project2

April 10, 2025

1 Machine Learning in Python - Project 2

Due Friday, Apr 11th by 4 pm.

Gustavo Marroig Arias, Fahad Bahzad, Omer Karahasanoglu, Hariaksh Pandya

1.1 Setup

[1]:
```
# 1. Loading required libraries

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns

# Plotting defaults and seed
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80
seed = 1

# sklearn modules
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.metrics import make_scorer, f1_score, accuracy_score, recall_score, precision_score,
confusion_matrix, roc_curve, auc, precision_recall_curve
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PolynomialFeatures, FunctionTransformer
from imblearn.pipeline import Pipeline as ImPipeline
from imblearn.under_sampling import RandomUnderSampler
```

[2]:
```
# 2. Defining plotting functions that will be used througout

def histogram_plot(data, feature):
    # Plot for 'default' target in dark red with transparency
    sns.histplot(data=data[data['target'] == 'default'], x='feature', color='darkred', label='Default',
kde=False, stat='percent', bins=30, alpha=0.6)

    # Plot for 'prepaid' target in dark green with transparency
    sns.histplot(data=data[data['target'] == 'prepaid'], x='feature', color='darkgreen', label='Prepaid',
kde=False, stat='percent', bins=30, alpha=0.6)

    # Customize the chart
    plt.title(f'Overlayed Histograms of {feature} as Percentages', fontsize=14, weight='bold')
    plt.xlabel(f'{feature}', fontsize=12)
    plt.ylabel('Percentage', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.legend(title='Target', loc='upper left', bbox_to_anchor=(1.01, 1), borderaxespad=0)
    plt.tight_layout()
    plt.show()
```

```python
def stacked_plot(data, feature, colors):
    # Calculate the counts for each combination of 'feature' and 'target'
    counts = data.groupby(['target', 'feature'], observed=False).size().unstack(fill_value=0)

    # Calculate percentages within each target category
    percentages = counts.div(counts.sum(axis=1), axis=0) * 100
    percentages = percentages.reindex(['prepaid', 'default'])

    # Track the bottom position for stacking
    bottoms = np.zeros(len(percentages.index))

    for i, category in enumerate(percentages.columns):
        plt.bar([0,1], percentages[category], label=category, bottom=bottoms, color=colors[i])
        bottoms += percentages[category]  # Update the bottom position for stacking

    # Annotate with percentage values
    for i, target in enumerate(percentages.index):
        cumulative_height = 0
        for category in percentages.columns:
            value = percentages.loc[target, category]
            cumulative_height += value
            if value > 5:  # Add condition to only annotate values > 5%
                plt.text(i, cumulative_height - value / 2,  # Position in the middle of the bar
                         f"{value:.1f}%",
                         ha='center', va='center', color='white', fontsize=12)

    # Customize the chart
    plt.xticks([0,1], percentages.index)
    plt.xlabel('Target Variable (Prepaid/Default)', fontsize=12)
    plt.ylabel('Percentage')
    plt.title(f'Stacked Percentage Distribution of {feature}', fontsize=14, weight='bold')
    plt.legend(title='Categories', loc='upper left', bbox_to_anchor=(1.01, 1), borderaxespad=0)
    plt.tight_layout()
    plt.show()

def roc_pr_curves(pipeline, baseline, X_test, y_test):

    # Convert target labels to binary
    y_test_binary = np.where(y_test == "default", 1, 0)

    models = {"Main Model": pipeline, "Baseline Model": baseline}
    curve_types = {"ROC Curve": {"func": roc_curve, "x_label": "False Positive Rate (FPR)", "y_label": "True
Positive Rate (TPR)"},
                   "Precision-Recall Curve": {"func": precision_recall_curve, "x_label": "Recall",
"y_label": "Precision"}}

    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    for ax, (curve_name, curve_data) in zip(axes, curve_types.items()):
        for name, model in models.items():
            y_prob = model.predict_proba(X_test)[:, 1]
            curve_vals = curve_data["func"](y_test_binary, y_prob)

            if curve_name == "ROC Curve":
                fpr, tpr, _ = curve_vals
                auc_score = auc(fpr, tpr)
                ax.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})", linewidth=2)
                if name == "Baseline Model":
                    ax.plot([0, 1], [0, 1], linestyle="--", color="gray", label="Chance")

            else:
                precision, recall, _ = curve_vals
                ax.plot(recall, precision, label=name, linewidth=2)

        ax.set_xlabel(curve_data["x_label"])
        ax.set_ylabel(curve_data["y_label"])
        ax.set_title(curve_name)
        ax.legend()

    plt.tight_layout()
    plt.show()

def cm_plot(pipeline, baseline, X_test, y_test):
    y_test_binary = np.where(y_test == "default", 1, 0)
    models = {"Main Model": pipeline, "Baseline Model": baseline}
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
    for ax, (name, model) in zip(axes, models.items()):
        y_pred = model.predict(X_test)
        cm = confusion_matrix(y_test_binary, y_pred)

        sns.heatmap(cm, annot=True, fmt='d', xticklabels=['Prepaid','Default'],
yticklabels=['Prepaid','Default'], cmap='Blues', ax=ax)
        ax.set_title(name)
        ax.set_xlabel("Predicted Labels")
        ax.set_ylabel("True Labels")

    # Shared legend
    handles, labels = axes[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc="upper right")

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

[3]:

```
# 3. Loading the data
dataset = pd.read_csv("freddiemac.csv", low_memory=False)
```

2 Introduction

Accurately assessing credit risk is crucial for Freddie Mac. As a leading government-sponsored entity, Freddie Mac plays a vital role in the U.S. housing market by purchasing and guaranteeing mortgages. However, the ability to predict mortgage defaults remains a key challenge. To support decision-making, we have conducted an in-depth analysis of loan-level credit performance data provided by the Federal Housing Finance Agency (FHFA) to develop a classification model to distinguish between prepaid and defaulted mortgages. The primary objective of this project is to construct a robust and well-validated classification model that can effectively classify historical loans (i.e., whether they have defaulted or were prepaid). By doing so, we can provide Freddie Mac with actionable insights into the characteristics and risk factors that contribute to loan defaults. Furthermore, while the focus of the project is on distinguishing between default and prepaid outcomes on inactive loans, we have also applied our developed model to identify active loans that may be at risk of default in the future.

In our exploratory data analysis, we systematically examined all features and our approach was to analyze the distribution of features across the prepaid and default mortgages to identify key differences. The most significant distinction was observed in FICO scores. Defaulted accounts consistently had lower FICO scores compared to prepaid accounts.

Our modelling approach employs logistic regression with L1 regularization to facilitate feature selection. Initially, we had over 900 features after one-hot encoding, but through a systematic tuning process using Stratified 5-fold cross-validation, we narrowed them down to just five key predictors that exhibit strong discriminatory power between default and prepaid cases. To determine the optimal penalty term, we evaluated various metrics including accuracy, recall, precision, and F1-score so as not to naively pick what maximizes recall. Additionally, we plotted the solution path for the model coefficients and found that FICO score is by far the most influential feature, which aligns with our exploratory data analysis findings. We then built a logisitic regression based on these features without any scaling or regularization to ease interpretability. Despite the refined feature selection, our final model exhibits only marginal improvement over the baseline model, which consists of logistic regression using only FICO score. This further reinforces the significance of FICO as the dominant predictor. However, our model's inclusion of additional features—particularly interactions with FICO—enables better differentiation of borderline cases.

3 Exploratory Data Analysis and Feature Engineering

To begin our exploratory data analysis (EDA), we first load the data and divide it into active and inactive loans, focusing exclusively on distinguishing prepaid mortgages from defaulted ones. After creating the dataset of inactive loans, we perform an 80/20 train-test split and specifically stratified sampling based on the `loan_status`. This is to guarantee that both the training and testing sets have a representative distribution of prepaid and defaulted loans.

[ ]:
```
# Splitting the dataset to focus on the inactive dataset
active = dataset[dataset['loan_status'] == 'active']
inactive = dataset[dataset['loan_status'] != 'active']

# Creating the training and testing datasets
X_train, X_test = train_test_split(inactive, test_size=0.2, random_state=seed,
stratify=inactive['loan_status'])
y_train, y_test = X_train['loan_status'], X_test['loan_status']
X_train, X_test = X_train.drop(columns=['loan_status']), X_test.drop(columns=['loan_status'])
```

3.1 Missing data analysis

Using pandas `.info()`, we investigated the dataset and observed missing values in the columns `cd_msa`, `flag_sc`, `rr_ind`, and `id_loan_rr`. Upon deeper analysis and consulting the data brief, we discovered these are not genuinely missing values but are encoded representations:

- `cd_msa`: Represents the Metropolitan Statistical Area (MSA) code, with nulls indicating either non-MSA areas or unknown locations.
- `flag_sc`: Identifies loans that exceed conforming loan limits, where nulls represent non-conforming loans.
- `rr_ind`: Indicates whether a loan is part of the Relief Refinance Program, with nulls representing non-participating loans.
- `id_loan_rr`: Identification number for the Relief Refinance Program and is populated only for loans that are part of the program.

To simplify interpretation, we will use feature engineering during our pipeline process such that for `cd_msa`, further investigation revealed no relationship between the specific codes and our target variables. However, there could be a distinction based on whether the property is in an MSA or not. To capture this, we will encode `cd_msa` as binary: **1** for loans located in an MSA and **0** for non-MSA or unknown areas. For `flag_sc`, we will apply one-hot encoding, converting the column into binary values: **1** for conforming-loans and **0** for non-conforming loans (`NaN` values will be treated as **0**). Finally, for `rr_ind`, we will apply one-hot encoding, with **1** indicating Relief Refinance loans and **0** indicating non-participating loans (`NaN` values will be treated as **0**). Note that we will exclude `id_loan_rr` from further analysis as it is irrelevant (it is just an identification number).

More importantly, after consulting the dataset documentation, we noticed that there are some encoded values that are actually missing but are encoded as 9s (e.g., FICO scores that are not available are 9999). Dealing with these was customized for each feature and will be explained.

3.2 Variables

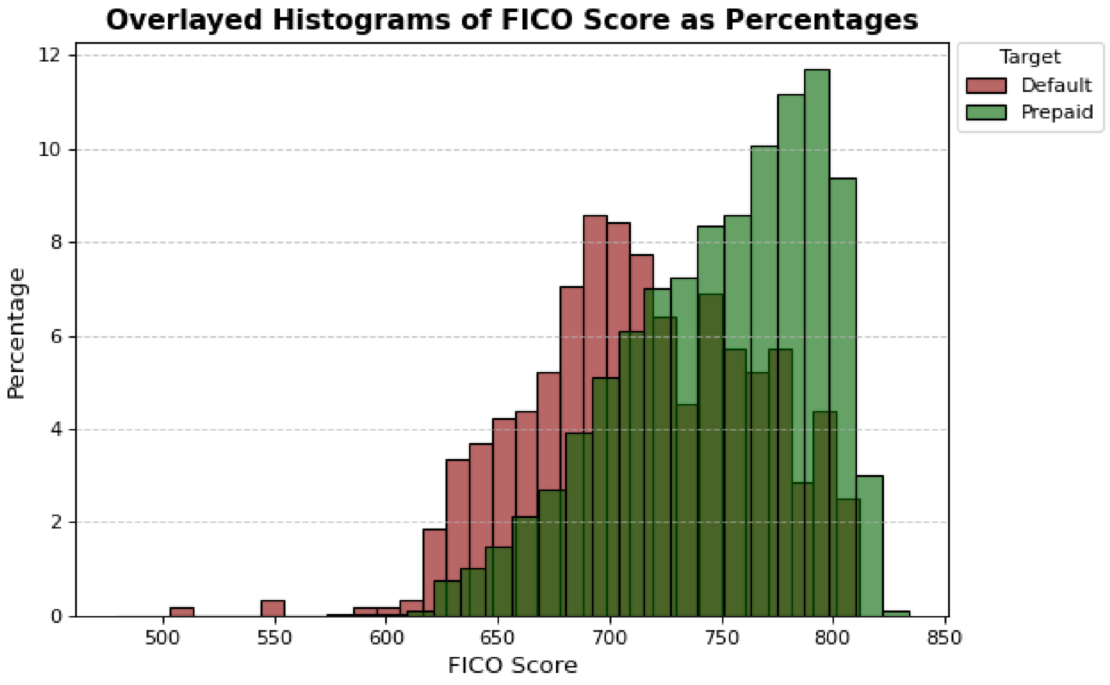
In this section, we explore all relevant variables and examine various plots to uncover meaningful insights. This analysis includes assessing the distribution of key features and identifying patterns related to default. Additionally, we address feature engineering steps tailored to each variable noting that all of these steps will be implemented within our pipeline.

3.2.1 FICO Score

The first feature we explore is the FICO score which is a critical indicator in credit risk. The FICO score, which ranges from 300 to 850, is a measure of a borrower's creditworthiness. It is typically calculated using factors like payment history, credit utilization, length of credit history, types of credit used, and recent credit inquiries. This score captures a snapshot of a person's financial health and their likelihood of repaying debts. Therefore, we anticipate that this will be one of the most highly discriminatory features. As can be seen from the overlayed histogram plots below (we have used percentage data of each group rather than actual count frequency to account for the imbalanced data), there is a clear distinction in the distribution of scores between default and prepaid mortgages.

Note that where the FICO score was 9999, as per the documentation this indicates the score is unavailable. We explored the missing values for the FICO scores further and they are minimal; only 23 observations missing (20 in prepaid and 3 in default). Given the vast majority of the data is intact (100,767 observations for prepaid and 597 for default), the missing values constitute a miniscule percentage of the training data. Thus, this justifies not investigating the missing values further. Therefore, we decided to impute them as mode values as a proxy (to be representative of a typical borrower).

[5]:
```
data = pd.DataFrame({'feature': X_train['fico'], 'target': y_train})
histogram_plot(data[data['feature'] != 9999],"FICO Score")
# Note: For illustrative purposes, we plot the historgram while excluding the 23 missing values but we
impute them in the pipeline.
```



3.2.2 Date and term variables

In our exploration of `dt_first_pi`, `dt_matr`, and `orig_loan_term`, we found that these variables were internally consistent, with no discrepancies. As in the loan term (`orig_loan_term`) aligned perfectly with the difference between `dt_first_pi` and `dt_matr`, confirming the accuracy of our dataset. We then examined the distribution of mortgage origination years, finding that all observations fell within the five-year window of 2017 to 2021. This limitation is critical, as it affects the generalizability of our findings and we must be cautious when extending our conclusions beyond this period. We discuss this further at the end of our report.

Furthermore, we investigated whether systemic patterns existed regarding loan term length and default status. Our findings indicated no substantial differences, with 30-year loans accounting for 84% of defaults and 83% of non-defaults, and shorter-term loans (<30 years) making up 16%of defaults and 17% of non-defaults. This suggests a roughly representative balance. Most loans had a term of 360 months (83,634 observations), followed by 180 months (12,126 observations) and 240 months (3,855 observations), with only a small portion falling outside these categories. Given these findings, we concluded that further investigation into loan term distributions was unnecessary.

3.2.3 First time home buyer

The `flag_fthb` variable indicates whether a borrower—or one among a group of borrowers—qualifies as a first-time homebuyer. This is an important variable to investigate because first-time homebuyers may have less experience, making them more susceptible to default. This is because homeownership comes with financial responsibilities such as property taxes, maintenance costs, and unexpected expenses, which inexperienced buyers may struggle to manage.

One particularly intriguing aspect of our findings is that we initially assumed a majority of borrowers would fall under the first-time homebuyers category. However, this dataset reveals that most borrowers are not classified as first-time homebuyers. This is important to note because it means that the dataset may not be representative of typical mortgages since according to the National Mortgage Database, 43.2% of borrowers are first-time homebuyers whereas in our dataset, it is approximately half that number (FHFA, 2025). This suggests that repeat buyers play a larger role in this dataset which should be taken into consideration.

Upon diving deeper into the data, we found that the percentage of first-time homebuyers within the **default class** is **28.1%**, while in the **prepaid class**, this percentage is lower at **21.7%**. This difference aligns with our hypothesis that first-time homebuyers are more likely to experience mortgage default. This finding is important because it suggests that being a first-time homebuyers may be a risk factor for mortgage default. It is possible that this variable interacts with other features such as FICO credit score, which could provide further insight into borrower risk. We plan to explore these potential interactions in future analyses.

Note that in our analysis of the `flag_fthb` variable, the documentation indicates that missing values are encoded as 9. While we did not find any missing values in our training dataset, our pipeline includes the creation of a dedicated “Missing” level via a one-hot encoder, ensuring that any missing values will be appropriately categorized if they arise if the model was trained on a different dataset.

3.2.4 Location variables

Following from our discussion in the missing data analysis, when applying the proposed encoding, we observed that **15.2%** of **default loans** were associated with properties in **non-MSA or unknown areas**, compared to **8.9%** for **prepaid loans**. This suggests that properties outside MSAs may carry a higher risk of default, which warrants further exploration. We believe this variable is important to consider because it indirectly captures economic prosperity and stability. Borrowers residing in MSAs are more likely to have access to better job opportunities, financial stability, and economic growth, which could impact mortgage outcomes. By incorporating this feature, we aim to reflect these underlying socioeconomic patterns in our analysis.

We initially explored `zipcode` as a variable but found it to be too detailed for meaningful analysis, given the granularity of the data. Instead, we aggregated default rates at the state level using the `st` variable to determine if there were any significant geographic patterns. While certain states exhibited higher or lower default rates, further investigation revealed that these variations were largely attributable to differences in loan counts rather than underlying risk factors. The highest default rate observed was 1.29%, which is insufficient to indicate a systemic pattern for some states. If default rates were very large and not close to each other, it could suggest that borrowers in certain states are inherently riskier, warranting further investigation. However, given the relatively low variation, we conclude that state-level default rates do not serve as a strong predictive factor in our analysis and incorporating them would increase our features substantially with minimal value.

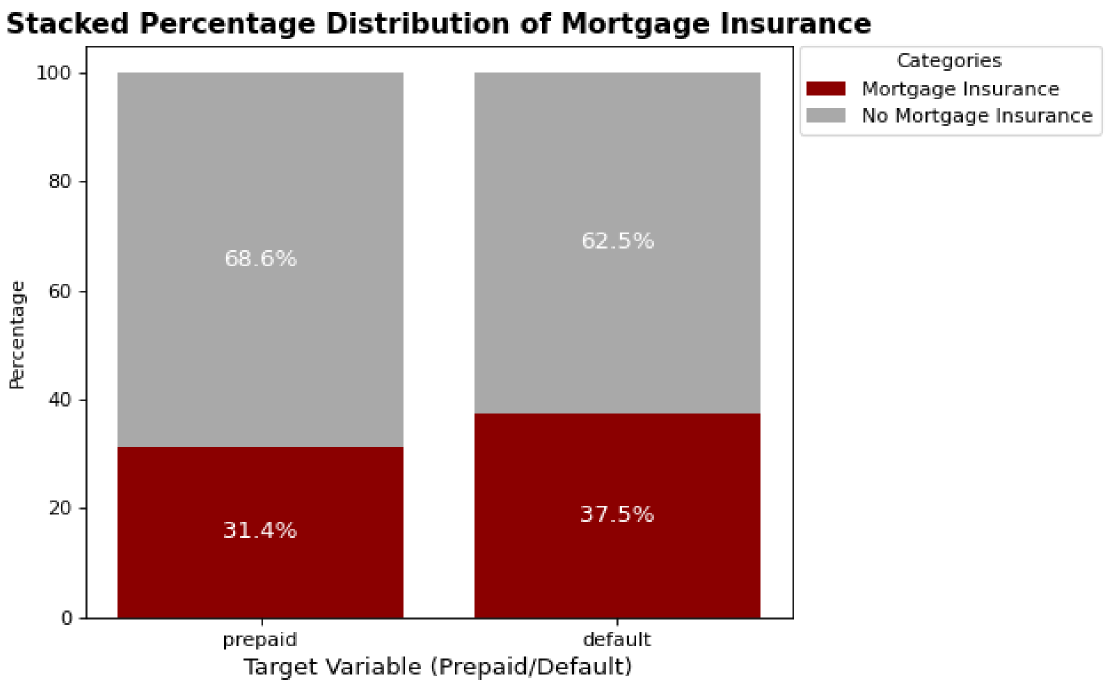
3.2.5 Mortgage insurance

We investigated the variable `mi_pct`, which represents the percentage of loss coverage provided by a mortgage insurer at the time of Freddie Mac's purchase of the loan. Values reported as less than 1% or greater than 55% are marked as "Not Available" (999), and loans without mortgage insurance are indicated by a value of zero. Our analysis of its distribution revealed no significant differences across the dataset. However, an important observation emerged when categorizing loans into two groups: those with mortgage insurance versus those without (0%). We found that defaulted mortgages tend to have mortgage insurance more somewhat more frequently compared to non-defaulted loans. This suggests that lenders may take out mortgage insurance on loans they perceive as higher risk, potentially due to underwriting assessments that capture hidden risk factors not explicitly reflected in other variables. Recognizing this pattern, we constructed a new feature that differentiates between insured and non-insured mortgages to capture this implicit risk assessment by financial institutions.

Note that amounts of mortgage insurance reported that are less than 1% or greater than 55% are disclosed as "Not Available", which is indicated by the value 999. However, because we are using this feature as an indicator flag, these values **still represent mortgage insurance percentages** and will therefore be categorized accordingly rather than being treated as missing (we interpret that less than 1% still contains mortgage insurance even if it is minimal since we cannot disect this category even further with the limited information).

We also explored the variable `mi_cancel_ind`, which indicates whether mortgage insurance was reported as canceled after Freddie Mac's purchase of the mortgage loan. If a loan did not originally have mortgage insurance, this field is marked as Not Applicable. Our analysis revealed that loans with canceled mortgage insurance were minimal, and the distribution did not exhibit significant explanatory power on top of the `mi_pct` variable. As a result, we determined that this feature does not provide meaningful additional information for our modeling approach.

[6]:
```
data = pd.DataFrame({'feature': X_train['mi_pct'], 'target': y_train})
data['feature'] = data['feature'].apply(lambda x: "Mortgage Insurance" if x != 0 else "No Mortgage
Insurance")
stacked_plot(data, "Mortgage Insurance", colors=['darkred','darkgray'])
```



3.2.6 Number of units

The feature cnt_units denotes whether a mortgage is for a one-, two-, three-, or four-unit property, with 99 indicating unavailable data. Our exploration of the distribution across default and prepaid cases revealed that the vast majority of mortgages, **98.1% of prepaid and 95.1% of default cases**, are for one-unit properties. Given the limited percentage of mortgages beyond one unit, we opted against stratification and instead categorized them as **either one-unit or multi-unit** to prevent overfitting. While cnt_units alone may not possess significant explanatory power to distinguish between default and prepaid, its interaction with other factors could be valuable for example, a high FICO score borrower taking a mortgage for more than one unit may present a heightened risk.

Moreover, the feature uses 99 to indicate "Not Available." In our analysis, we did not find any missing data within our training dataset. However, for robustness, we included a mechanism in our pipeline to handle potential missing values should the model be trained on a different dataset. Specifically, the preprocessing step will categorize 99 as a distinct level, ensuring missing data is appropriately handled in case the model is trained on a dataset with missing values.

3.2.7 Occupancy status

The variable occpy_sts identifies whether a mortgage corresponds to an owner-occupied property, a second home, or an investment property. Our analysis revealed that a larger proportion of defaulted loans (vs. prepaid) were associated with investment properties(11.6% vs. 7.2%), which aligns with the expectation that such properties may inherently carry higher risk. This could be attributed to factors such as reliance on rental income, market fluctuations, or differences in borrower intent compared to primary residences. Given these observations, we incorporated occpy_sts into the model, specifically for interaction effects. This allows us to capture scenarios where investment property ownership may amplify other risk factors. For instance, a borrower with a high FICO score taking a loan on an investment property could exhibit a different risk profile compared to an owner-occupied loan under similar conditions. For the occupancy_status variable, we did not find any missing data in our training dataset. However, to ensure robustness, we implemented a mechanism in our pipeline to handle missing values as denoted by 9 should they be encountered in another training dataset.

3.2.8 Combined loan to value ratio

We anticipate that the combined loan-to-value ratio will be one of the strongest predictors of mortgage default. Economically, a higher CLTV indicates that the borrower has less equity in their home, which can indicate potential problems. This is because when homeowners have limited equity, they are less likely to prioritize mortgage payments during financial distress, as they have less financial stake in retaining ownership. This makes high-CLTV mortgages more sensitive to market downturns—if property values decline, borrowers with little equity may find themselves underwater, owing more than their home's worth, which significantly raises their probability of default.

Upon exploring the data, we noticed distinct groupings in CLTV values, which is understandable given regulatory frameworks, banking risk metrics, and loan structuring practices. Financial institutions classify loans based on CLTV thresholds to determine risk exposure, and underwriting guidelines often segment loans into risk tiers accordingly. Given this natural segmentation, we categorized CLTV into three distinct levels as part of our feature engineering:

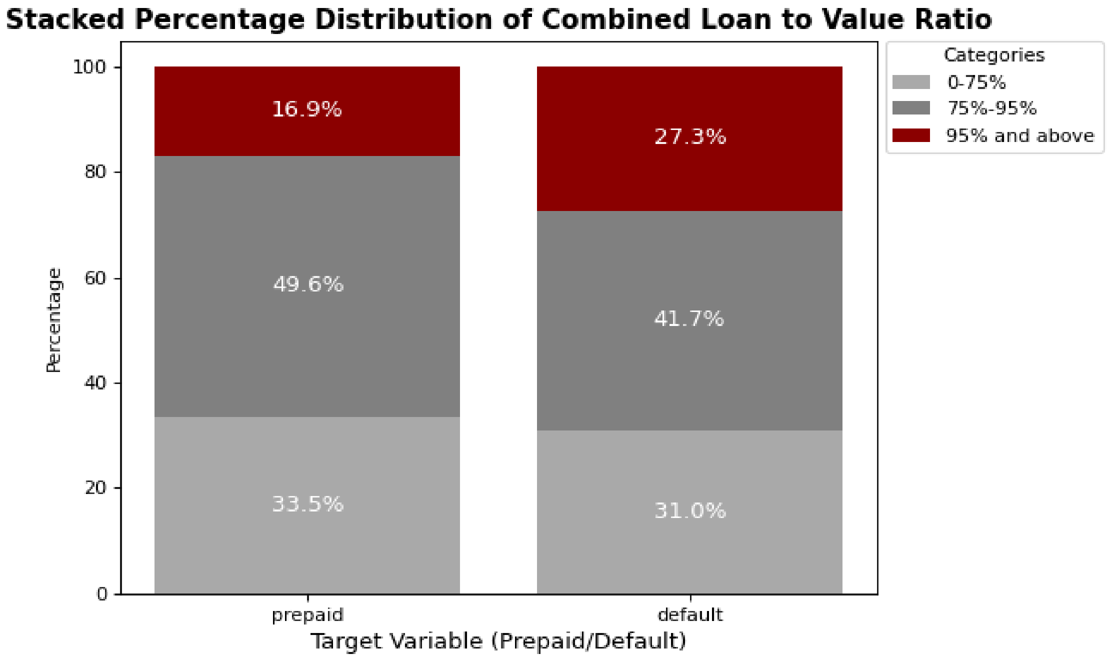
- **0-75% (Low Risk)**: Generally considered very safe, as borrowers have substantial equity.
- **75%-95% (Moderate Risk)**: Somewhat riskier but still widely accepted in conventional lending.
- **Above 95% (High Risk)**: Highly risky, as borrowers have minimal equity, making them vulnerable to financial instability.

Additionally, we observed outliers in defaulted loans with CLTV values above 100%, meaning these borrowers owe more than their home's worth. As shown in the plot below, **27.3% of defaulted mortgages have CLTV above 95%**, compared to only **16.9% in the prepaid class**, highlighting its importance as a predictive variable

As a note, after comparing the original loan to value ratio and the combined loan to value ratio, we found the definitions to be very similar, with the primary difference being that CLTV incorporates secondary financing while LTV does not. Based on our analysis of the training data, out of 101,364 rows, only 2,260 had different values for LTV and CLTV. We further investigated whether the distribution of defaults was significantly overrepresented in cases where the two metrics differed, but our findings indicated that the proportions were consistent across both metrics. As a result, LTV does not provide additional useful information in this context, so we decided to use CLTV exclusively for simplicity and focus in the analysis.

Additionally, it is important to point out that the combined loan-to-value ratio had only one missing value, recorded as 999. Given that we only encountered 1 mortgage in the training dataset that had no cltv (indicated by 999), we therefore decided to impute it with the median when training (not mean because we want to avoid outliers).

[7]:
```
data = pd.DataFrame({'feature': X_train['cltv'], 'target': y_train})
data = data[data['feature']!=999]
bins = [0, 75, 95, float('inf')]
labels = ['0-75%', '75%-95%', '95% and above']
data['feature'] = pd.cut(data['feature'], bins=bins, labels=labels, right=False)
stacked_plot(data, "Combined Loan to Value Ratio", ['darkgray','gray','darkred'])
```



3.2.9 Debt-to-income ratio

The debt-to-income (DTI) ratio is a fundamental measure of a borrower's financial health and risk level. It represents the proportion of a borrower's monthly debt obligations relative to their gross monthly income. This metric is crucial because it directly impacts a borrower's ability to manage mortgage payments, and higher DTI ratios indicate a higher likelihood of financial distress. When borrowers allocate a significant percentage of their income toward debt repayment, they have less financial flexibility to handle unexpected expenses which can lead to increased default risk.

Given its strong predictive potential, we expect DTI to have high discriminatory power in distinguishing defaulted loans from prepaid mortgages. Instead of treating DTI as a continuous variable, we categorized it into three industry-standard risk levels:

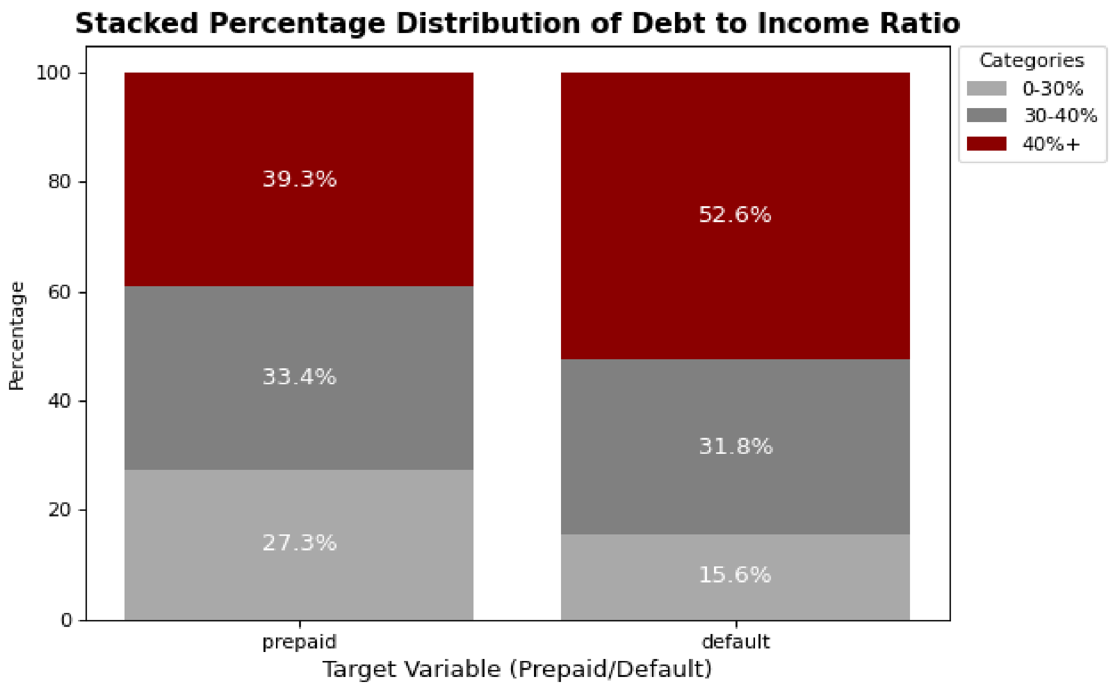
- **0-30% (Low Risk)**: Borrowers typically have ample income relative to debt payments, making defaults unlikely.
- **30-40% (Moderate Risk)**: Indicates higher debt burden but still within acceptable lending standards.
- **Above 40% (High Risk)**: Considered risky, as borrowers are allocating a substantial portion of their income toward debt repayment, leaving little room for financial shocks.

Additionally, in our dataset, 999 values correspond to **DTI ratios exceeding 65%**, meaning these are **not missing values but extreme risk cases**, which we classified into the **high-risk bucket**.

From our initial data exploration, DTI emerges as a key feature to include in our model. The plot below clearly shows a distinction between defaulted versus prepaid mortgages. Notably, **52.6% of defaulted mortgages had DTI above 40%**, compared to **only 39.3% in the prepaid class**, reinforcing the idea that borrowers with high debt burdens are significantly more likely to default. Conversely, **safer loans**—those with DTI under 30%—account for **27.3% of prepaid mortgages** but only **15.6% among defaults**.

Also, debt-to-income ratios greater than 65% are indicated as "Not Available" (999 per the documentation). However, since we are categorizing based on risk levels, these values inherently fit within the high-risk bucket rather than being treated as missing. As a result, we directly classify them within the high-risk category.

[8]:
```python
data = pd.DataFrame({'feature': X_train['dti'], 'target': y_train})
bins = [0, 30, 40, float('inf')]
labels = ['0-30%', '30-40%', '40%+']
data['feature'] = pd.cut(data['feature'], bins=bins, labels=labels, right=False)
stacked_plot(data, "Debt to Income Ratio", ['darkgray','gray','darkred'])
```



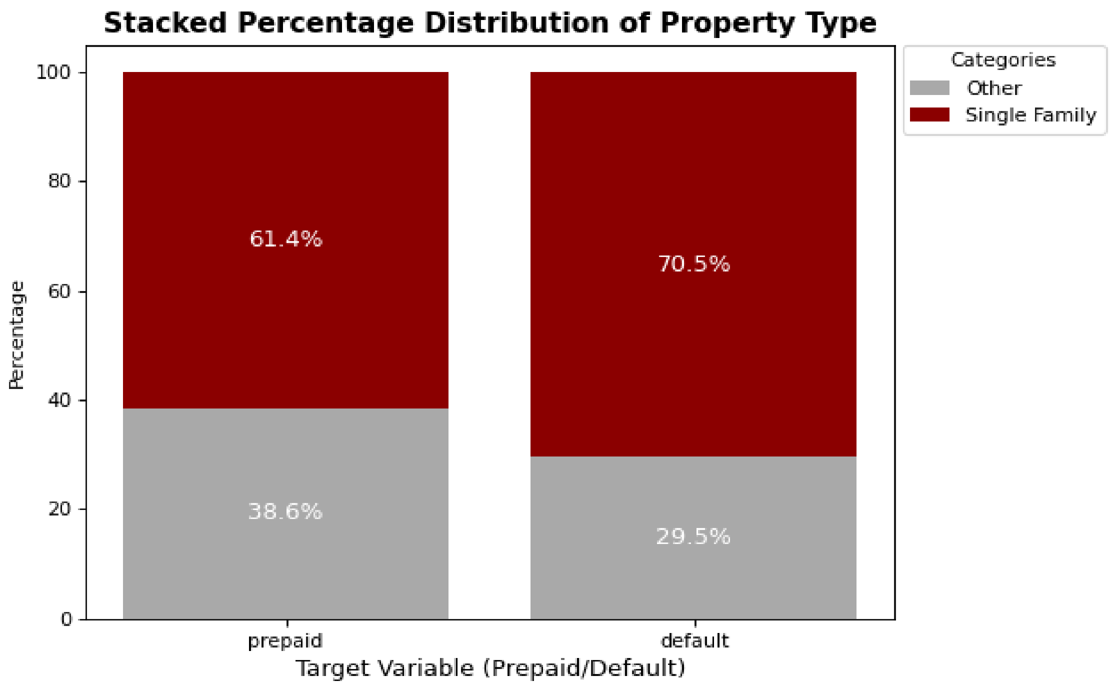
3.2.10 Original unpaid balance

We investigated the Original Unpaid Balance feature, which represents the remaining mortgage balance at the time of loan origination, rounded to the nearest $1,000. We observed that defaulted mortgages tend to have slightly lower unpaid balances compared to prepaid mortgages. This could be attributed to several factors: smaller loan amounts may indicate borrowers with lower financial flexibility, which increases vulnerability to default if income constraints or unexpected expenses arise. Conversely, loans with higher original balances are often associated with borrowers with stronger creditworthiness, which could explain why prepaid loans tend to have slightly higher values. Given its potential role in loan performance, we opted to include it in our model as a numerical feature in our analysis. While the direct impact of it alone may be subtle, examining its interactions with other features could provide deeper insights into borrower behavior and risk assessment which we will investigate when selecting features for our final model.

3.2.11 Property type

In our investigation of the property type (`prop_type`) variable, which categorizes properties secured by mortgages into condominium (CO), planned unit development (PU), cooperative share (CP), manufactured home (MH), and single-family home (SF), we found a key distinction between default and prepaid classes. Specifically, **SF properties are more prevalent in the default class (70.5%) compared to the prepaid class (61.4%)**, which aligns with the general trend that most mortgages are issued for single-family homes. Upon deeper analysis of the other categories, we observed that it was planned unit developments (PU) that was the driver behind this difference. PU properties constitute **29.6% of the prepaid category but only 19.8% of defaults**. Given this insight, we simplified the property type variable by categorizing it as “Single Family” or “Other” to simplify our modelling approach since there is minimal information gain by having all five levels and it will substantially increase our feature space unnecessarily. Additionally, while our training split contained no missing data (denoted as 99 per the documentation), we ensured robustness by incorporating a mechanism in our pipeline to classify any future missing values, which would be properly adapted within the one-hot encoder.

[9]:
```
data = pd.DataFrame({'feature': X_train['prop_type'], 'target': y_train})
data['feature'] = data['feature'].apply(lambda x: "Single Family" if x == "SF" else "Other")
stacked_plot(data, "Property Type", colors=['darkgray','darkred'])
```



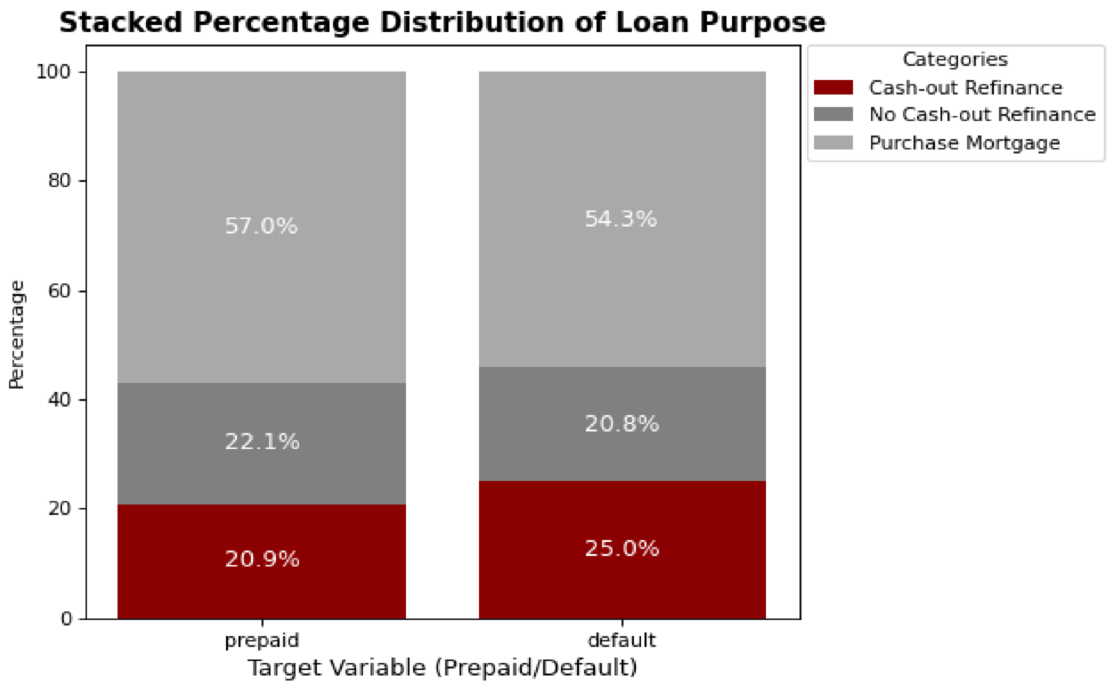
3.2.12 Loan purpose

This feature indicates whether a mortgage is a Cash-out Refinance mortgage, No Cash-out Refinance mortgage, Refinance mortgage (not specified), Purchase mortgage, or Not Available. While we did not observe extreme differences across categories, we noticed a higher proportion of Cash-out Refinance mortgages among defaulted loans. A cash-out refinance allows homeowners to replace their existing mortgage with a new loan that is **larger than the original balance**, enabling them to take out the difference in cash. This type of refinancing is typically used for debt consolidation, home improvements, or other expenses. However, it can also indicate financial distress, as borrowers who tap into their home equity may be **at higher risk of default**, especially if they are struggling with debt or economic uncertainty.

Since cash-out refinancing may be a potential risk factor, we incorporated mortgage type into our pipeline as a categorical variable using one-hot encoding. Additionally, we investigated possible interactions between mortgage type and other features to assess whether this classification has predictive power in identifying high-risk loans. Further feature selection will determine the significance of this variable in our final model.

The variable `loan_purpose` uses 9 to indicate "Not Available." In our training dataset, we did not encounter any missing values, so no immediate adjustments were required. However, for robustness, we designed our pipeline to handle potential missing values in future datasets.

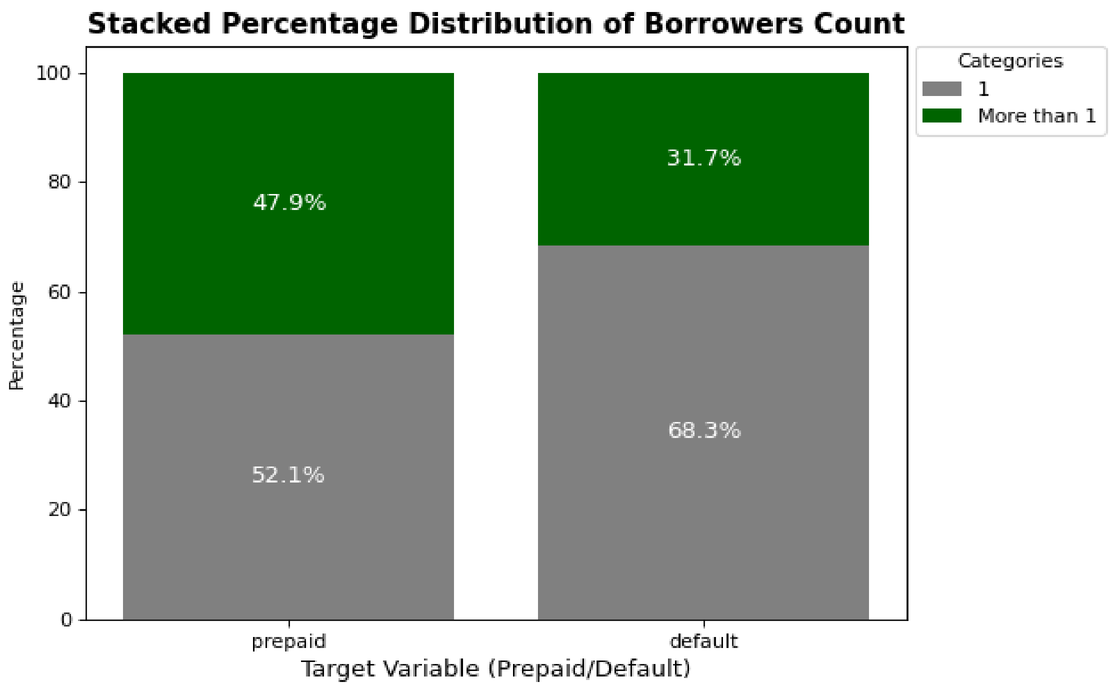
[10]:
```
data = pd.DataFrame({'feature': X_train['loan_purpose'], 'target': y_train})
data['feature'] = data['feature'].map({'C': 'Cash-out Refinance','N': 'No Cash-out Refinance','R':
'Refinance (not specified)','P': 'Purchase Mortgage'})
stacked_plot(data, "Loan Purpose", colors=['darkred', 'gray', 'darkgray'])
```



3.2.13 Number of borrowers

We explored the variable `cnt_borr`, which represents the number of borrowers obligated to repay the mortgage note. To simplify the analysis, we categorized it into two groups: single borrower versus more than one borrower. As expected, loans with multiple borrowers were significantly more common among prepaid loans (47.9%) compared to defaulted loans (31.7%), which aligns with economic intuition—having multiple borrowers generally reduces risk, as income sources are diversified, making repayment more stable. Given its potential discriminatory power, we included `cnt_borr` as a feature in our analysis and applied one-hot encoding to capture its influence on default and prepayment behavior.

[11]:
```
data = pd.DataFrame({'feature': X_train['cnt_borr'], 'target': y_train})
data['feature'] = data['feature'].apply(lambda x: "1" if x == 1 else "More than 1")
stacked_plot(data, "Borrowers Count", colors=['gray','darkgreen'])
```



3.2.14 Seller, service, and channel entities

Our initial goal was to examine whether certain origination channels, sellers, or servicers had a notable impact on loan performance, particularly default rates. The idea was that certain entities might have **different risk assessment strategies**, focusing on higher-risk borrowers or specific mortgage classes. If a seller or servicer consistently originated or managed a disproportionate number of **defaulted loans**, this could signal an underlying risk factor tied to their practices. However, after conducting an initial exploration, we found no meaningful distinction between defaulted and prepaid loans based on origination channel. Additionally, while the dataset contained 45 unique sellers, none displayed an unusually high default rate. Similarly, no clear patterns emerged among servicers that could effectively discriminate between defaulting and prepaid loans. Including these features as categorical features would significantly expand the feature space, introducing computational complexity without clear classification benefits. Therefore, given the lack of strong discriminative power in these variables, we opted to exclude them from further modeling.

3.2.15 Exceeding conforming loan limits

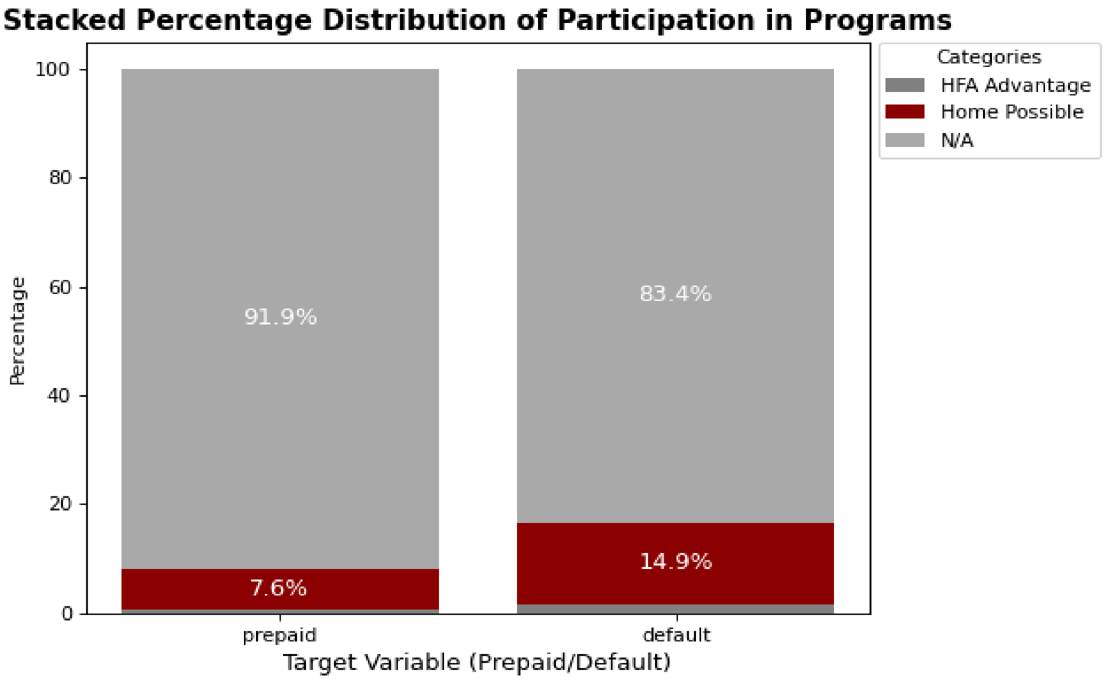
We explored the feature `flag_sc`, which identifies mortgages that exceed conforming loan limits. Upon analyzing loan performance, we observed that for default cases, 96.1% of loans did not exceed conforming loan limits, and for prepaid cases, 95.7% also did not. We further investigated potential **interactions** with other variables to determine if `flag_sc` could play a role in identifying riskier loan segments. However, our analysis did not reveal strong discriminatory power in distinguishing mortgage outcomes.

3.2.16 Participation in programs

We explored the variable `program_ind`, which represents participation in different mortgage programs, including Home Possible (H), HFA Advantage (F), Refi Possible (R), and Not Available (9).

Our analysis revealed that defaulted mortgages are nearly twice as likely to be part of the Home Possible program compared to prepaid loans (14.9% vs. 7.6%), suggesting a potential risk factor. After conducting some research, we found out that the Home Possible program is designed to assist low- to moderate-income borrowers by offering flexible financing options, including low down payments (as little as 3%) and reduced mortgage insurance requirements (Freddie Mac, 2025). While this program provides valuable opportunities for homeownership, it may also indicate higher-risk loans, as borrowers in this category could face financial instability. Given this insight, we incorporated **program_ind** into our analysis to assess its impact on loan performance and potential interactions with other risk factors.

[12]:
```
data = pd.DataFrame({'feature': X_train['program_ind'], 'target': y_train})
data['feature'] = data['feature'].replace({'H': 'Home Possible', 'F': 'HFA Advantage', 'R': 'Refi Possible',
'9':'N/A'})
stacked_plot(data, "Participation in Programs", colors=['gray', 'darkred', 'darkgray'])
```



3.2.17 HARP Loans

As mentioned earlier, we explored the **rr_ind** variable and found that it contained a significant number of missing values but after consulting the documentation, we discovered that missing values in this context represents that it is not part of Freddie Mac's Relief Refinance program. Approximately 1% of the prepaid mortgages were part of the program while 4.9% of the defaulted mortgages were part of the program. Despite being a limited subset, as only 1,046 loans from the training dataset were part of the Relief Refinance Program, it could serve as a signal for higher risk. More importantly, when consulting the documentation further we found that there is a more critical featuer that we can engineer from this data.

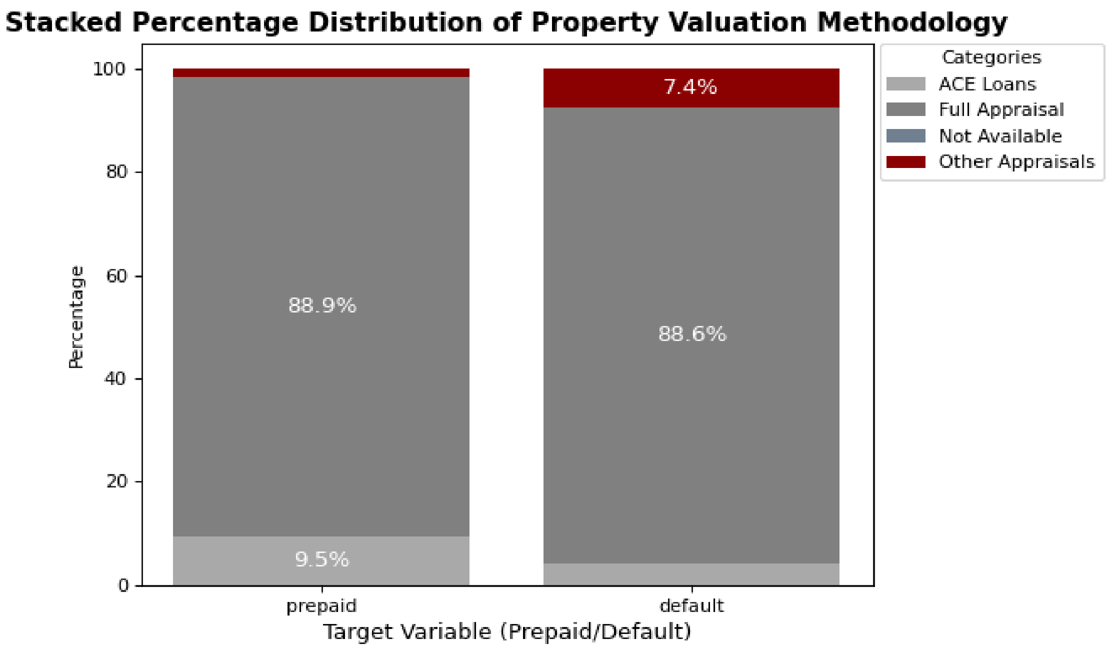
Loans that qualify as both Relief Refinance and have an Original Loan-to-Value above 80 are classified as HARP loans. The *Home Affordable Refinance Program (HARP)* was a government initiative designed to assist homeowners who owed more on their mortgage than the market value of their home. These borrowers often face significant challenges refinancing through conventional programs as they pose a higher risk for default.

In the training data set, we found that there are 216 HARP eligible loans and of these, more than 5% have defaulted. This is almost 10 times the regular default rate in the entire data set which suggests that this feature could have significant discriminatory power. Therefore, we decided to incorporate it into our pipeline.

3.2.18 Property valuation methodology

We explored the variable `property_val`, which indicates the method used to obtain a property appraisal, if any. The values range from ACE Loans, Full Appraisal, Other Appraisals (including desktop, drive-by, external, and AVM), ACE+PDR, and Not Available. Our analysis revealed that Other Appraisals are notably more common among defaulted loans, accounting for **7.4% of all defaults**, whereas their presence in prepaid loans is negligible. This observation suggests a potential economic explanation—perhaps these appraisal methods are less accurate or indicate lower-quality loans where borrowers are unable to secure a full appraisal. Given this insight, we decided to include `property_val` as a feature for further exploration. To effectively incorporate it into our modeling process, we applied one-hot encoding to capture the different levels of appraisal methods and assess their impact on loan performance. Also, `property_val` designates 9 as representing "Not Available". Although our training dataset did not contain any missing values, we integrated measures into our pipeline to accommodate such cases should they arise in future training datasets.

[13]:
```
data = pd.DataFrame({'feature': X_train['property_val'], 'target': y_train})
data['feature'] = data['feature'].replace({1: 'ACE Loans', 2: 'Full Appraisal',3: 'Other Appraisals', 4:'ACE
+ PDR', 9:'Not Available'})
stacked_plot(data, "Property Valuation Methodology", colors=['darkgray', 'gray', 'slategray','darkred'])
```



3.2.19 Unimportant features

In our analysis, we also explored several variables that turned out to be useless for this specific case because there was no distinction in any of the mortgages. One such feature was `ppmt_pnlty`, which indicates whether the borrower has ever been obligated to pay a penalty upon prepayment. While this could have been a valuable indicator suggesting that borrowers making prepayments might be at lower risk of default, all loans in our dataset had no prepayment penalties. Similarly, the `prod_type`, which denotes whether a mortgage is fixed-rate or floating-rate. However, all loans in our dataset were fixed-rate mortgages, meaning this feature provided no variability. This is a critical limitation for our model that we discuss further towards the end of the report because it means that our model may not be generalizable on floating rate mortgages. Additionally, the `io_ind` feature, which signals whether a loan initially required interest-only payments, was also considered because it could highlight a potential difference between interest-only payment loans. However, all loans in our dataset did not have interest-only periods. Finally, we excluded variables such as `id_loan`, a unique identifier for each loan, and `id_loan_rr`, the pre-relief refinance loan sequence number, as they serve only as unique indicators and contain no meaningful information relevant to default risk.

4 Model Fitting and Tuning

4.1 Baseline Model

For our baseline comparison, we have chosen a simple Logistic Regression model with no penalty term, using only the FICO score as the predictor. This decision is based on our observation that FICO score appears to be the strongest discriminatory feature between prepaid and default mortgages. Our goal is to determine whether incorporating additional features can surpass the predictive strength of FICO alone. If our main model significantly outperforms this baseline, it would demonstrate the added value of leveraging broader data inputs beyond just credit score.

[14]:
```
# Create the pipeline for the baseline model with undersampling
baseline = ImPipeline([('preprocessor', ColumnTransformer(transformers=[('fico_num',
SimpleImputer(missing_values=9999, strategy='mean'), ['fico'])],
                                                              remainder='drop')),
                        ('undersampler', RandomUnderSampler(random_state=seed)),
                        ('model', LogisticRegression(max_iter=200, penalty=None, fit_intercept=True,
random_state=seed))])

# Fit the baseline to X_train and y_train
baseline.fit(X_train, np.where(y_train == "default", 1, 0));
```

4.2 Main model

Our modeling approach is designed to be entirely reproducible. This guarantees that no manual manipulations are applied to the data outside of the pipeline. We employ logistic regression as the primary modeling technique, utilizing L1 regularization in the initial phase to assist in feature selection. This approach enables the identification of the most relevant predictors while mitigating the risk of overfitting, which enhances the generalizability of the final model. Once the important features are determined, we will use a standard logistic regression model with no penalty and no scaling to ease interpretation.

The decision to use logistic regression is because it can provide soft labels i.e., probability of default estimates rather than just binary classification. This probability measure is crucial because regulators require it in capital adequacy assessments as well as when setting loan default provisions (Federal Reserve System, 2020). Moreover, logistic regression is further motivated by the need for model interpretability. Rather than employing black-box models such as neural networks and relying on post-hoc explainable methods, we prioritize direct transparency. In highly regulated sectors such as the financial services sector, understanding the key drivers of risk estimation is fundamental for accountability and compliance. This was further validated by multiple research papers such as Alonso et. al (2020) which confirmed that logistic regression is commonly used in credit risk modeling to estimate the Probability of Default as it aligns with Basel II's Internal Ratings-Based (IRB) framework.

Note that we standardize all features including categorical ones so that all features contribute equally to the penalty which would and thus, prioritize features based on importance and not scale. For categorical features, although standardizing them technically imposes higher penalties on features that are rare when one-hot encoded, we do this consciously in order to pick the most important categorical features since we have a lot of different ones as well as interactions among them. Additionally, when using one-hot encoding, we are intentionally not dropping any feature by using `drop=None`. This is to ensure that regularization applies equal penalties to each category or level to help in unbiased feature selection.

[ ]:
```
# Custom functions and class to be fed as part of the pipeline

def categorize_cltv(X):
    bins = np.array([0, 75, 95, np.inf])
    labels = np.array(['0-75%', '75%-95%', '95% and above'])
    indices = np.digitize(X, bins, right=False) - 1
    return labels[indices]

def categorize_dti(X):
    bins = np.array([0, 30, 40, np.inf])
    labels = np.array(['0-30%', '30%-40%', '40% and above'])
    indices = np.digitize(X, bins, right=False) - 1
    return labels[indices]

class harp_transformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = pd.DataFrame(X, columns=['rr_ind', 'ltv'])  # Convert to DataFrame
        X = X.assign(rr_ind=X['rr_ind'].fillna('N'))  # Replace NaN in 'rr_ind'
        X['ltv'] = X['ltv'].astype(float)  # Ensure 'ltv' is numeric
        return ((X['rr_ind'] == 'Y') & (X['ltv'] >= 80)).map({True: 'Y', False: 'N'}).to_numpy().reshape(-1,
1)

    def get_feature_names_out(self, input_features=None):
        return ["harp_loan"]

# Defining the transformations for the features chosen as discussed earlier in the exploratory data analysis
section

cd_msa_preprocess = Pipeline([('value_replacer', FunctionTransformer(lambda X: np.where(pd.isna(X), "No /
Unknown", "Yes").astype(object), feature_names_out = 'one-to-one')),
                              ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])

mi_pct_preprocess = Pipeline([('mi_pct_transform', FunctionTransformer(lambda X: np.where(X == 0, "No",
"Yes").astype(object), feature_names_out = 'one-to-one')),
                              ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])

cnt_units_preprocess = Pipeline([('cnt_units_transform', FunctionTransformer(lambda X: np.where(X == 99,
"Not Available", np.where(X > 1, "More than 1", "1")), feature_names_out = 'one-to-one')),
                                 ('encoder', OneHotEncoder(drop = None, sparse_output=False))])

occpy_sts_preprocess = Pipeline([('imputer', SimpleImputer(missing_values='9', strategy='constant',
fill_value='Not Available')),
                                 ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown
= 'ignore'))])

dti_pct_preprocess = Pipeline([('dti_pct_transform', FunctionTransformer(lambda X: np.where(X == 999, "Not
Available", "Available"), feature_names_out = 'one-to-one')),
                               ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])
```

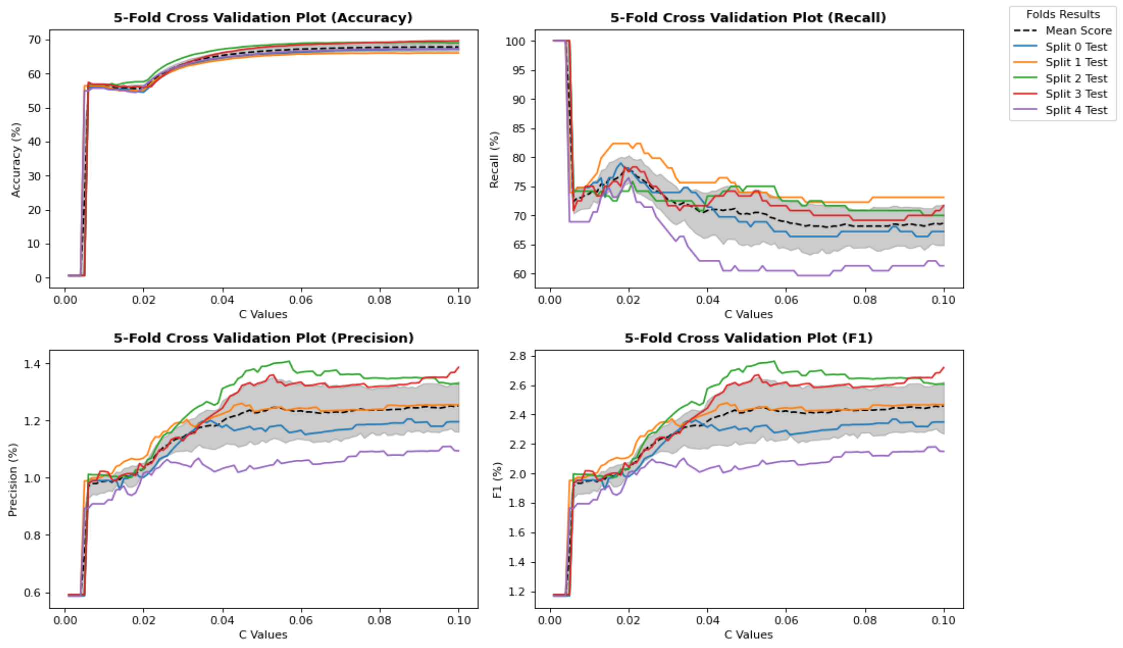
```
prop_type_preprocess = Pipeline([('imputer', SimpleImputer(missing_values=99, strategy='constant',
fill_value='Not Available')),
                                 ('value_replacer', FunctionTransformer(lambda X: np.where(X == 'SF', 'SF',
'Other'), feature_names_out = 'one-to-one')),
                                 ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown
= 'ignore'))])

loan_purpose_preprocess = Pipeline([('imputer', SimpleImputer(missing_values=9, strategy='constant',
fill_value='Not Available')),
                                   ('encoder', OneHotEncoder(drop = None, sparse_output=False,
handle_unknown = 'ignore'))])

cnt_borr_preprocess = Pipeline([('cnt_borr_transform', FunctionTransformer(lambda X: np.where(X == 99, "Not
Available", np.where(X > 1, "More than 1", "1")), feature_names_out = 'one-to-one')),
                                ('encoder', OneHotEncoder(drop = None, sparse_output=False))])

program_ind_preprocess = Pipeline([('imputer', SimpleImputer(missing_values='9', strategy='constant',
fill_value='N/A')),
                                   ('encoder', OneHotEncoder(drop = None, sparse_output=False,
handle_unknown = 'ignore'))])

property_val_preprocess = Pipeline([('value_replacer', FunctionTransformer(lambda X: np.vectorize({1: 'ACE
Loans', 2: 'Full Appraisal', 3: 'Other Appraisals', 4: 'ACE + PDR', 9: 'Not Available'}.get)(X),
feature_names_out = 'one-to-one')),
                                   ('encoder', OneHotEncoder(drop = None, sparse_output=False,
handle_unknown = 'ignore'))])

rr_ind_preprocess = Pipeline([('imputer', SimpleImputer(strategy='constant', fill_value='N')),
                              ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])

harp_preprocess = Pipeline([('harp_transformer', harp_transformer()),
                            ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])

flag_sc_preprocess = Pipeline([('imputer', SimpleImputer(strategy='constant', fill_value='N')),
                               ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])

cltv_preprocess = Pipeline([('imputer', SimpleImputer(missing_values=999, strategy='median')),
                            ('value_replacer', FunctionTransformer(categorize_cltv, feature_names_out =
'one-to-one')),
                            ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])

dti_preprocess = Pipeline([('value_replacer', FunctionTransformer(categorize_dti, feature_names_out =
'one-to-one')),
                           ('encoder', OneHotEncoder(drop = None, sparse_output=False, handle_unknown =
'ignore'))])

# Combine the transformations in the preprocessing aspect of the model and drop all other columns
preprocessor = ColumnTransformer(transformers=[('fico_num', SimpleImputer(missing_values=9999,
strategy='most_frequent'), ['fico']),
                                               ('cd_msa_cat', cd_msa_preprocess, ['cd_msa']),
                                               ('flag_fthb', OneHotEncoder(drop = None, sparse_output=False,
handle_unknown = 'ignore'), ['flag_fthb']),
                                               ('mi_pct_cat', mi_pct_preprocess, ['mi_pct']),
                                               ('cnt_units_cat', cnt_units_preprocess, ['cnt_units']),
                                               ('occpy_sts_cat', occpy_sts_preprocess, ['occpy_sts']),
                                               ('cltv_cat', cltv_preprocess, ['cltv']),
                                               ('dti_cat', dti_preprocess, ['dti']),
                                               ('orig_upb_num', 'passthrough', ['orig_upb']),
                                               ('int_rt_num', 'passthrough', ['int_rt']),
                                               ('prop_type_num', prop_type_preprocess, ['prop_type']),
                                               ('loan_purpose_num', loan_purpose_preprocess,
['loan_purpose']),
                                               ('cnt_borr_cat', cnt_borr_preprocess, ['cnt_borr']),
                                               ('program_ind_cat', program_ind_preprocess, ['program_ind']),
                                               ('property_val_cat', property_val_preprocess,
['property_val']),
                                               ('rr_ind_cat', rr_ind_preprocess, ['rr_ind']),
                                               ('harp_cat', harp_preprocess, ['rr_ind','ltv']),
                                               ('flag_sc_cat', flag_sc_preprocess, ['flag_sc'])],
                                remainder='drop')
```

```
# Create the model
# Note: We are not initializing the penalty here but that will be a tuned parameter in the next step,
#       this code serves to just set up the structure of the pipeline.
model = LogisticRegression(max_iter=200, penalty='l1', solver='liblinear', C=1.0, fit_intercept=True,
random_state=seed)

# Include interactions between all the features to investigate them further
interaction = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

# Create the machine learning pipeline combining all the steps
pipeline = ImPipeline([('preprocessor', preprocessor),
                       ('interaction', interaction),
                       ('scaler', StandardScaler()),
                       ('undersampler', RandomUnderSampler(random_state=seed)),
                       ('model', model)])

# Fit the pipeline to X_train and y_train
pipeline.fit(X_train, np.where(y_train == "default", 1, 0))

# Extract the feature names from the preprocessor in the pipeline
feature_names = np.concatenate([pipeline.named_steps['preprocessor'].transformers_[0][1].
↪get_feature_names_out(['fico']),
                                pipeline.named_steps['preprocessor'].transformers_[1][1].
↪get_feature_names_out(['cd_msa']),
                                pipeline.named_steps['preprocessor'].transformers_[2][1].
↪get_feature_names_out(['flag_fthb']),
                                pipeline.named_steps['preprocessor'].transformers_[3][1].
↪get_feature_names_out(['mi_pct']),
                                pipeline.named_steps['preprocessor'].transformers_[4][1].
↪get_feature_names_out(['cnt_units']),
                                pipeline.named_steps['preprocessor'].transformers_[5][1].
↪get_feature_names_out(['occpy_sts']),
                                pipeline.named_steps['preprocessor'].transformers_[6][1].
↪get_feature_names_out(['cltv']),
                                pipeline.named_steps['preprocessor'].transformers_[7][1].
↪get_feature_names_out(['dti']),
                                pipeline.named_steps['preprocessor'].transformers_[8][1].
↪get_feature_names_out(['orig_upb']),
                                pipeline.named_steps['preprocessor'].transformers_[9][1].
↪get_feature_names_out(['int_rt']),
                                pipeline.named_steps['preprocessor'].transformers_[10][1].
↪get_feature_names_out(['prop_type']),
                                pipeline.named_steps['preprocessor'].transformers_[11][1].
↪get_feature_names_out(['loan_purpose']),
                                pipeline.named_steps['preprocessor'].transformers_[12][1].
↪get_feature_names_out(['cnt_borr']),
                                pipeline.named_steps['preprocessor'].transformers_[13][1].
↪get_feature_names_out(['program_ind']),
                                pipeline.named_steps['preprocessor'].transformers_[14][1].
↪get_feature_names_out(['property_val']),
                                pipeline.named_steps['preprocessor'].transformers_[15][1].
↪get_feature_names_out(['rr_ind']),
                                pipeline.named_steps['preprocessor'].transformers_[16][1].
↪get_feature_names_out(['rr_ind','ltv']),
                                pipeline.named_steps['preprocessor'].transformers_[17][1].
↪get_feature_names_out(['flag_sc'])])

# Include all feature names with the interactions
all_features = pipeline.named_steps['interaction'].get_feature_names_out(feature_names)
```

We tuned the model parameter C (the inverse of the L1 regularization strength) using 5-fold cross-validation. Given the imbalanced nature of the dataset, we employed StratifiedKFold sampling to ensure each fold maintained a consistent proportion of default and prepaid mortgages. To assess the effect of different values of C, we plotted key classification metrics (accuracy, recall, precision, and F1-score) across individual folds as well as their mean values (to visualize uncertainity). Accuracy provides a general measure of correctness but is less informative in imbalanced settings. Recall quantifies the model's ability to correctly identify default cases, which is crucial for risk assessment, while precision reflects how many predicted defaults are actually correct. The F1-score integrates both recall and precision, providing a balanced metric in situations where false positives and false negatives are both important. While maximizing recall is a priority for our purposes to ensure that defaults are detected, an indiscriminate increase in recall can degrade overall model performance, as classifying all instances as default would yield 100 percent recall but poor accuracy. Thus, our approach balances recall with other metrics to ensure robust model generalization.

[16]:
```
# Grid of tuning parameters
C_values = np.linspace(0.001, 0.1, num=100)

# Create the folds randomly based on the notebook's random state
# and the number of folds chosen.
cv = StratifiedKFold(5, shuffle=True, random_state=seed)

# Create the grid search
scoring = {'accuracy': make_scorer(lambda y_true, y_pred: accuracy_score(y_true, y_pred) * 100,
greater_is_better=True),
           'f1': make_scorer(lambda y_true, y_pred: f1_score(y_true, y_pred, pos_label='default') * 100,
greater_is_better=True),
           'precision': make_scorer(lambda y_true, y_pred: precision_score(y_true, y_pred,
pos_label='default') * 100, greater_is_better=True),
           'recall': make_scorer(lambda y_true, y_pred: recall_score(y_true, y_pred, pos_label='default') *
100, greater_is_better=True),}

gs = GridSearchCV(pipeline,
                  param_grid = {'model__C': C_values},
                  cv = cv,
                  scoring = scoring,
                  refit = 'recall')

# Fit the grid search
gs.fit(X_train, y_train)

# Extract the full results as a dataframe
cv_results = pd.DataFrame(gs.cv_results_)

score_dict = {"Accuracy": cv_results.filter(regex='(split[0-9]+|mean)_test_accuracy').assign(C=gs.
↪param_grid['model__C']),
              "Recall": cv_results.filter(regex='(split[0-9]+|mean)_test_recall').assign(C=gs.
↪param_grid['model__C']),
              "Precision": cv_results.filter(regex='(split[0-9]+|mean)_test_precision').assign(C=gs.
↪param_grid['model__C']),
              "F1": cv_results.filter(regex='(split[0-9]+|mean)_test_f1').assign(C=gs.
↪param_grid['model__C'])}

# Create a 2x2 subplot
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Initialize an empty list to collect line objects for the legend
handles = []
labels = []

# Loop through the scores and plot in the corresponding subplot
for ax, (score_name, score_df) in zip(axes.flat, score_dict.items()):
    d = score_df.melt(id_vars=('C', f'mean_test_{score_name.lower()}'),
                      var_name='fold',
                      value_name=score_name.lower())

    line_black = sns.lineplot(x='C', y=score_name.lower(), color='black', data=d, linestyle='--',
label=f'Mean {score_name}', ax=ax)
    line_colored = sns.lineplot(x='C', y=score_name.lower(), hue='fold', data=d, ax=ax)

    ax.set_xlabel("C Values")
    ax.set_ylabel(f"{score_name} (%)")
    ax.set_title(f'5-Fold Cross Validation Plot ({score_name})', fontweight='bold')

    # Collect legend handles and labels from the last plotted lines
    if not handles:
        handles, labels = ax.get_legend_handles_labels()

    ax.get_legend().remove()

# Place a single legend outside the plot
```

```python
custom_labels = ["Mean Score", "Split 0 Test", "Split 1 Test", "Split 2 Test", "Split 3 Test", "Split 4
Test"]

# Apply the custom labels to the legend
fig.legend(handles, custom_labels, title='Folds Results', loc='upper right', bbox_to_anchor=(1.15, 1))

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

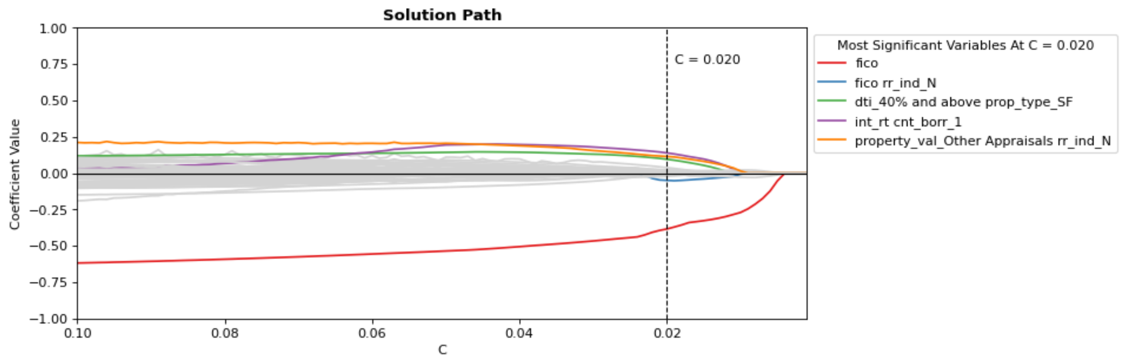


As we can see from the 5-fold cross-validation plots, accuracy, precision, and F1 score all increase as C increases, which is expected since C is the inverse of the L1 penalty; thus, a higher C corresponds to a lower penalty and thus, more features are included in the model which may lead to overfitting. However, in the context of classifying defaults in imbalanced data, we particularly care more about recall which is the measure of a model's ability to correctly identify positive instances because there is a higher financial cost to missing a default. While recall is crucial for detecting defaults, it must be considered alongside other metrics, as blindly maximizing recall could result in excessive false positives. Recall **peaks at around C = 0.02** and it appears to be well-defined in that neighborhood, suggesting that this value provides the best balance between correctly identifying positive instances and by looking at the other plots as well, we can see that at that value for C, overall accuracy and precision are also not very far off from their maximum. Note that the outlier performance close to $C = 0$ is just because at a tiny C value the model is essentially predicting the base case without parameters. Since we use undersampler, the model happens to just allocate all observations to the positive class i.e., default and hence by definition, we get 100% recall, so we disregarded it. Based on this analysis, we select the optimal $C = 0.02$, which corresponds to an L1 regularization parameter of 50.

Next, we plot the solution path of the features to analyze their importance in the model. This allows us to observe how coefficients change as the regularization strength varies, helping identify which features remain influential across different penalty values. By visualizing this path, we can effectively select the most significant features that contribute meaningfully to our predictions while filtering out less impactful ones. Note that we filter coefficients to be above 0.05, allowing us to prioritize features with stronger predictive power while avoiding noise from small, insignificant effects—especially critical given the large number of features (>900) under exploration. Since we standardized all features, each coefficient in the logistic regression model represents the effect of a 1 standard deviation increase in the corresponding feature on the log-odds of the outcome. This means that a coefficient of 0.05 indicates that increasing the feature by one standard deviation results in a multiplicative change of in the odds of the event occurring of `exp(0.05)` $\approx$ `1.05` i.e., approximately 5%. Note that in the next plot, we flip the x-axis to ease interpretability because a higher C corresponds to a lower penalty term.

[17]:
```
ws = [] # Store coefficients

# Loop through all the C_values
for C in C_values:

    # Update the model pipeline based on the current C value
    model = LogisticRegression(max_iter=100, penalty='l1', solver='liblinear', C=C, random_state=seed)
    pipeline = ImPipeline([('preprocessor', preprocessor),
                           ('interaction', interaction),
                           ('scaler', StandardScaler()),
                           ('undersampler', RandomUnderSampler(random_state=seed)),
                           ('model', model)])

    pipeline.fit(X_train, np.where(y_train == "default", 1, 0))

    # Create a temporary copy of the current model's coefficients and append
    # it to the coefficients storage list
    w_temp = np.copy(pipeline.named_steps['model'].coef_).reshape(-1)
    ws.append(w_temp)

# Create a data frame for plotting that has the solution path
sol_path = pd.DataFrame(data = ws, columns = all_features).assign(C = C_values).melt(id_vars = ('C'))
```

[18]:
```
# Identify significant features
chosen_C = C_values[19]
chosen_significance = 0.05
significant_features = sol_path[(sol_path['C'] <= chosen_C) & (abs(sol_path['value']) >=
chosen_significance)]['variable'].unique()

# Create a palette so that only the significant features are colored, and rest are just gray
colors = sns.color_palette("Set1", len(significant_features))
palette = {feature: color for feature, color in zip(significant_features, colors)}
palette.update({feature: 'lightgray' for feature in sol_path['variable'].unique() if feature not in
significant_features})

# Plot the data
fig, ax = plt.subplots(figsize=(10, 4))
sns.lineplot(x='C', y='value', hue='variable', data=sol_path, palette=palette, ax=ax)

# Filter the legend to include only the featuires that are significant
handles, labels = plt.gca().get_legend_handles_labels()
filtered_handles = [handle for handle, label in zip(handles, labels) if label in significant_features]
ax.legend(filtered_handles, significant_features, title=f'Most Significant Variables At C = {chosen_C:.3f}',
          loc='upper left', bbox_to_anchor=(1, 1))

# Add some visual enhancements to the plot
ax.set_title('Solution Path', fontweight='bold')
ax.set_xlabel('C')
ax.set_ylabel('Coefficient Value')
ax.set_xlim(max(C_values), min(C_values))
ax.set_ylim(-1, 1)
plt.axvline(x=0, color='black', linestyle='-', linewidth=1)
plt.axhline(y=0, color='black', linestyle='-', linewidth=1)

# Add a verticle for the chosen C value to illustrate it
plt.axvline(x=chosen_C, color='black', linestyle='--', linewidth=1)
plt.annotate(f'C = {chosen_C:.3f}', xy=(0.1, 0.75), xytext=(chosen_C-0.001, 0.75), fontsize=10,
color='black')
plt.show()
```



As we can see from the plot above, out of the 900+ features, only 13 features emerge as truly critical in influencing the model's predictions. Among these, the **FICO score** stands out as the most important determinant. This aligns with expectations, as FICO scores are the most direct measure in the dataset of creditworthiness. We now proceed to fit a logistic regression model using these variables. To ensure direct interpretability of the coefficients, we will not apply regularization or standardization to the features.

[22]:
```
# Select the features and define a function to add to the pipeline
selected_features = sol_path[(sol_path['C'] <= chosen_C) & (abs(sol_path['value']) >
chosen_significance)]['variable'].unique()

def select_features(dataset, feature_names):
    """
    Chooses specific features for the modelling step.

    >>> Inputs:
    dataset:             Numpy array of the dataset i.e., rows as observations
                         and columns as features with all interactions
    feature_names:       Numpy array of all the names of the features in the dataset
                         incl. all interactions

    >>> Outputs:
    new_dataset:    Numpy array of dataset after removing all comments except
                    those in selected_features.
    """

    # Loop through all the selected features and get their index from the feature names
    indicies = [np.where(feature_names == feature)[0][0] for feature in selected_features]

    # Return the new dataset with only the selected features
    new_dataset = dataset[:, indicies]

    return(new_dataset)

# Define the feature selection step before modelling
feature_selector = FunctionTransformer(select_features, kw_args={'feature_names': interaction.
↪get_feature_names_out(feature_names)})

# Define the model
model = LogisticRegression(max_iter=200, penalty=None, fit_intercept=True, random_state=seed)

### Recreate the pipeline with the feature selector without any standardization
pipeline = ImPipeline([('preprocessor', preprocessor),
                       ('interaction', interaction),
                       ('feature_selector', feature_selector),
                       ('undersampler', RandomUnderSampler(random_state=seed)),
                       ('model', model)])

# Fit the pipeline to X_train and y_train
pipeline.fit(X_train, np.where(y_train == "default", 1, 0))

# Extract the intercept and coefficients
intercept, coefficients = pipeline.named_steps['model'].intercept_, pipeline.named_steps['model'].coef_

# Create a DataFrame of features with corresponding coefficients
feature_coeff_df = pd.DataFrame({"Feature": selected_features, "Coefficient": coefficients[0]})
```

```python
# Add the intercept
intercept_df = pd.DataFrame({"Feature": "Intercept", "Coefficient": intercept}, index=[0])
parameters_df = pd.concat([feature_coeff_df, intercept_df], ignore_index=True)

# Display the result
parameters_df = parameters_df.set_index(["Feature"])

# Remove rows where values are 0
parameters_df = parameters_df[parameters_df["Coefficient"] != 0]

print(parameters_df)
```

```
                                          Coefficient
Feature
fico                                        -0.012993
fico rr_ind_N                               -0.001295
dti_40% and above prop_type_SF               0.645852
int_rt cnt_borr_1                            0.175151
property_val_Other Appraisals rr_ind_N       4.079480
Intercept                                    9.728340
```

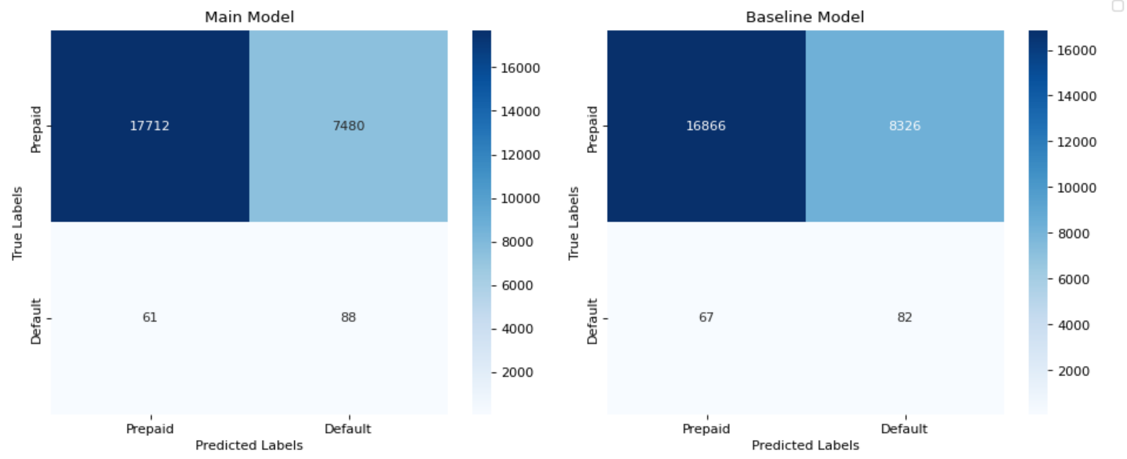
In our final model, each coefficient represents the change in log-odds for a **one-unit increase** in the corresponding feature, holding all other variables constant.

- **Intercept (9.728340)**: This defines the baseline log-odds of default when all features are zero. Given its large positive value, it suggests that the baseline model prediction is default if all features are 0.
- **fico (-0.012993)**: A one-point increase in FICO score reduces the log-odds of default by 0.012993. This means that borrowers with higher FICO scores have lower chances of default which makes sense because higher scores are associated with better borrowers.
- **fico rr_ind_N (-0.001295)**: This coefficient measures the interaction between the FICO score whether a borrower is part of the Relief Refinance Program, which was designed to help homeowners refinance their mortgages under more favorable terms, often due to financial hardship or risk of default. The indicator N signifies that the borrower was not in the program, meaning they likely had stronger financial stability and lower risk factors. The negative coefficient (-0.001295) suggests that for borrowers not enrolled in the program, the impact of FICO score on reducing default risk is slightly reinforced.
- **dti_40% and above prop_type_SF (0.645852)**: A borrower with a debt-to-income ratio above 40% and a mortgage on a single-family property experiences an increase in log-odds of default by 0.645852, meaning they are significantly more likely to default compared to those without these characteristics.
- **int_rt cnt_borr_1 (0.175151)**: For single borrowers, each percentage point increase in interest rate raises log-odds of default by 0.175151, suggesting that higher interest rates lead to a greater probability of default specifically for single borrowers.
- **property_val_Other Appraisals rr_ind_N (4.079480)**: This feature captures the interaction between alternative property valuation methods and whether a borrower was part of the Relief Refinance program. The rr_ind_N indicator means the borrower was not enrolled in the program, suggesting stronger financial characteristics. However, despite this more favorable borrower profile, the fact that their property valuation was conducted using alternative appraisal methods rather than a full standard valuation introduces an element of risk. The large positive coefficient indicates that when other appraisal methods are used for a mortgage not part of the Relief Refinance program, the log-odds of default increase significantly.

4.3 Results and diagnosis

To consider the results, we will first begin by analyzing the confusion matrix and computing different metrics from it. Note that we omit the code that computes these metrics from the report but explain the ratios. The main model achieves a slightly higher overall accuracy of 70.24% compared to the baseline accuracy of 66.88%. Accuracy represents the proportion of correctly classified instances out of all predictions, meaning the main model has a greater ability to correctly identify default and non-default cases. This improvement can be attributed to the inclusion of additional features beyond FICO, which allow the model to better capture subtle relationships between prepaid behavior and default likelihood, particularly for marginal cases. A key indication of this improvement is the reduction in false positives, where the main model misclassified 7,480 non-default cases compared to 8,326 in the baseline. Examining detailed metrics, we observe an increase in recall from 55.03% in the baseline to 59.06% in the main model, signifying better identification of actual default cases, which aligns with our goal of prioritizing recall. However, precision remains low at 1.16% versus 0.98% in the baseline, indicating that false positives are still prevalent, which can be problematic for practical decision-making. The F1-score, which provides a combination of precision and recall, improved slightly from 1.92% to 2.28%, suggesting a modest overall balance between identifying defaults and limiting false positives. Furthermore, the false positive rate (FPR) decreased from 33.05% in the baseline to 29.69% in the main model, meaning fewer non-default cases were incorrectly classified.

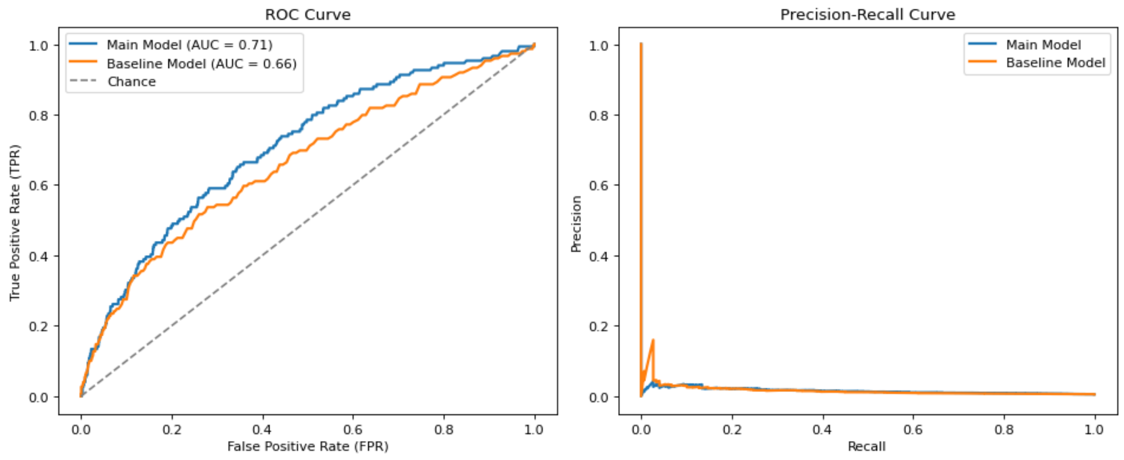
```
[20]: cm_plot(pipeline, baseline, X_test, y_test)
```



To further evaluate the performance of both models, we plotted the ROC curve and the Precision-Recall (PR) curve. The ROC curve illustrates the trade-off between True Positive Rate and False Positive Rate at different classification thresholds. Ideally, we want the curve to hug the top-left corner, indicating high recall with a low false positive rate. The straight diagonal line represents pure chance, meaning a classifier that performs no better than random guessing. We can also calculate the area under the curve (AUC) and an AUC of 1.0 indicates perfect classification, while an AUC of 0.5 means the model performs no better than random chance (the diagonal line in an ROC plot). In our case, the main model has an AUC of 0.71, while the baseline has 0.66, meaning both models are better than random guessing, but the main model shows a slight improvement in distinguishing defaults from non-defaults. However, as can be seen from the plots below, the difference between the curves is subtle, indicating that while the additional features contribute to improved differentiation, the gain is not substantial.

Similarly, the Precision-Recall curve helps evaluate how well the model balances the precision and recall trade-off, particularly in imbalanced datasets. The ideal curve would be higher across all recall values i.e., hugging the top right, signifying strong precision even at higher recall thresholds. Here, the PR curves of both models are nearly identical, with only slight improvements in precision for the main model. This indicates that while recall has increased slightly, precision remains very low, meaning many positive predictions are false alarms.

[21]:
```
roc_pr_curves(pipeline, baseline, X_test, y_test)
```



In addition, we further evaluated a wide range of approaches via a comprehensive grid search across 1,000+ hyperparameter combinations for four model types: Random Forest, Gradient Boosting, Decision Trees, Logistic Regression, and Neural Networks totaling over 5,000 models fit using 5-fold cross-validation. Given the results of these analyses, we did not identify a model that holistically outperforms our main model across all metrics that would justify discarding its advantages of interpretability and transparency as well as considering the expensive computational requirements to train it (i.e., Gradient Boosting and Neutral Networks). For example, deep decision trees did perform better in terms of recall but had worse accuracy.

5 Discussion & Conclusions

5.1 Findings and Implications

Our model identifies several critical risk factors that aid in assessing default likelihood. First, **FICO score** plays the most significant role, with higher scores reducing the probability of default, as expected—borrowers with strong credit scores are less likley to default. This effect is slightly strengthened for mortgages not part of the Relief Refinance program, as these borrowers likely did not require financial assistance and therefore exhibit more stable financial profiles.

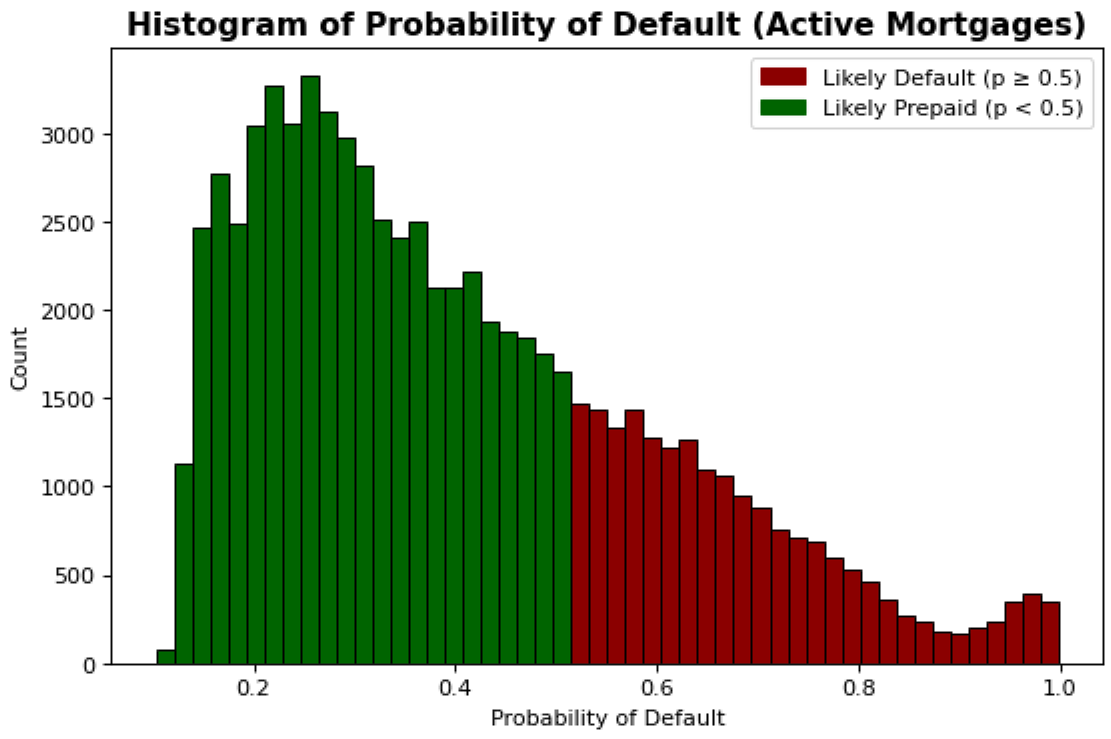
Second, **debt-to-income ratio (DTI) above 40% for single-family properties** increases the probability of default. This is intuitive since highly leveraged borrowers face greater financial constraints—most of their income goes toward debt payments, leaving little flexibility to absorb financial shocks such as unexpected expenses or income disruptions.

Third, **interest rates impact single borrowers**, where each percentage point increase in the interest rate raises the likelihood of default. This is understandable, as single borrowers typically have fewer financial buffers and thus, higher interest rates mean higher monthly mortgage payments which pressures these individuals and increasing default risk.

Lastly, **mortgages valued using other appraisal methods and not part of the Relief Refinance program** increase the probability of default. This effect is quite strong and captures a potential underlying risk that if the mortgages are not part of the Relief Refinance program (i.e., required assistance to refinance) but still used other appraisal methods (which are presumably cheaper) than a standard full appraisal raises a potential red flag.

To support effective decision-making, we applied our model to predict default probabilities on the active mortgages. As can be seen by the histogram below, most cases are predicted as non-default, with approximately 29% classified as likely to default which again highlights the limitation of our model in that it usually is more conservative and predicts many more defaults than what would most likely happen. However, these probability estimates allow for a more targeted approach to risk management i.e., cases with higher probabilities should be prioritized for further investigation, as the model is more confident in their likelihood of defaulting. Meanwhile, cases with mid-range probabilities warrant additional scrutiny as their classification remains uncertain.

[23]:
```python
inactive_probabilities = pipeline.predict_proba(active)[:, 1]

# Plot histogram
ax = sns.histplot(inactive_probabilities, bins=50, kde=False, edgecolor="black")

# Color bars conditionally based on probability of default
for patch in ax.patches:
    patch.set_facecolor("darkred" if patch.get_x() >= 0.5 else "darkgreen")

plt.xlabel("Probability of Default")
high_risk_patch = mpatches.Patch(color="darkred", label="Likely Default (p >= 0.5)")
low_risk_patch = mpatches.Patch(color="darkgreen", label="Likely Prepaid (p < 0.5)")
plt.legend(handles=[high_risk_patch, low_risk_patch], loc="upper right")
plt.title("Histogram of Probability of Default (Active Mortgages)", weight='bold', fontsize=14)
plt.show()
```



5.2 Limitations

One major limitation is that although the dataset comprises approximately 200,000 observations, its size can be misleading when considering the counts of loan status. The default mortgages are just 746 i.e., constitutes less than 0.4% of the data. This imbalance poses a risk of overfitting the model to the few default cases. The issue is further because we have to split the dataset into test-train splitting sets, which even though we stratify, it still means we train our model on even less defaults which exacerbates the risk of overfitting. Thus, it is important to be take caution in interpreting results, especially when generalizing predictions to broader populations. This exacerbates the trade-off between recall and precision and should therefore, be understood that predictions made by the model are typically much more conservative.

Building from that, one major limitation of our model stems from the decision to use undersampling to address the imbalance of the dataset. While this approach effectively reduces training time and helps prevent redundancy in the majority class, it inevitably results in the loss of valuable information. As removing a substantial portion of prepaid (the majority class) examples may lead to the model missing important patterns. However, while oversampling would allow us to retain all data, it introduces a major risk of overfitting, where the model could learn from duplicated examples of the default class rather than generalizing effectively to unseen data. Since our primary objective is to build a generalizable model, we prioritize avoiding overfitting and accept the trade-off of possible information loss.

Another key limitation of this analysis is that the data is confined to originations in the five-year period between 2017 and 2021. As a result, applying this model to mortgages initiated outside of this timeframe presents challenges due to shifting economic conditions, interest rate fluctuations, and evolving regulatory policies. For instance, the mortgage market is highly influenced by factors such as inflation, employment rates, and central bank policies. A period of low interest rates, like 2020, might lead to increased refinancing activity and housing demand, whereas a rising rate environment (like the one we have experienced between 2022 and 2024) could suppress affordability and borrowing capacity. Furthermore, external shocks—such as economic recessions could significantly alter mortgage behavior beyond the data period and hence, impact the generalizability of our model.

A further important limitation of our model is that it was trained on data solely of fixed-rate mortgages, meaning the findings cannot be directly applied to floating-rate mortgages. Fixed-rate loans maintain a constant interest rate throughout the loan term, making them less sensitive to short-term fluctuations in market conditions. In contrast, floating-rate mortgages, which have interest rates that adjust periodically based on market benchmarks, respond dynamically to economic changes. Because of this, borrowers with floating-rate mortgages may experience varying monthly payments over time, which impacts default risks differently than fixed-rate loans. As a result, we should be very cautious when interpreting the results of our model to mortgages with floating rates.

5.3 Potential Areas for Future Research

There are many avenues to enhance the model. Potential ideas include:

- **Macroeconomic Indicators**: External economic conditions play a significant role in influencing mortgage default rates. According to Ngene et al. (2016), macroeconomic indicators such as housing prices, unemployment rates, and interest rates significantly influence mortgage default rates. These should be considered in the future to capture exogenous economic trends that affect borrowers' ability to meet payment obligations.

- **Granular Creditworthiness Metrics**: Currently, the model relies on the FICO score as an aggregate representation of creditworthiness. While it is the most useful feature, the FICO score consolidates diverse credit history attributes, such as payment history, credit utilization, etc. Disaggregating this information could allow us to identify specific credit characteristics that better predict default risk. Additionally, exploring alternative weighting or combinations of these attributes could enhance model precision.
- **Detailed Employment Information**: Although debt-to-income ratio provides a snapshot of financial health, it does not account for nuances in employment status. Including more granular employment data, such as job type, skill level, and sector, could offer deeper insights. For instance, individuals employed in industries with higher volatility or lower skill requirements may face elevated default risks, even if their DTI ratios seem safe now.

6 Generative AI statement

Generative AI was used to debug code and assist with complicated styling of visualizations (e.g., how to multi-color histogram with patche and create dynamic stacked bar charts). It was also used to streamline finding potential research papers and citing them.

7 References

1. **Alonso, A., & Carbó, J. M.** (2020). *Understanding the performance of machine learning models to predict credit default: A novel approach for supervisory evaluation.* European Banking Authority Research Workshop. Retrieved from https://www.eba.europa.eu/sites/default/files/document_library/About%20Us/EBA%20Research%20Workshops/2020/Papers/936774/2.2%20Understanding%20the%20performance%20of%20machine%20learning%20models.pdf
2. **Federal Housing Finance Agency (FHFA).** (2025). *National Mortgage Database: New Residential Mortgage Statistics.* FHFA. https://www.fhfa.gov/data/dashboard/nmdb-new-residential-mortgage-statistics
3. **Federal Reserve System.** (2020). *Interagency guidance on credit risk review systems.* Federal Reserve System. https://www.federalreserve.gov/supervisionreg/srletters/SR2013.htm
4. **Ngene, G. M., Hassan, M. K., Hippler, W. J., & Julio, I.** (2016). *Determinants of mortgage default rates: Pre-crisis and crisis period dynamics and stability.* Journal of Housing Research, 25(1), 39-64. https://www.jstor.org/stable/24861826