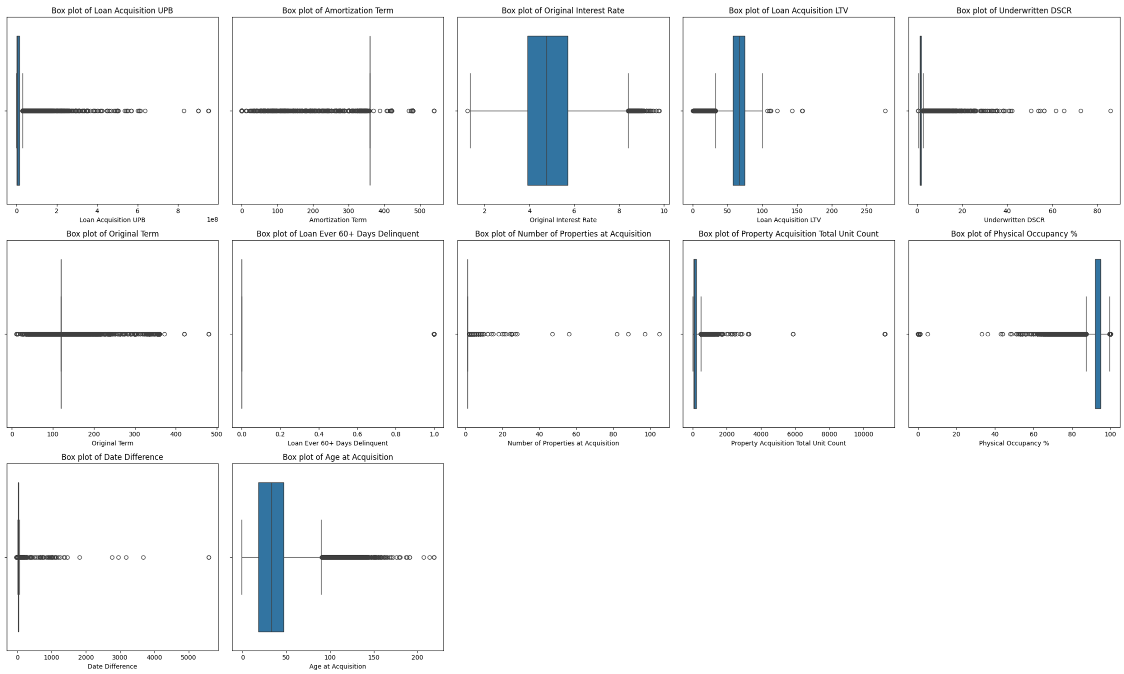
There are not strong enough correlations in either the categorical or numeric data to determine which features would be best to include in models. As the team has already removed unnecessary features based on domain knowledge and timelines of data, it is within the scope to include all remaining features in model training. Ultimately, the team will determine the top performing features upon model completion.

### Outliers

[32]:

```python
# Check for outliers in numeric data
print("\nBox plots for numeric features:")
numerical_features = df_unique.select_dtypes(include=[np.number]).columns.
 ↪tolist()

# Define the number of plots per row
plots_per_row = 5
num_features = len(numerical_features)
num_rows = (num_features // plots_per_row) + int(num_features % plots_per_row !
 ↪= 0)

fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(plots_per_row * 5,␣
 ↪num_rows * 5))

# Flatten axes array for easy iteration
axes = axes.flatten()

for i, feature in enumerate(numerical_features):
    sns.boxplot(x=df_unique[feature], ax=axes[i])
    axes[i].set_title(f'Box plot of {feature}')

# Remove any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()

# Save the figure as a .jpeg file
plt.savefig('/content/drive/MyDrive/outliers.jpeg', format='jpeg', dpi=300)
plt.show()
```

```
Box plots for numeric features:
```




[33]:

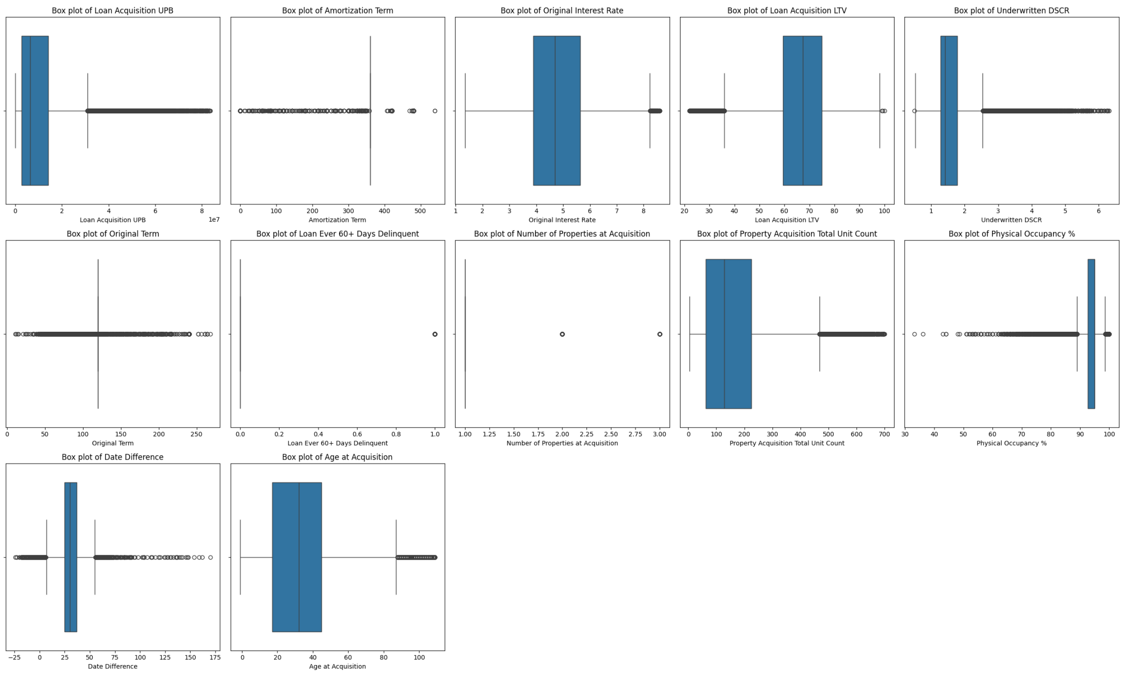
```
# Handle outliers in numeric features with Z score

# Define the target feature in order to not change it with z-score calculation
target_feature = 'Loan Ever 60+ Days Delinquent'

# List of numerical features excluding the target feature
numerical_features = df_unique.select_dtypes(include=[np.number]).columns.
 ↪tolist()
numerical_features.remove(target_feature)  # Remove the target feature

# Calculate Z-scores for each numeric feature
z_scores = pd.DataFrame()
for feature in numerical_features:
    z_scores[feature] = stats.zscore(df_unique[feature].dropna())

# Define a threshold for Z-scores
threshold = 3

# Identify outliers based on Z-scores
outliers = (z_scores.abs() > threshold).any(axis=1)

# Filter out rows with outliers
df_final = df_unique[~outliers]

print(f"Original dataset shape: {df_unique.shape}")
```

```
print(f"Dataset shape after removing outliers: {df_final.shape}")
```

```
Original dataset shape: (66487, 25)
Dataset shape after removing outliers: (58325, 25)
```

[34]:
```
# Validate outlier removal AFTER z-scores
# Check for outliers in numeric data
print("\nBox plots for numeric features after outlier removal:")
numerical_features = df_final.select_dtypes(include=[np.number]).columns.
 ↪tolist()

# Define the number of plots per row
plots_per_row = 5
num_features = len(numerical_features)
num_rows = (num_features // plots_per_row) + int(num_features % plots_per_row !
 ↪= 0)

fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(plots_per_row * 5,␣
 ↪num_rows * 5))

# Flatten axes array for easy iteration
axes = axes.flatten()

for i, feature in enumerate(numerical_features):
    sns.boxplot(x=df_final[feature], ax=axes[i])
    axes[i].set_title(f'Box plot of {feature}')

# Remove any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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

```
Box plots for numeric features after outlier removal:
```




### 1.0.4 Model Preparation

Train/Test Split

[35]:
```
# Define the target variable column name
target_column = 'Loan Ever 60+ Days Delinquent'

# Separate features and target variable
X = df_final.drop(target_column,  axis=1)
y = df_final[target_column]

# Identify numeric and categorical columns
numeric_features = X.select_dtypes(include=['int64', 'float64', 'bool']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Create preprocessing pipelines for both numeric and categorical data
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing pipelines
preprocessor = ColumnTransformer(
```

```
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create the final pipeline with preprocessor, SMOTE, and classifier
pipeline = ImbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=42)), # Balance Target Feature
    ('classifier', LogisticRegression(max_iter=1000))
])
```

[36]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y)
```

[39]:
```
# Train the model
pipeline.fit(X_train, y_train)

# Make predictions on the training set
y_train_pred = pipeline.predict(X_train)

# Model Evaluation
y_pred = pipeline.predict(X_test)

# Print training set
print("Training Set Performance:")
print("Confusion Matrix:")
print(confusion_matrix(y_train, y_train_pred))
print("Classification Report:")
print(classification_report(y_train, y_train_pred))

# Print test set
print("\nTest Set Performance:")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Get feature names AFTER preprocessing
# Fit the preprocessor on the training data
preprocessor.fit(X_train)

# Transform the training data to get the feature names after preprocessing
```

```python
feature_names_transformed = preprocessor.get_feature_names_out()

# Access the LogisticRegression model within the pipeline
logistic_model = pipeline.named_steps['classifier']

# Create a DataFrame to view feature importance
coefficients = logistic_model.coef_[0] #Extract coefficients from the logistic␣
↪regression model
feature_importance = pd.DataFrame({
    'Feature': feature_names_transformed,
    'Coefficient': coefficients
})

# Sort features by the coefficients
feature_importance = feature_importance.sort_values(by='Coefficient',␣
↪ascending=False)

# Print top 10 features
print("\nTop 10 Features:")
print(feature_importance.head(10))
```

```
Training Set Performance:
Confusion Matrix:
[[32363 13068]
 [  300   929]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.71      0.83     45431
           1       0.07      0.76      0.12      1229

    accuracy                           0.71     46660
   macro avg       0.53      0.73      0.48     46660
weighted avg       0.97      0.71      0.81     46660


Test Set Performance:
Confusion Matrix:
[[8015 3343]
 [  67  240]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.71      0.82     11358
           1       0.07      0.78      0.12       307

    accuracy                           0.71     11665
```

```
   macro avg       0.53      0.74      0.47     11665
weighted avg       0.97      0.71      0.81     11665


Top 10 Features:
                                          Feature  Coefficient
40            cat__Specific Property Type_Seniors     2.356700
34  cat__Specific Property Type_Dedicated Student     1.997893
62       cat__Prepayment Provision Category_Open3     1.594319
23                       cat__Lien Position_First     1.532592
55                 cat__MCAS Deal ID_MCAS 2020-01     1.204078
54                 cat__MCAS Deal ID_MCAS 2019-01     1.160315
51               cat__MCIRT Deal ID_MCIRT 2022-01     0.936843
44               cat__MCIRT Deal ID_MCIRT 2018-02     0.787116
45               cat__MCIRT Deal ID_MCIRT 2019-01     0.776461
3                      num__Loan Acquisition LTV     0.772756
```

[40]:

```python
# Define the parameter grid for logistic regression
param_grid = {
    'classifier__C': [0.01, 0.1, 1, 10, 100],
    'classifier__solver': ['lbfgs', 'liblinear']
}

# Create GridSearchCV object
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='f1_macro',␣
 ↪n_jobs=-1)

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

# Get the best model
best_model = grid_search.best_estimator_

# Evaluate the best model
y_pred = best_model.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[8030 3328]
 [  66  241]]
              precision    recall  f1-score   support

           0       0.99      0.71      0.83     11358
           1       0.07      0.79      0.12       307

    accuracy                           0.71     11665
   macro avg       0.53      0.75      0.47     11665
```

```
weighted avg       0.97      0.71      0.81     11665
```

### 1.0.5 Modeling

[41]:
```
# Initialize a dictionary to hold metrics
metrics = {'Model': [], 'Accuracy': [], 'Precision': [], 'Recall': [], 'F1␣
↪Score': [], 'CV Mean F1 Score': []}

# Define the models to compare
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'KNN': KNeighborsClassifier(),
    'SVM': SVC()
}

# Create a function to build pipelines for each model
def create_pipeline(model):
    return ImbPipeline(steps=[
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=42)),
        ('classifier', model)
    ])

# Train and evaluate each model
for name, model in models.items():
    print(f"Training {name}...")
    pipeline = create_pipeline(model)
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    print(f"Results for {name}:")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print("\n")

     # Extract classification report metrics
    report = classification_report(y_test, y_pred, output_dict=True)

    # Collect metrics
    metrics['Model'].append(name)
    metrics['Accuracy'].append(report['accuracy'])
    metrics['Precision'].append(report['macro avg']['precision'])
    metrics['Recall'].append(report['macro avg']['recall'])
    metrics['F1 Score'].append(report['macro avg']['f1-score'])

    # Perform cross-validation and collect mean F1 score
    cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring='f1_macro')
```

```python
    metrics['CV Mean F1 Score'].append(cv_scores.mean())
```

```
Training Logistic Regression…
Results for Logistic Regression:
[[8015 3343]
 [  67  240]]
              precision    recall  f1-score   support

           0       0.99      0.71      0.82     11358
           1       0.07      0.78      0.12       307

    accuracy                           0.71     11665
   macro avg       0.53      0.74      0.47     11665
weighted avg       0.97      0.71      0.81     11665



Training KNN…
Results for KNN:
[[10157  1201]
 [  169   138]]
              precision    recall  f1-score   support

           0       0.98      0.89      0.94     11358
           1       0.10      0.45      0.17       307

    accuracy                           0.88     11665
   macro avg       0.54      0.67      0.55     11665
weighted avg       0.96      0.88      0.92     11665



Training SVM…
Results for SVM:
[[9316 2042]
 [ 102  205]]
              precision    recall  f1-score   support

           0       0.99      0.82      0.90     11358
           1       0.09      0.67      0.16       307

    accuracy                           0.82     11665
   macro avg       0.54      0.74      0.53     11665
weighted avg       0.97      0.82      0.88     11665
```

**Extract metrics for all models**

[42]:
```
# Convert the metrics dictionary to a DataFrame
metrics_df = pd.DataFrame(metrics)

# Print a clean table using tabulate
print("Model Comparison Metrics:")
print(tabulate(metrics_df, headers='keys', tablefmt='pretty'))
```

```
Model Comparison Metrics:
+---+---------------------+--------------------+--------------------+-----------
---------+---------------------+---------------------+
|   |        Model        |      Accuracy      |     Precision      |
Recall        |      F1 Score       |  CV Mean F1 Score   |
+---+---------------------+--------------------+--------------------+-----------
---------+---------------------+---------------------+
| 0 | Logistic Regression | 0.7076725246463781 | 0.5293464739697475 |
0.7437144849904185 | 0.47399089678081396 | 0.40369748858948373 |
| 1 |         KNN         | 0.8825546506643807 | 0.5433477664724456 |
0.6718854766948119 | 0.5522494986532505  | 0.49179508249090775 |
| 2 |         SVM         | 0.8162023146163737 | 0.5402012149372046 |
0.7439836347753568 | 0.5286682513315014  |  0.483589250530267  |
+---+---------------------+--------------------+--------------------+-----------
---------+---------------------+---------------------+
```

[43]:
```
%%capture
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
!pip install nbconvert
```

[ ]:
```
%shell jupyter nbconvert --to pdf '/content/drive/MyDrive/ADS 504 Group 3 Final␣
↪Project/Group3_FinalProject.ipynb'
```