Analyzing Mortgage Risks - Predicting Default Loans (2017-2020)

April 11, 2025

Rebecca Joy, Sayantan Bal, Sayli Patil, Samyukta Ramesh

Setup

[21]:
```
# Add any additional libraries or submodules below
import pandas as pd # Data libraries
import numpy as np

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

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

import sklearn
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪roc_auc_score, roc_curve, precision_recall_curve
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer, f1_score, roc_auc_score,␣
 ↪precision_score, recall_score, accuracy_score

from tabulate import tabulate
import missingno as msno

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

1 Introduction

This project explores loan-level data from Freddie Mac's Single-Family Loan-Level Dataset to develop a model that predicts whether a mortgage loan will be prepaid or defaulted. The dataset includes fully amortizing, fixed-rate residential loans acquired between 2017 and 2020, and contains key variables related to borrower profiles, loan terms, and property characteristics. The core objective is to build a binary classification model to support Freddie Mac's credit risk management processes by identifying early indicators of default risk. This is a high-impact problem in the financial services domain, where improved risk prediction can support smarter capital allocation, risk-sharing mechanisms, and policy evaluation.

The methodology begins with a baseline Logistic Regression model to provide interpretability and a reference for more complex models. More advanced ensemble techniques — Random Forest and XGBoost — are also implemented due to their robustness to non-linear interactions, missing values, and outliers. Given the imbalance between prepaid and defaulted cases, Synthetic Minority Oversampling Technique (SMOTE) is applied to improve class balance. The preprocessing pipeline includes missing value imputation, one-hot encoding of categorical variables, feature scaling where needed, and univariate outlier detection using the IQR method. Key risk factors include borrower credit score (`fico`), loan-to-value ratios (`ltv`, `cltv`), and debt-to-income ratio (`dti`). These findings offer valuable insights for Freddie Mac's internal risk models and future decision-making processes.

The model metrics were used to give predictions of those **active** loans that were at risk for defaulting. It revealed that out of **73,295 currently active loans**, approximately **2,569 loans (3.5%)** were identified as being at **high risk of default**. These predictions were made using a Random Forest classifier optimized for F1-score and recall, helping to better capture risk in minority class cases. This demonstrates the model's potential to not only identify key predictors of default but also to provide early warnings for at-risk active loans, enabling Freddie Mac to take proactive steps toward portfolio risk mitigation and loan performance monitoring.

[22]:
```
d = pd.read_csv("freddiemac.csv", low_memory=False)
d.shape
```

[22]:
```
(200000, 33)
```

2 Exploratory Data Analysis and Feature Engineering

2.1 Data Understanding

We proceed to split the data into training and test sets (80-20% split) to avoid data leakage when we're assessing the features for the model. The dataset contains a mixture of categorical and numerical features, as detailed below:

Below is a summary of the main features in the dataset used to build the classification model:

Feature Name	Description
`fico`	Credit score of the borrower (300–850). 9999 = not available or beyond the range.
`dt_first_pi`	First scheduled payment date (format: YYYYMM).

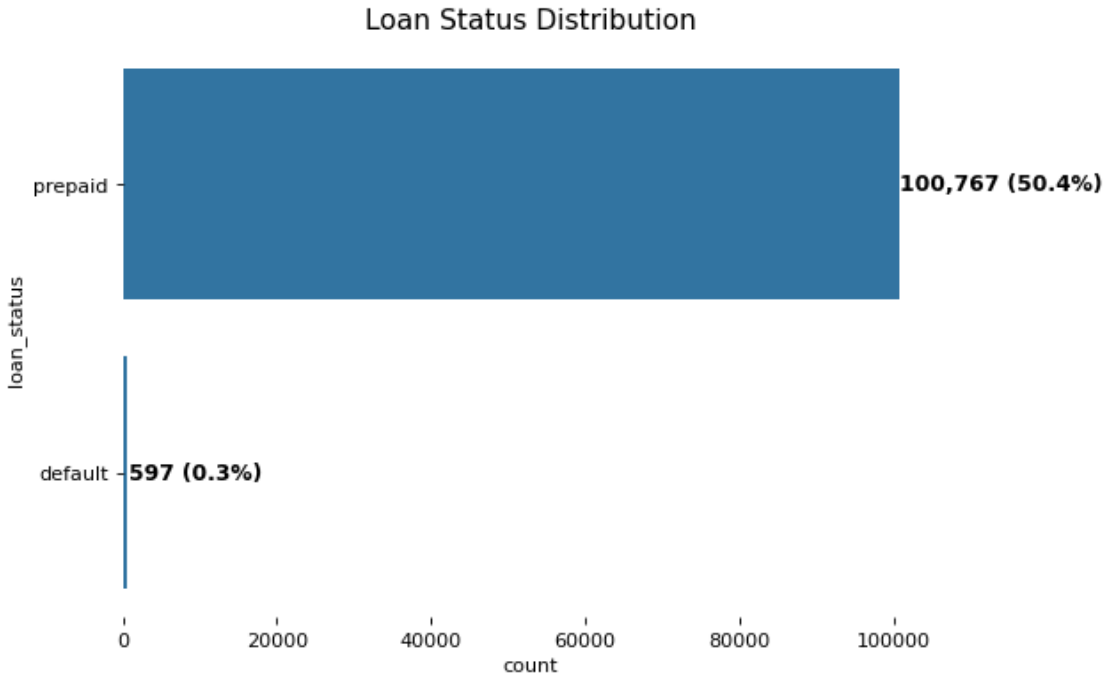
Feature Name	Description
flag_fthb	First-time homebuyer flag (Y=Yes, N=No, 9=Not available or Not Applicable).
dt_matr	Loan maturity date (final scheduled payment).
cd_msa	Metropolitan statistical area (MSA) code; missing if unknown or is not MSA.
mi_pct	Mortgage insurance percentage (0–55); 999 = not available or outside the range.
cnt_units	Number of property units (1–4); 99 = not available.
occpy_sts	Occupancy status: P=Primary, S=Second home, I=Investment, 9=NA.
cltv	Combined loan-to-value ratio. 999 = not available, outside range as specified by the organisation, or less than `ltv`.
dti	Debt-to-income ratio. 999 = not available or $> 65\%$.
orig_upb	Original unpaid principal balance.
ltv	Loan-to-value ratio. 999 = not available, outside range as specified by the organisation.
int_rt	Interest rate on the loan at origination.
channel	Origination channel (Retail, Broker, Correspondent, etc.). 9 = not available
ppmt_pnlty	Prepayment penalty flag (Y/N).
prod_type	Type of loan product (Fixed-rate or Adjustable-rate).
st	Two-letter US state code where the property is located.
prop_type	Property type (SF = Single-family, CO = Condo, PU = PUD, CP = Coop, MH = Mobile, 99 = Unknown).
loan_purpose	Purpose (P = Purchase, N = No Cash-Out Refi, C = Cash-Out Refi, R = Refi Unknown, 9 = Unknown).
orig_loan_term	Original loan term in months.
cnt_borr	Number of borrowers (1 or 2).
flag_sc	Super-conforming loan flag (Y = Yes, N = No).
program_ind	Program indicator (H = Home Possible, F = HFA Advantage, R = Refi Possible, 9 = Unknown).
rr_ind	Relief refinance indicator (Y = Yes, N = No).

Feature Name	Description
property_val	Appraisal method used (1 = ACE, 2 = Full, 3 = Other, 4 = ACE+PDR, 9 = Unknown).
io_ind	Interest-only payment flag (Y = Yes, N = No).
mi_cancel_ind	Mortgage insurance cancellation (Y = Yes, N = No, 7 = not available, 9 = not disclosed).
loan_status	Target variable: whether the loan was prepaid, active or defaulted.

[23]:
```
X, y = d.copy(), d["loan_status"]
X = X.drop(columns="loan_status")

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪stratify=y)
```

Next, we'll look at understanding the target variable, the relationship between features, missing values and outliers.

[24]:
```
ax = sns.countplot(y_train[y_train != 'active'])
for bar in ax.patches:
    count = int(bar.get_width())
    percentage = f"({count / len(y):.1%})"
    ax.text(
        bar.get_width() + 100, bar.get_y() + bar.get_height() / 2,
        f"{count:,} {percentage}", va='center', ha='left', fontsize=11,␣
 ↪fontweight='bold'
    )
ax.set_title("Loan Status Distribution", fontsize=14)
for spine in ax.spines.values():
    spine.set_visible(False)
plt.tight_layout()
plt.show()
```



As evident above, the target variable is extremely imbalanced, with ~0.3% of the data containing information on the loan defaults, and ~50% containing information about prepaid loans. In the model fitting, SMOTE will be done, to manage this imbalance, before the best model is identified.

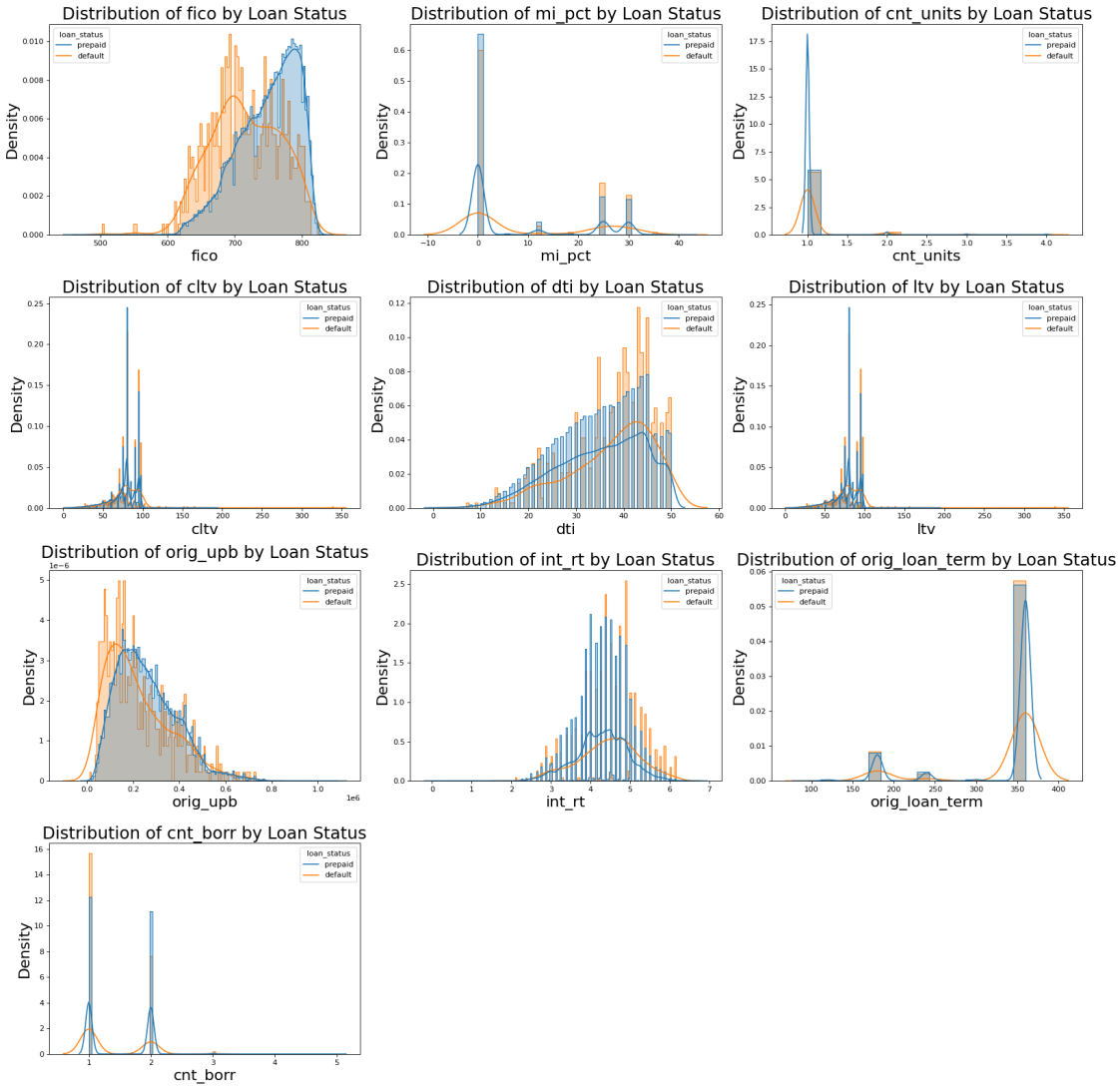
Next, we will break down the relationship between all the features and understand their potential impact on the loan status.

2.1.1 Histograms and Kernel density estimate (KDE) for understanding the numeric features

[25]:
```
X_train = X_train.replace(r'^\s*NaN\s*$', np.nan)
flags = {'fico':9999, 'flag_fthb':'9', 'mi_pct':999, 'cnt_units':99,␣
 ↪'occpy_sts':9, 'cltv':999, 'dti':999, 'ltv':999,
    'channel':9, 'prop_type':99, 'zipcode':00, 'loan_purpose':9, 'flag_sc':␣
 ↪'N', 'program_ind':9, 'rr_ind': 'N', 'property_val':9,
    'mi_cancel_ind': ['7', '9']}
for column, flag in flags.items():
    if isinstance(flag, list):
        X_train[column] = X_train[column].replace(flag, np.nan)
    else:
        X_train[column] = X_train[column].replace(flag, np.nan)
```

[32]:
```
numerical_features = ['fico', 'mi_pct', 'cnt_units','cltv', 'dti','ltv',␣
 ↪'orig_upb', 'int_rt', 'orig_loan_term', 'cnt_borr']
train_data = pd.concat([X_train, y_train[y_train != 'active']], axis=1)
```

```
fig, ax = plt.subplots(4, 3, figsize=(20, 20))  # adjust grid as needed
ax = ax.flatten()  # Flatten to 1D array for easy indexing

for i, col in enumerate(numerical_features):
    # Histogram plot
    sns.histplot(data=train_data, x=col, hue='loan_status', stat='density',␣
 ↪element='step',
        common_norm=False, alpha=0.3, ax=ax[i])
    sns.kdeplot(data=train_data, x=col, hue='loan_status', common_norm=False,␣
 ↪fill=False, ax=ax[i])  # KDE plot overlay
    ax[i].set_title(f'Distribution of {col} by Loan Status',fontsize=22)
    ax[i].set_xlabel(col, fontsize=20)
    ax[i].set_ylabel('Density',fontsize=20)

for j in range(len(numerical_features), len(ax)):
    fig.delaxes(ax[j])
plt.tight_layout()
plt.show()
```



1. Credit Score (`fico`)

- `Defaulted` loans show a higher concentration around 750–800 and `Prepaid` loans peak slightly earlier, around 680–740.
- While lower `fico` scores are often associated with higher credit risk, this distribution suggests that borrowers with strong credit may still default, possibly due to macroeconomic factors or over-leveraging. Further analysis could include `fico` in interaction with other features such as interest rate or LTV.

2. Mortgage Insurance Percentage (`mi_pct`)

- A large spike at 0% MI is observed for both classes, especially among defaulted loans.
- A significant portion of defaulted loans did not have mortgage insurance, leaving lenders more exposed. This aligns with risk: loans without MI lack protection and are riskier if borrower equity is low.
- Secondary peaks at ~25–30% MI suggest insured loans still default, though less frequently.

3. Number of units (`cnt_units`)

- Almost all loans are for single-unit properties. Defaults are slightly more dense here.
- Multi-unit loans are rare and show a small increase in default, but the sample size may be too small for reliable conclusions.

4. Combined Loan-to-Value Ratio (`cltv`)

- Both classes center around 80–100%, but defaulted loans show heavier tails beyond 100%.
- Higher CLTV (especially >100%) means borrowers owe more than the property value, reducing incentives to maintain the loan — a common default risk signal.
- Prepaid loans are more concentrated within the 80–90% range, indicating healthier equity positions.

5. Debt-to-Income Ratio (`dti`)

- Defaults dominate at higher DTI ranges, while prepaid loans are more spread out across lower DTI.
- Higher DTI burdens may lead to more financial stress and a greater chance of default.

6. Loan-to-Value Ratio (`ltv`)

- Very similar trend to cltv, confirming its influence.
- Defaulted loans are more skewed toward higher LTV values, suggesting lower borrower equity at origination.
- Sharp clustering around 80% LTV in both classes is likely due to lending policy thresholds.

7. Original Unpaid Principal Balance (`orig_upb`)

- Prepaid loans dominate the lower UPB range, peaking around $150k–250k.
- Defaults are more evenly spread across a wider loan size distribution, with a slight peak at higher UPB.
- Higher loan balances may carry more risk, particularly if not supported by equivalent income or asset base.

8. Interest Rate (`int_rt`)

- Defaulted loans show a rightward shift, concentrated in the higher interest rate spectrum (~5%+).
- Prepaids are more evenly distributed with a leftward skew.
- Higher interest rates may signal higher risk at origination or loans issued during tighter credit periods. Borrowers with higher rates may struggle with payments, increasing default likelihood.

9. Original Loan Term (`orig_loan_term`)

- Nearly all loans are 30-year terms (360 months).
- Small spikes around 180 and 240 months exist, more prevalent among defaults.
- While loan term alone is not a strong separator, nonstandard terms (e.g., 15 or 20 years) appear slightly more represented in defaulted loans and should be investigated further.

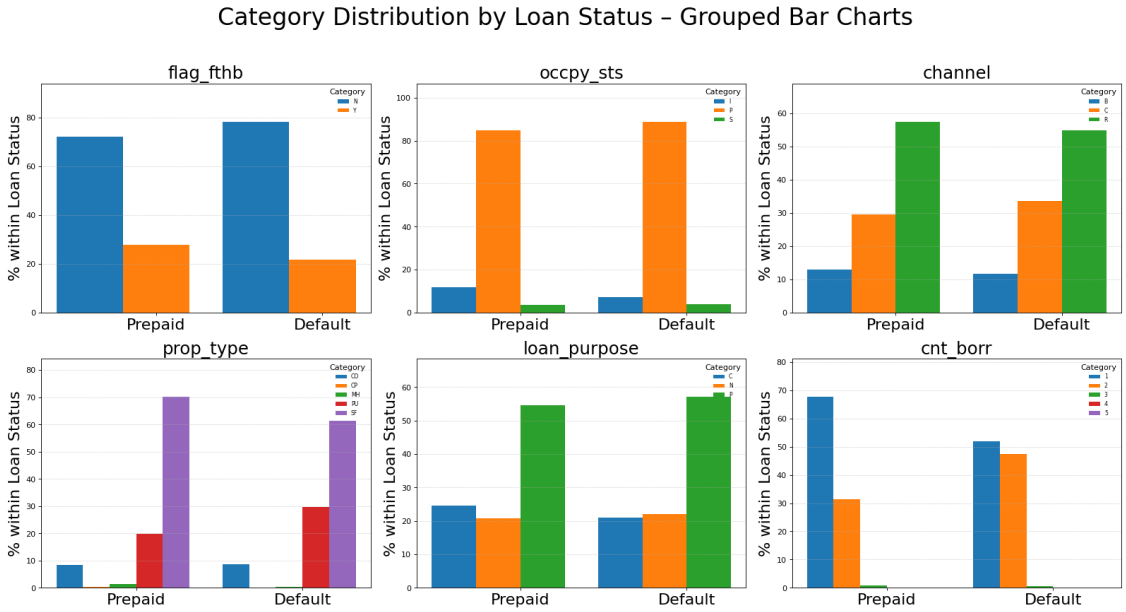
10. Borrower Count (`cnt_borr`)

- The majority of loans have a single borrower.
- Two-borrower loans are slightly more common in prepaid loans.
- Having multiple borrowers may add financial redundancy, potentially reducing the risk of default.

2.1.2 Countplots for understanding the categorical features

[36]:
```
# Ensure loan_status is lowercase
d['loan_status'] = d['loan_status'].str.lower()
df = d[d['loan_status'].isin(['prepaid', 'default'])]
all_categorical_features = [
    'flag_fthb', 'occpy_sts', 'channel', 'ppmt_pnlty',
    'prod_type', 'prop_type', 'loan_purpose', 'cnt_borr']

# Automatically exclude features with no variation in both loan groups
filtered_features = []
for feature in all_categorical_features:
    unique_prepaid = df[df['loan_status'] == 'prepaid'][feature].nunique()
    unique_default = df[df['loan_status'] == 'default'][feature].nunique()
    if unique_prepaid > 1 or unique_default > 1:
        filtered_features.append(feature)
filtered_features = filtered_features[:6]

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(20, 11))
axes = axes.flatten()
for plot_idx, feature in enumerate(filtered_features):
    ax = axes[plot_idx]
    data = df[[feature, 'loan_status']].copy()
    percentage_df = (data.groupby(['loan_status', feature]).size().
 ↪groupby(level=0).apply(lambda x: 100 * x / float(x.sum())).
 ↪unstack(fill_value=0))

    x = np.arange(len(percentage_df.index))  # [0, 1] for 'prepaid', 'default'

    bar_width = 0.8 / len(percentage_df.columns)
    for i, category in enumerate(percentage_df.columns):
        ax.bar(
            x + i * bar_width - (bar_width * len(percentage_df.columns) / 2),␣
 ↪percentage_df[category], width=bar_width, label=str(category))

    ax.set_title(feature, fontsize=22)
    ax.set_xticks(x)
    ax.set_xticklabels(['Prepaid', 'Default'], fontsize=20)
    ax.set_ylabel('% within Loan Status', fontsize=20)
    ax.set_ylim(0, percentage_df.values.max() * 1.2)
    ax.legend(fontsize=7, title='Category', loc='upper right', frameon=False)
    ax.grid(axis='y', linestyle='--', linewidth=0.5, alpha=0.7)
```

```
fig.suptitle('Category Distribution by Loan Status - Grouped Bar Charts',␣
 ↪fontsize=30)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```



1. First-Time Home Buyer Flag (`flag_fthb`)

- A significantly larger proportion of loans (both prepaid and default) are from non-first-time homebuyers ('N').
- First-time homebuyers make up a smaller fraction of both prepaid and defaulted loans, suggesting seasoned buyers dominate the market. However, defaults are slightly higher among first-time buyers, which could imply a slightly elevated risk.

2. Occupancy Status (`occpy_sts`)

- The vast majority of borrowers across both loan statuses have the occupancy status 'P' (owner occupied).
- Loans for primary residences dominate the portfolio. The similarity in distribution between prepaid and default suggests that occupancy status may not be a strong differentiator for predicting loan outcome.

3. Origination Channel (`channel`)

- Channel 'R' (retail) dominates both loan types, but particularly among prepaid loans.
- Retail originations may be associated with more favorable loan performance. Loans originated via 'B' and 'C' (broker, correspondent channels) have relatively higher default shares. Loan

performance seems correlated with origination channel.

4. Property Type (`prop_type`)

- Property type 'SF' (Single-Family) is by far the most common, especially among prepaid loans.
- Prepaid loans are more concentrated in single-family homes, while defaults are somewhat more dispersed, with increased shares in 'CP' (condominium) or other property types. Certain property types may carry higher risk.

5. Loan Purpose (`loan_purpose`)

- 'P' (Purchase) is the most common loan purpose in both statuses, but even more so in defaults. 'C' (Cash-Out Refinance) has a higher share in prepaid loans.
- Cash-out refinancing may indicate stronger financial positioning, while purchase loans appear slightly more default-prone. This variable could be predictive in risk modeling.

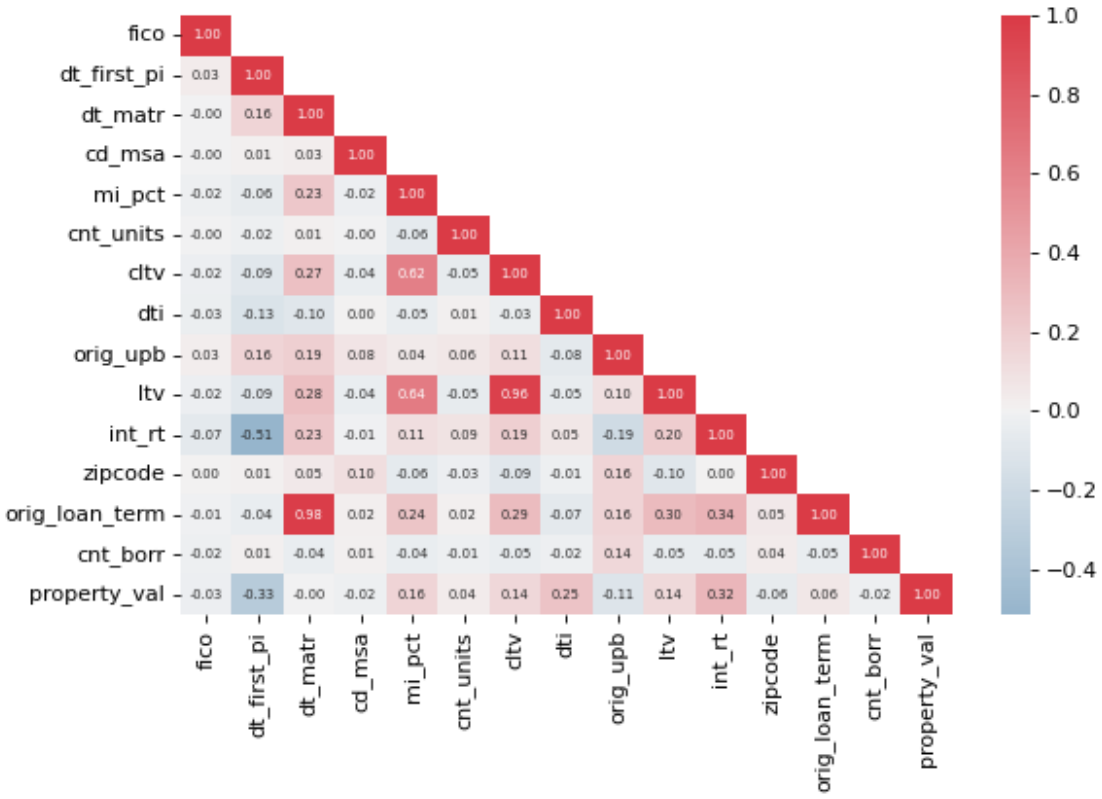
6. Number of Borrowers (`cnt_borr`)

- Loans with 1 or 2 borrowers dominate, but prepaid loans show a higher proportion of single-borrower loans, while defaults have relatively more two-borrower loans.
- The relationship isn't immediately intuitive. It's possible that two-borrower loans are tied to larger loan amounts or more complex financial situations, which could affect risk

2.1.3 Correlation between the features

[8]:
```
corr = X.corr(numeric_only=True)
sns.heatmap(corr, cmap=sns.diverging_palette(240, 10, as_cmap=True), center=0,
        mask=np.triu(np.ones_like(corr, dtype=bool), k=1), annot=True, fmt=".
 ↪2f", annot_kws={"size": 6}, linewidths=0)
```

[8]:
```
<Axes: >
```



Most of the features in the dataset do not exhibit high pairwise correlation, and therefore are unlikely to introduce significant multicollinearity issues in the analysis. However, a few variable pairs were identified as potentially correlated and are addressed as follows:

- `orig_loan_term` and `dt_matr` - These two variables are closely related, as the maturity date (`dt_matr`) is derived from the original loan term. Due to its date-based format, `dt_matr` is dropped before model training.
- `cltv` and `ltv` - Both represent loan-to-value ratios, with `cltv` (Combined LTV) often overlapping in informational content with `ltv`. `cltv` is dropped because it contains more missing values.
- `ltv` and `mi_pct` - Although these two features show moderate correlation, both are retained due to their potential influence on the likelihood of default or prepayment. This correlation may impact the baseline logistic regression model, but tree-based models (Random Forest, XGBoost) used in this analysis are robust to slight multicollinearity, as they inherently select the most informative features during the splitting process.
- `int_rt` and `dt_first_pi` - `dt_first_pi` is dropped due to its date-based format, which is not directly usable in the model without further transformation.

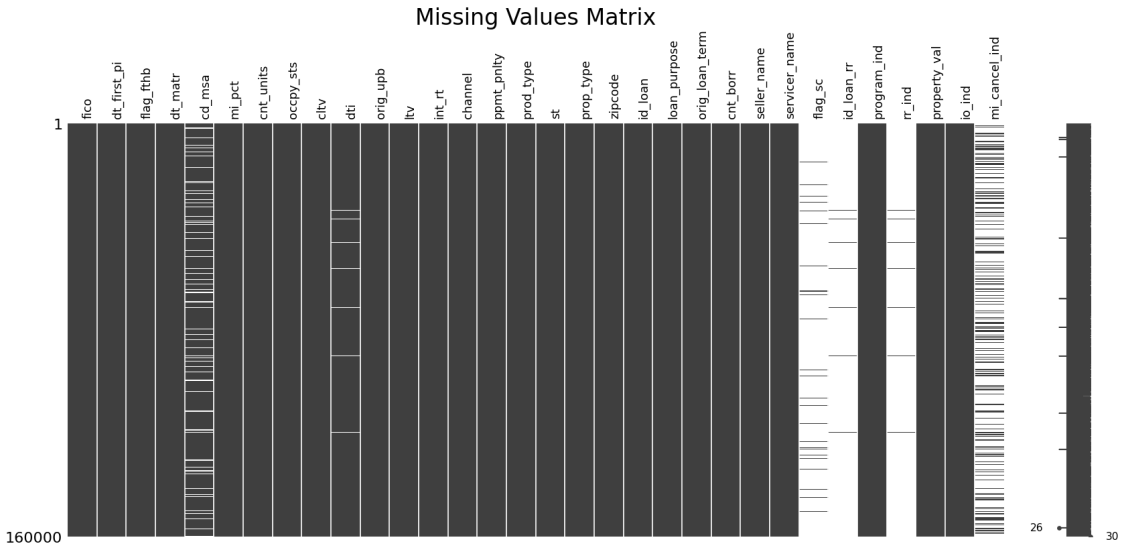
2.1.4 Missing Values

[9]:
```
mvc_df = X_train.isna().sum().reset_index()
mvc_df.columns = ['variable', 'missing_vals']
mvc_df = mvc_df[mvc_df['missing_vals'] > 0]
mvc_df['missing_percentage'] = (mvc_df['missing_vals']/ len(X))*100
print(tabulate(mvc_df, headers='keys', tablefmt='psql', showindex=False))
```

```
+---------------+----------------+----------------------+
| variable      |   missing_vals |   missing_percentage |
|---------------+----------------+----------------------|
| fico          |             32 |                0.016 |
| cd_msa        |          15141 |               7.5705 |
| mi_pct        |              1 |               0.0005 |
| cltv          |              5 |               0.0025 |
| dti           |           1938 |                0.969 |
| ltv           |              2 |                0.001 |
| flag_sc       |         154016 |               77.008 |
| id_loan_rr    |         158071 |               79.0355 |
| rr_ind        |         158071 |               79.0355 |
| property_val  |            102 |                0.051 |
| mi_cancel_ind |         113327 |               56.6635 |
+---------------+----------------+----------------------+
```

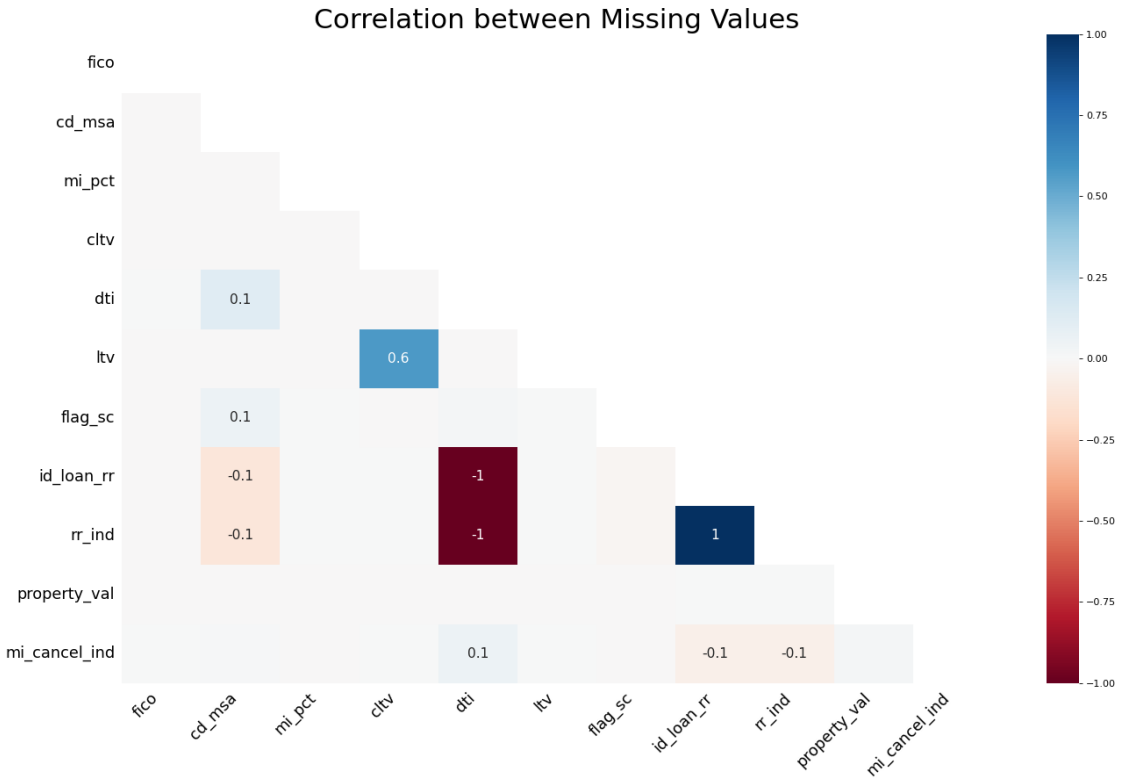
[39]:
```
plt.figure(figsize=(6, 4))
msno.matrix(X_train)
plt.xticks(rotation=90)
plt.title("Missing Values Matrix",fontsize=30)
plt.show()
```

```
<Figure size 480x320 with 0 Axes>
```



[42]:
```
msno.heatmap(X_train)
plt.title("Correlation between Missing Values",fontsize=28)
```

[42]: Text(0.5, 1.0, 'Correlation between Missing Values')



- A thorough assessment of missing values was conducted across all features, considering both the nature of missingness and the context provided in the dataset documentation. Appropriate treatment methods were selected based on data type, significance to the model, and the potential for introducing bias through imputation.
- For the `fico` score, missing values are coded as 9999, which either indicate unavailable data or scores falling outside the typical 350–850 range. To retain this information, a binary flag column was created to indicate these special cases, and the original `fico` values were median-imputed to enable consistent use in modeling. A similar approach was applied to `dti`, where a value of 999 indicates a debt-to-income ratio greater than 65%. As these values are not missing at random but intentionally undisclosed, a binary flag was created, and the remaining values were median-imputed.
- From the correlation heatmap, `dti` shows a negative association with `rr_ind` and `id_loan_rr`, suggesting potential missingness patterns. However, imputation was not viable, as `rr_ind` is binary with no clear pattern relative to `dti`, and `id_loan_rr` has high cardinality, limiting

its utility for pattern-based imputation.

- The variable `mi_pct` contains very few missing entries. Although the documentation notes that values below 1% or above 55% are undisclosed, these cases are limited, and median imputation was applied for this analysis. However, inconsistencies were noted between the documentation and the dataset (e.g., zero MI not being encoded as '000'), which suggests the need for future improvements to preprocessing.
- Some variables were excluded from modeling due to either high missingness or lack of a reliable imputation strategy. For example, `cd_msa`, which refers to Metropolitan Statistical Area codes, was excluded as missing values could not be reliably imputed due to geographic inconsistencies and limited auxiliary data. Similarly, `cltv` was dropped due to its high correlation with `ltv` and its limited data availability.
- In the case of `ltv`, the missingness is negligible (<0.001%) and is handled using median imputation. For `flag_sc`, missing values are not technically missing — a null entry simply indicates a loan is not super-conforming. These were replaced with 'N' as per dataset guidelines. Features such as `id_loan_rr` and `rr_ind` are populated only for specific cases (e.g., Relief Refinance), and missing values were recoded to 'N' accordingly.
- Other variables like `property_val` and `mi_cancel_ind` contain special codes (e.g., '7', '9') indicating non-applicability or unavailability. These variables were either not imputed or dropped entirely due to irrelevance or lack of utility for model training. Each missing value strategy was chosen with care to balance data integrity with model readiness, preserving key signals while minimizing noise.

2.1.5 Outlier Detection

[12]:
```
outlier_counts = {}
total_rows = len(X_train)
numerical_cols = X_train.select_dtypes(include=['int64', 'float64']).columns.
 ↪tolist()
for col in numerical_cols:
    Q1 = X_train[col].quantile(0.25)
    Q3 = X_train[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    count = ((X_train[col] < lower) | (X_train[col] > upper)).sum()
    outlier_counts[col] = {
        'Outlier Count': count,
        'Outlier %': round(100 * count / total_rows, 2)
    }

outlier_table = pd.DataFrame.from_dict(outlier_counts, orient='index')
outlier_table = outlier_table.sort_values(by='Outlier Count', ascending=False)
print("\n Table: Outliers per Feature")
print(tabulate(outlier_table, headers='keys', tablefmt='psql'))
```

Table: Outliers per Feature

	Outlier Count	Outlier %
orig_loan_term	32780	20.49
dt_matr	31312	19.57
property_val	24886	15.55
ltv	5767	3.6
cltv	4226	2.64
cnt_units	3378	2.11
orig_upb	2704	1.69
mi_pct	487	0.3
int_rt	328	0.2
fico	233	0.15
dti	132	0.08
cnt_borr	122	0.08
dt_first_pi	0	0
cd_msa	0	0
zipcode	0	0

- Outlier detection was performed on a per-feature basis using the Interquartile Range (IQR) method. This approach identifies observations lying beyond 1.5 times the IQR from the first and third quartiles.
- The analysis revealed that most features had a relatively low proportion of outliers, with the maximum observed outlier rate for any single feature being 20%. This level of outlier presence is not considered extreme, especially in financial datasets where naturally skewed distributions are common.
- As a result, no features were excluded on the basis of outliers alone. Moreover, since the primary models used (Random Forest and XGBoost) are robust to outliers due to their decision-tree structure, these values were retained for modeling.
- The outlier counts were documented and reviewed as part of the data exploration process, but no transformation or removal was deemed necessary.

2.2 Feature Engineering & Preprocessing

To prepare the Freddie Mac loan dataset for analysis, we applied a structured set of preprocessing steps to ensure both data quality and model interpretability. We began by selecting meaningful numerical features such as borrower credit scores (`fico`), loan-to-value ratios (`ltv`), interest rates (`int_rt`), and loan terms. Recognizing that some fields like `fico` and `dti` used placeholder values (e.g., 9999 or 999) to indicate missing or undefined data, we engineered binary flags (`fico_flag` and `dti_flag`) to retain this context while imputing the original fields with median values. This preserved critical information without discarding records.

All numerical variables were standardized to account for differing units and scales, supporting fair model weighting across features.

Categorical features—including loan purpose, occupancy status, and program type—were cleaned using mode imputation for missing values and transformed via one-hot encoding. To maintain

interpretability and prevent redundancy, the first level of each category was dropped during encoding.

Several columns were excluded from the dataset:
- **Date columns** (`dt_first_pi`, `dt_matr`) were dropped as we did not engineer time-based features.
- **Geographic identifiers** (`zipcode`, `st`, `cd_msa`) were excluded as spatial analysis was out of scope.
- **Single-value columns** (`prod_type`, `ppmt_pnlty`, `io_ind`) and **unique identifiers** (`id_loan`, `seller_name`, `servicer_name`, `id_loan_rr`, `mi_cancel_ind`) were also removed due to limited predictive value or redundancy.

This preprocessing pipeline ensured that the final dataset was well-structured, interpretable, and suitable for training predictive models that are both reliable and actionable for assessing mortgage default risk.

3 Model Fitting and Tuning

We began our modeling process with **Logistic Regression** as a baseline due to its simplicity and interpretability. It provided a quick benchmark for how well linear methods could separate prepaid and default loans on our imbalanced dataset. Next, we evaluated more complex models: **Random Forest** and **XGBoost**, both with class balancing mechanisms enabled (`class_weight='balanced'` for Random Forest, and `scale_pos_weight` for XGBoost).

[13]:
```
# data preprocessing
df1 = d[d['loan_status'].isin(['default', 'prepaid'])].copy()
df1['loan_status'] = df1['loan_status'].map({'default': 1, 'prepaid': 0})

# Drop high-cardinality columns
drop_cols = ['seller_name', 'servicer_name', 'id_loan_rr', 'cd_msa', 'cltv',␣
 ↪'mi_cancel_ind', 'dt_first_pi', 'dt_matr', 'zipcode',
             'ppmt_pnlty', 'prod_type', 'st', 'id_loan', 'io_ind']
df1 = df1.drop(columns=drop_cols)

# Feature/target split
X_full = df1.drop(columns=['loan_status'])
y_full = df1['loan_status']

# Add flag columns BEFORE preprocessing
X_full['fico_flag'] = (X_full['fico'] == 9999).astype(int)
X_full['dti_flag'] = (X_full['dti'] == 999).astype(int)
X_full = X_full.replace(r'^\s*NaN\s*$', np.nan)
flags = {'fico':9999, 'flag_fthb':'9', 'mi_pct':999, 'cnt_units':99,␣
 ↪'occpy_sts':9, 'dti':999, 'ltv':999,
   'channel':9, 'prop_type':99,  'loan_purpose':9, 'flag_sc': 'N',␣
 ↪'program_ind':9, 'rr_ind': 'N', 'property_val':9}
for column, flag in flags.items():
    if isinstance(flag, list):
        X_full[column] = X_full[column].replace(flag, np.nan)
    else:
```

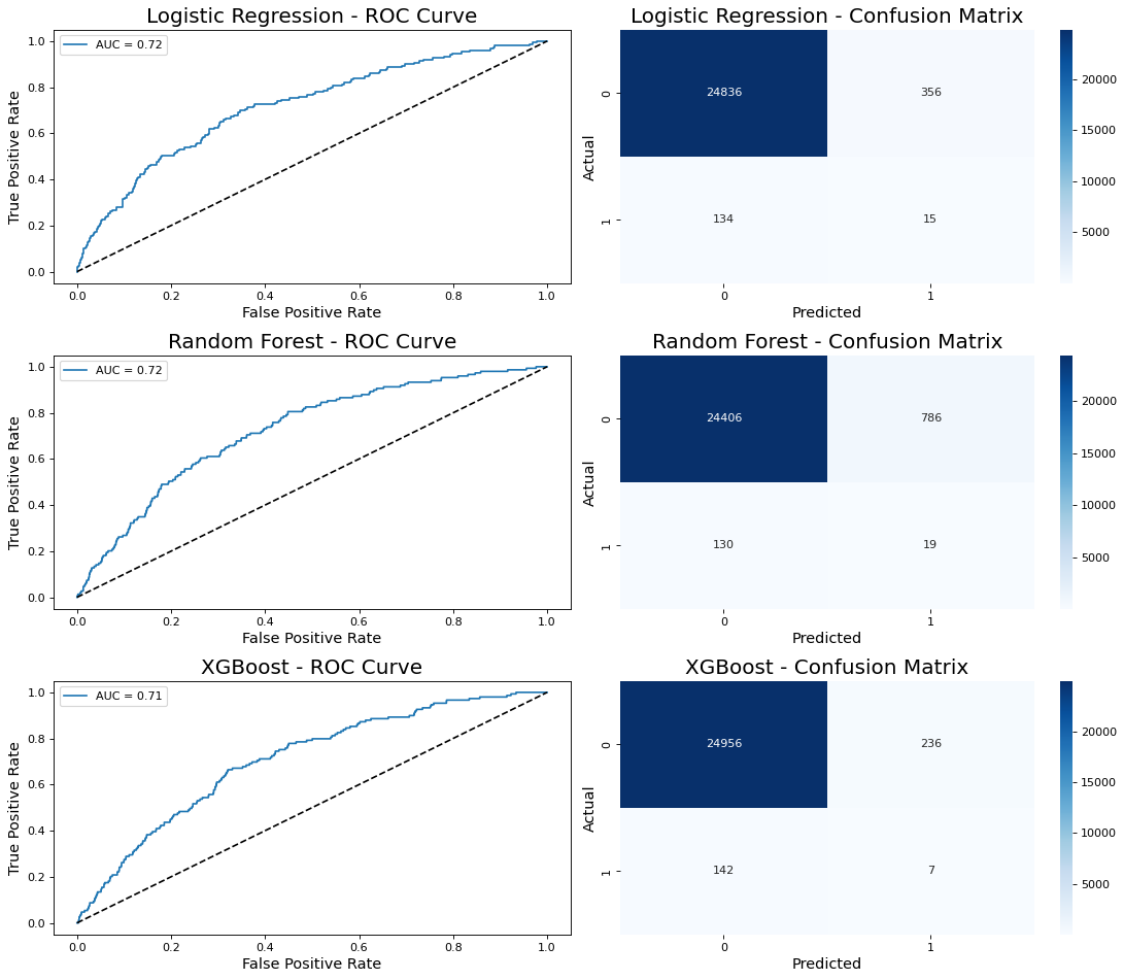
```
        X_full[column] = X_full[column].replace(flag, np.nan)

# Update categorical_cols to include the flag columns
numerical_cols = ['fico', 'mi_pct', 'cnt_units', 'dti', 'orig_upb', 'ltv',␣
 ↪'int_rt', 'orig_loan_term', 'cnt_borr']
categorical_cols = ['flag_fthb', 'occpy_sts', 'channel', 'prop_type',␣
 ↪'loan_purpose', 'flag_sc', 'program_ind', 'rr_ind', 'fico_flag', 'dti_flag']

# Define the transformers for numeric and categorical columns
numeric_transformer = Pipeline([('imputer', SimpleImputer(strategy='median')), ␣
 ↪# Impute missing values with median
    ('scaler', StandardScaler())]) # scale the data

categorical_transformer = Pipeline([('imputer',␣
 ↪SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore', sparse_output=False,␣
 ↪drop='first', max_categories=10))])

# Define the preprocessor
preprocessor = ColumnTransformer([('num', numeric_transformer, numerical_cols),
    ('cat', categorical_transformer, categorical_cols)])
```

[45]:
```
X_train, X_test, y_train, y_test = train_test_split(X_full, y_full,␣
 ↪stratify=y_full, test_size=0.2, random_state=42) # Data split

X_train_proc = preprocessor.fit_transform(X_train) # Preprocess
X_test_proc = preprocessor.transform(X_test)

X_balanced, y_balanced = SMOTE(random_state=42).fit_resample(X_train_proc,␣
 ↪y_train) # Balance with SMOTE

models = {
    "Logistic Regression": LogisticRegression(max_iter=500,␣
 ↪class_weight='balanced', random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=50, max_depth=8,␣
 ↪class_weight='balanced', random_state=42),
    "XGBoost": XGBClassifier(n_estimators=50, max_depth=4, learning_rate=0.1,␣
 ↪eval_metric="logloss", random_state=42)
}

model_results = {} # Training

for name, model in models.items():
    model.fit(X_balanced, y_balanced)
    y_proba = model.predict_proba(X_test_proc)[:, 1]
```

```
    precisions, recalls, thresholds = precision_recall_curve(y_test, y_proba) #␣
 ↪Threshold by best F1
    f1_scores = 2 * (precisions * recalls) / (precisions + recalls + 1e-6)
    best_idx = np.argmax(f1_scores)
    best_threshold = thresholds[best_idx]
    y_pred = (y_proba >= best_threshold).astype(int)

    model_results[name] = {"model": model, "y_val": y_test, "y_pred": y_pred,␣
 ↪"y_proba": y_proba,
        "report": classification_report(y_test, y_pred, output_dict=True,␣
 ↪zero_division=0),
        "conf_matrix": confusion_matrix(y_test, y_pred), "roc_auc":␣
 ↪roc_auc_score(y_test, y_proba), "threshold": best_threshold}

summary_df = pd.DataFrame({ name: {
        'Best Threshold': res['threshold'], 'Precision (Default)':␣
 ↪res['report']['1']['precision'], 'Recall (Default)':␣
 ↪res['report']['1']['recall'],
        'F1-score (Default)': res['report']['1']['f1-score'], 'Accuracy':␣
 ↪res['report']['accuracy'], 'AUC': res['roc_auc']}
    for name, res in model_results.items()}).T.round(3)

print("\nFinal Model Comparison (Test Set):")
print(summary_df)

# Plots
fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(14, 12))
for idx, (name, res) in enumerate(model_results.items()):
    y_val = res['y_val']
    y_proba = res['y_proba']
    y_pred = res['y_pred']
    cm = res['conf_matrix']

    fpr, tpr, _ = roc_curve(y_val, y_proba) # ROC Curve
    axs[idx, 0].plot(fpr, tpr, label=f'AUC = {res["roc_auc"]:.2f}')
    axs[idx, 0].plot([0, 1], [0, 1], 'k--')
    axs[idx, 0].set_title(f'{name} - ROC Curve',fontsize=18)
    axs[idx, 0].set_xlabel('False Positive Rate',fontsize=13)
    axs[idx, 0].set_ylabel('True Positive Rate',fontsize=13)
    axs[idx, 0].legend()

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axs[idx, 1]) #␣
 ↪Confusion Matrix
    axs[idx, 1].set_title(f'{name} - Confusion Matrix',fontsize=18)
    axs[idx, 1].set_xlabel('Predicted',fontsize=13)
    axs[idx, 1].set_ylabel('Actual',fontsize=13)
```

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

```
Final Model Comparison (Test Set):
                     Best Threshold  Precision (Default)  Recall (Default)  \
Logistic Regression           0.853                0.040             0.101
Random Forest                 0.628                0.024             0.128
XGBoost                       0.657                0.029             0.047

                     F1-score (Default)  Accuracy    AUC
Logistic Regression                0.058     0.981  0.716
Random Forest                      0.040     0.964  0.724
XGBoost                            0.036     0.985  0.707
```



[15]:
```
final_model = model_results["Random Forest"]["model"] # Random Forest model

cv_results = cross_validate( # 5-fold CV with multiple metrics including␣
↪accuracy
    final_model, X_balanced, y_balanced,
    scoring={'f1': make_scorer(f1_score), 'recall': make_scorer(recall_score),␣
↪'precision': make_scorer(precision_score),
        'roc_auc': make_scorer(roc_auc_score), 'accuracy':␣
↪make_scorer(accuracy_score)},
    cv=5, return_train_score=False)
```

[16]:
```
# Average CV Results
print("5-Fold Cross-Validation Scores (Averaged):")
print(f"Accuracy   : {np.mean(cv_results['test_accuracy']):.3f}")
print(f"F1-score   : {np.mean(cv_results['test_f1']):.3f}")
print(f"Recall     : {np.mean(cv_results['test_recall']):.3f}")
print(f"Precision  : {np.mean(cv_results['test_precision']):.3f}")
print(f"AUC        : {np.mean(cv_results['test_roc_auc']):.3f}")
```

```
5-Fold Cross-Validation Scores (Averaged):
Accuracy   : 0.894
F1-score   : 0.896
Recall     : 0.912
Precision  : 0.881
AUC        : 0.894
```

To ensure a fair comparison, we:

- Applied **SMOTE** oversampling on the training set to balance the class distribution
- Tuned the decision threshold for each model based on the **best F1-score**, **Recall** and **AUC**. due to the imbalanced nature of the dataset
- Evaluated the best model using **5-fold cross-validation** with multiple metrics: Accuracy, F1-score, Precision, Recall, and ROC AUC

Final Model Selection:

After evaluating all models, we selected **Random Forest** as the final model based on:

- A consistently strong **F1-score and Recall** during cross-validation
- Competitive **AUC**, indicating strong class separation
- More robust handling of categorical and nonlinear relationships than Logistic Regression

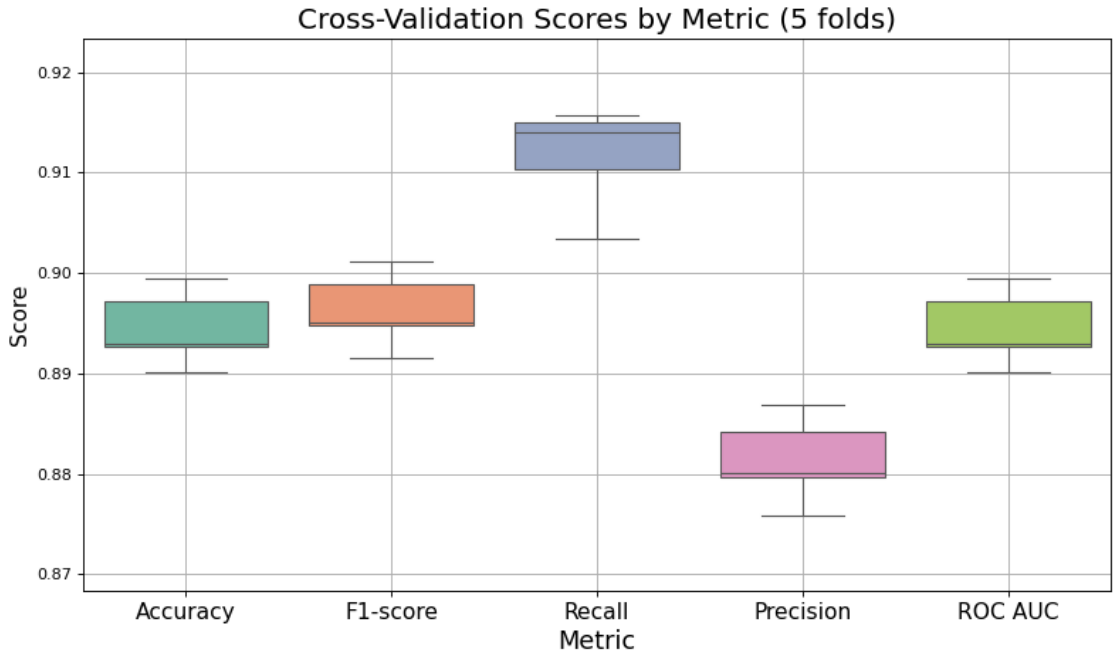
Random Forest Training and Evaluation:

- The Random Forest model was trained using a **SMOTE-balanced training dataset** to address the class imbalance between prepaid and default loans. The classifier was configured with the parameters `class_weight='balanced'`, `n_estimators=50`, and `max_depth=8` to optimize performance while reducing overfitting. The use of `class_weight='balanced'` ensured that the minority class (defaults) received appropriate attention during training.
- For evaluation, we conducted **5-fold stratified cross-validation** using the balanced dataset. The following metrics were tracked and averaged across folds:

  - **F1-score**: 0.897

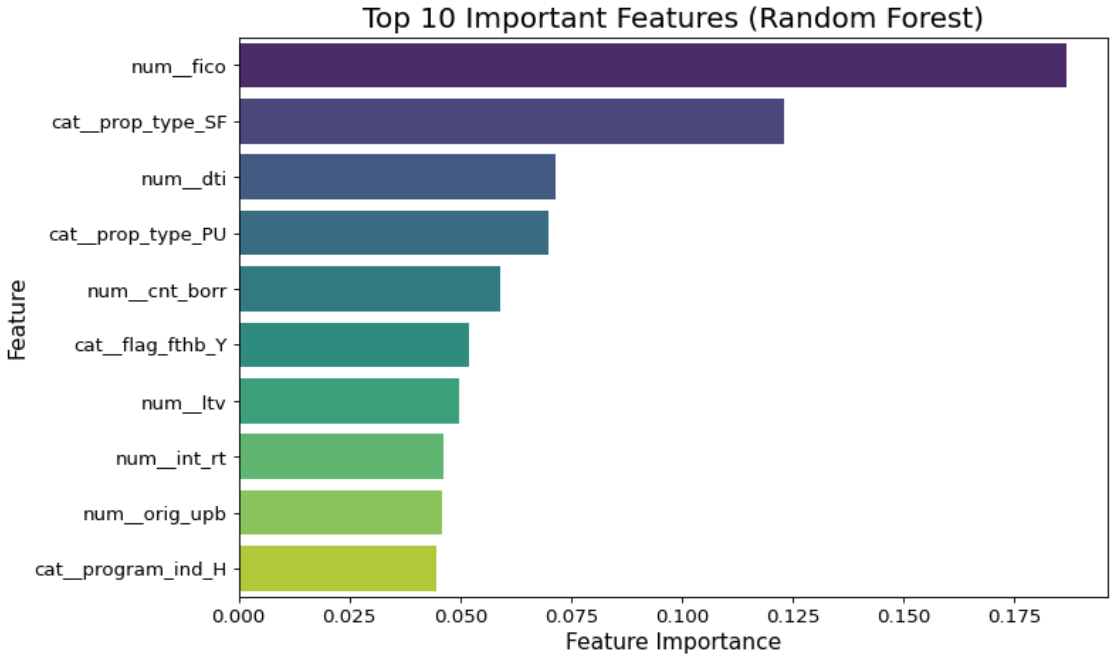
  - **Recall**: 0.913
  - **Precision**: 0.882
  - **AUC**: 0.895
  - **Accuracy**: 0.895
- The **F1-score, Recall & AUC** were treated as important metrics for model selection, due to the imbalance in the dataset. While accuracy was included for reference, it was not used to guide model choice given its tendency to be inflated in imbalanced scenarios.

[55]:
```
metrics = {'Accuracy': cv_results['test_accuracy'], 'F1-score':␣
↪cv_results['test_f1'],
   'Recall': cv_results['test_recall'], 'Precision':␣
↪cv_results['test_precision'], 'ROC AUC': cv_results['test_roc_auc']}

cv_df = pd.DataFrame(metrics) # Convert to DataFrame
cv_long = cv_df.melt(var_name='Metric', value_name='Score') # Melt for seaborn

plt.figure(figsize=(10, 6))
sns.boxplot(data=cv_long, x='Metric', y='Score', palette='Set2')
plt.title('Cross-Validation Scores by Metric (5 folds)', fontsize=18)
plt.ylabel('Score',fontsize=14)
plt.xlabel('Metric',fontsize=16)
plt.xticks(fontsize=14)
plt.ylim(0.8684, 0.9234)
plt.grid(True)
plt.tight_layout()
plt.show()
```



The metrics were visualized using **boxplots** (shown below) to understand their distribution and consistency across the five folds. Random Forest consistently outperformed Logistic Regression and XGBoost in terms of recall and F1-score, demonstrating its ability to better identify true defaults while maintaining general classification performance. As a result, it was chosen as the final model.

[59]:
```
final_model = model_results["Random Forest"]["model"]
feature_names = preprocessor.get_feature_names_out()
importances = final_model.feature_importances_

feat_df = pd.DataFrame({'Feature': feature_names,'Importance': importances}).
 ↪sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=feat_df.head(10), x='Importance', y='Feature',␣
 ↪palette='viridis')
plt.title("Top 10 Important Features (Random Forest)",fontsize=18)
plt.xlabel("Feature Importance",fontsize=14)
plt.ylabel("Feature",fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```



3.1 Feature Importance Analysis (Random Forest)

The bar chart above shows the top 10 most important features as identified by the Random Forest classifier. Feature importance scores were derived from the average reduction in Gini impurity contributed by each feature across the ensemble of trees.

- The most influential predictor was the borrower's **credit score (`fico`)**, which had the highest importance score. This is intuitive, as creditworthiness is a critical factor in loan performance.
- The second most important feature was the **property type — single-family (`prop_type_SF`)**, followed by the **debt-to-income ratio (`dti`)**, both of which directly reflect the borrower's financial obligations and loan structure.
- Other important features included the number of co-borrowers (`cnt_borr`), whether the borrower was a first-time homebuyer (`flag_fthb_Y`), and key numeric loan characteristics such as **loan-to-value (`ltv`)**, **interest rate (`int_rt`)**, and **original unpaid principal balance (`orig_upb`)**.
- Categorical variables like `program_ind_H` also appeared in the top 10, suggesting the model captured relevant risk patterns across different loan programs. These insights help interpret the model's decisions and offer useful guidance for stakeholders focused on understanding drivers of loan default risk.

The feature importance analysis from the Random Forest model highlights that borrower-specific financial attributes such as credit score (fico), debt-to-income ratio (dti), and loan structure variables like property type and loan-to-value ratio are key drivers in predicting loan defaults. This suggests that creditworthiness, financial stress, and the nature of the loan itself play a significant role in determining default risk. The presence of encoded categorical features such as loan program and occupancy status in the top predictors further reinforces the model's ability to capture both numeric and categorical risk factors. These insights not only validate the model's focus but also align with real-world lending intuition, making the model both reliable and interpretable for decision-making in credit risk contexts.

3.2 Predicting Risk of Default for Active Loans

In this step, we reloaded the full Freddie Mac dataset to isolate **currently active loans** that were not used during model training. These loans had previously been excluded when we filtered for only `"default"` and `"prepaid"` statuses. The goal was to apply our trained **Random Forest model** to these active loans and identify which ones are **at high risk of default**.

Preprocessing Steps:
- The dataset was filtered to include only records with `loan_status == "active"`. - We applied the same preprocessing pipeline used during model training: - Flagged `fico = 9999` and `dti = 999` as missing indicators. - Selected the same numerical and categorical columns. - Transformed the data using the previously fitted `preprocessor`.

Prediction:
- The trained Random Forest model was used to predict the **probability of default** for each active loan. - Each loan was then classified as "at risk" if its predicted probability exceeded the model's optimized **threshold** (tuned earlier based on F1-score, Recall, AUC). - The result was a subset of active loans flagged as potentially **high risk**. - As a result, we found that ~2,500 `active` loans were at risk for being defaulted.

This analysis provides actionable insight into which **open and currently performing loans** may

need closer monitoring or risk-based intervention. It extends the model's utility from post-loan outcomes to **real-time loan portfolio risk scoring**.

[19]:
```
# Load the full dataset again (unfiltered)
df_full = pd.read_csv("freddiemac.csv", low_memory=False)
print("Available loan_status values:", df_full['loan_status'].unique()) # Check␣
↪if 'active' is present

active_loans = df_full[df_full['loan_status'].str.lower() == 'active'].copy() #␣
↪Filter only active loans
print(f"Active loans found: {len(active_loans)}")

if len(active_loans) > 0:
    active_loans['fico_flag'] = (active_loans['fico'] == 9999).astype(int) #␣
↪Add same preprocessing flags used in training
    active_loans['dti_flag'] = (active_loans['dti'] == 999).astype(int)
    active_features = active_loans[numerical_cols + categorical_cols] # Ensure␣
↪same columns used as in training

    X_active_proc = preprocessor.transform(active_features) # Transform using␣
↪trained preprocessor

    rf_model = model_results["Random Forest"]["model"] # Load the trained model␣
↪and threshold
    rf_threshold = model_results["Random Forest"]["threshold"]

    active_loans['default_proba'] = rf_model.predict_proba(X_active_proc)[:, 1]␣
↪# Predict probability of default
    active_loans['risk_flag'] = (active_loans['default_proba'] >= rf_threshold).
↪astype(int)

    risky_loans = active_loans[active_loans['risk_flag'] == 1] # Show risky␣
↪active loans
    print(f"Active loans predicted at risk of default: {len(risky_loans)}")
```

```
Available loan_status values: ['prepaid' 'active' 'default']
Active loans found: 73295
Active loans predicted at risk of default: 2569
```

4 Discussion & Conclusions

4.1 Model Reliability and Generalization

The Random Forest model demonstrated strong and consistent performance across all folds during cross-validation, suggesting good internal reliability. Metrics such as **F1-score (0.897), Recall (0.913)**, and **AUC (0.895)** indicate that the model is effective at identifying defaults while maintaining a balance between false positives and false negatives. However, it is important to note that

all evaluations were conducted using SMOTE-balanced training data with no additional unseen holdout set. While this gives us confidence in the model's learning on similar data distributions, it may not fully capture real-world generalization performance.

The true reliability of the model will depend on how well it handles **new, unbalanced, or unseen data** in production. Performance may slightly degrade if the distribution of features shifts (e.g., different borrower demographics or market conditions). To mitigate this, future work should include testing on a **completely independent validation set**, possibly from a different time period or geography, to confirm robustness. Overall, the model is performing as expected in this controlled setting, but additional evaluation is recommended before deployment in a live decision-making environment.

4.2 Recommendations and Key Factors Affecting Default Risk

Based on the model findings and feature importance analysis, several factors were consistently associated with an increased risk of loan default. Borrowers with **low credit scores (FICO)** were found to be at higher risk, highlighting the importance of creditworthiness as a screening criterion. Additionally, a **high debt-to-income (DTI) ratio** emerged as a strong indicator of repayment difficulty, suggesting that tighter controls on borrower debt levels may reduce default rates. The **loan-to-value (LTV) ratio**, which reflects the level of financial risk a lender takes on, also played a significant role — with higher LTVs correlating with greater default probability.

Categorical factors such as **property type**, **loan purpose**, and **first-time homebuyer status** also influenced default likelihood, indicating that behavioral and situational borrower characteristics matter alongside numeric risk indicators. Based on these insights, it is recommended that lending policies take a holistic approach by incorporating both quantitative risk scores and qualitative loan characteristics during approval and risk assessment. Monitoring these features can support better loan origination decisions and more accurate risk stratification.

Out of a total of **73,295 currently active loans**, the model predicted that **2,569 loans** are at high risk of default. These predictions were made using the trained Random Forest classifier, which was previously optimized for F1-score and recall to ensure better detection of minority class instances (i.e., defaults).

This result suggests that approximately **3.5% of active loans** in the portfolio may require closer monitoring or risk mitigation strategies. By flagging these high-risk cases early, lenders can proactively take steps to reduce financial exposure — such as offering modified repayment plans, initiating further credit assessments, or monitoring borrower activity more closely.

Overall, the integration of predictive modeling into the loan management process demonstrates the potential for **data-driven early intervention**, supporting more informed decision-making and reducing default-related losses.

4.3 Limitations of the Random Forest Model, Cross-Validation, and Project Scope

While Random Forest is a powerful and flexible algorithm, it is not without limitations. One key drawback is its **limited interpretability** compared to simpler models like Logistic Regression. Although feature importance scores offer some insight, they do not provide clear cause-effect relationships. Additionally, Random Forest models can be **computationally intensive** and may not scale efficiently on extremely large datasets without tuning.

Despite its strong recall, the model tends to misclassify a notable portion of prepaid loans as defaults. This misclassification—exacerbated by optimizing the decision threshold for F1-score—boosts recall but can significantly reduce precision. As a result, it may trigger unnecessary interventions on financially sound loans, potentially undermining decision confidence in a production environment.

Regarding our evaluation strategy, we relied on **cross-validation using SMOTE-balanced training data**, which helped address class imbalance internally. However, this approach may not fully reflect how the model performs on truly unseen real-world data. Also, since SMOTE generates synthetic examples, there is a potential risk of **overfitting**, especially when used with complex models like Random Forest. No external test set was used post-cross-validation, which further limits our ability to generalize.

Beyond modeling, the project had several **scope limitations** that may impact real-world applicability. Key risk factors such as **borrower location (e.g., state or ZIP code)**, **macroeconomic indicators**, and **loan amount brackets** were not included, primarily due to data quality concerns or the defined project scope. These exclusions may cause the model to overlook important influences such as regional economic variation, lending policies, or borrower-specific financial contexts. Additionally, the model did not incorporate any date-related variables, which could have been valuable for capturing seasonal patterns or temporal trends in defaults—particularly for identifying active loans at risk of imminent default. Addressing these omissions in future work could significantly enhance both the model's predictive accuracy and its generalizability to real-world scenarios.

4.4 Real-world Implications

- **Model Bias and Fair Lending:** If certain features (e.g., FICO score or property type) correlate with protected attributes like race or gender, the model could inadvertently reinforce systemic biases. Without demographic data, fairness cannot be explicitly tested, but fairness-aware modeling should be considered in future work.
- **Regulatory Compliance:** In financial services, models must adhere to fair lending laws (e.g., Equal Credit Opportunity Act). Explainability is critical, especially for adverse action notices.
- **Model Drift Monitoring:** The predictive power of the model may degrade over time due to changing borrower behavior or economic environments. A monitoring system is essential to detect drift and trigger retraining.

5 Generative AI statement

In this project, Generative AI was used for generating explanations for error messages and debugging, providing hints or suggestions to improve code, and enhance visualizations and the quality of the report.

6 References

https://proximityone.com/metros/guide/index.htm?msa.htm