project2

April 11, 2025

1 Machine Learning in Python - Project 2

Due Friday, Apr 11th by 4 pm.

Group member: Dejie Li , Xingshen Chen

1.1 Setup

Install any packages here, define any functions if neeed, and load data

[3]:

```
# Add any additional libraries or submodules below

# Data libraries
import pandas as pd
import numpy as np
import math

# remove warnings
import warnings
warnings.filterwarnings('ignore')

# data cleaning
import missingno as msno

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

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

# sklearn modules
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
```

```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.under_sampling import RandomUnderSampler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import make_scorer, recall_score
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, confusion_matrix
```

[4]:

```python
# helper function
def classification_model_fit(model, X_test, y_test, model_name="Model"):
    """
    Evaluate the performance of a binary classification model and plot the␣
 ↪confusion matrix.

    Parameters:
    model      : Trained sklearn classification model (e.g., LogisticRegression)
    X_test     : Feature matrix of the test set
    y_test     : True labels of the test set
    model_name : Optional name to display in plot titles
    """

    # Get predicted class labels
    y_pred = model.predict(X_test)

    # Get predicted probabilities (used for ROC AUC)
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)[:, 1]
    else:
        y_prob = model.decision_function(X_test)

    # Compute evaluation metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_prob)

    # Print metrics
    print(f" Evaluation Metrics ({model_name})")
    print("-" * 30)
    print(f"Accuracy       : {acc:.4f}")
    print(f"Precision      : {prec:.4f}")
    print(f"Recall         : {rec:.4f}")
    print(f"F1 Score       : {f1:.4f}")
    print(f"ROC AUC Score  : {auc:.4f}")
```

```
# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)
labels = ['Prepaid (0)', 'Default (1)']

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted', fontsize=12)
plt.ylabel('Actual', fontsize=12)
plt.title(f'{model_name} - Confusion Matrix', fontsize=14)
plt.show()

# Optionally return the results as a dictionary
return {
    "accuracy": acc,
    "precision": prec,
    "recall": rec,
    "f1": f1,
    "roc_auc": auc,
    "confusion_matrix": cm
}
```

1.2 Introduction

1.2.1 Research Background and Significance

Credit default risk assessment is a crucial task in the credit industry. An effective default prediction model can help financial institutions identify high-risk borrowers, reduce losses from non-performing loans, and enhance the fairness and accuracy of credit decision-making. With the growing availability of data, machine learning-based risk modeling has become increasingly popular. This project aims to develop an interpretable classification model to predict loan defaults and identify key factors contributing to the likelihood of default, thereby supporting data-driven decisions in credit risk management.

1.2.2 Data Source

The dataset used in this study comes from the Freddie Mac Single-Family Loan-Level Dataset. We use a simplified version focusing on the years 2017 to 2020. It includes detailed loan-level information such as borrower credit scores, loan terms, property characteristics, geographic location, loan-to-value ratios, and repayment statuses. The target variable, loan_status, contains three possible values: default, prepaid, and active. Special codes such as 999 and 9 are used to indicate missing values in some features, and we apply data cleaning and imputation strategies during preprocessing to handle them appropriately.

1.2.3 Research Objective

The primary goal of this project is to build a classification model that accurately predicts the probability of loan default. We first extract all loans labeled as either default or prepaid and split them into training and test sets for model development and evaluation. Once validated, the trained model is then applied to the remaining active loans to predict potential defaults. Additionally, we aim to identify and interpret the most important predictors of loan default to support more informed credit risk assessments and policy design.

1.2.4 Methodology

We applied multiple classification algorithms including Logistic Regression, Lasso, Random Forest, and XGBoost to predict loan default. To address the severe class imbalance, we experimented with resampling techniques such as SMOTE and Random Undersampling. After evaluating model performance across precision, recall, accuracy, and AUC, we selected the Random Forest with undersampling as our final model. Preprocessing steps such as imputation and one-hot encoding were applied to ensure model compatibility and interpretability.

1.2.5 Conclusion

Our final model effectively identifies potential defaults with strong recall performance and reasonable interpretability. By analyzing feature importance from the Random Forest model, we identified key risk indicators such as low FICO scores and high loan amounts. While the model performs well, its limitation lies in the absence of a formal cost function to quantify the impact of different types of errors. Future work could focus on incorporating economic cost estimations to better guide decision-making.

1.3 Exploratory Data Analysis and Feature Engineering

1.3.1 Dataset Feature Desctiptions

The dataset used in this project originates from a mortgage loan performance dataset released by Fannie Mae, a government-sponsored enterprise in the United States. It includes detailed information about mortgage loans issued over multiple years, covering borrower characteristics, loan terms, property features, and loan performance outcomes such as whether the loan was prepaid or defaulted.

[5]:

```
# We load the data below
d = pd.read_csv("freddiemac.csv")
d.head()
```

[5]:

```
   fico  dt_first_pi flag_fthb  dt_matr   cd_msa  mi_pct  cnt_units occpy_sts  \
0   809       201705         N   204704      NaN       0          1         P
1   702       201703         N   203202      NaN       0          1         P
2   792       201703         N   204702      NaN       0          1         S
3   776       201703         N   204702      NaN       0          1         S
4   790       201703         N   204702  41620.0       0          1         I

   cltv  dti  …    seller_name                    servicer_name  flag_sc  \
0    75   38  …  Other sellers  SPECIALIZED LOAN SERVICING LLC       NaN
```

```
1    80   36  …  Other sellers                 Other servicers       NaN
2    60   36  …  Other sellers                 Other servicers       NaN
3    80   18  …  Other sellers                 Other servicers       NaN
4    75   42  …  Other sellers                    PNC BANK, NA       NaN

  id_loan_rr program_ind rr_ind property_val io_ind  mi_cancel_ind loan_status
0        NaN           9    NaN            2      N              7     prepaid
1        NaN           9    NaN            2      N              7      active
2        NaN           9    NaN            2      N              7     prepaid
3        NaN           9    NaN            2      N              7     prepaid
4        NaN           9    NaN            2      N              7      active

[5 rows x 33 columns]
```

We have the following numerical features in our data:

- **fico**: Credit score of the borrower at origination.
- **dt_first_pi**: First payment date (YYYYMM).
- **dt_matr**: Scheduled loan maturity date.
- **mi_pct**: Mortgage insurance percentage.
- **cltv**: Combined loan-to-value ratio.
- **dti**: Debt-to-income ratio.
- **orig_upb**: Original unpaid principal balance.
- **ltv**: Loan-to-value ratio at origination.
- **int_rt**: Interest rate of the loan.
- **orig_loan_term**: Original term length of the loan (in months).

We have the following categorical features in our data

- **flag_fthb**: First-time homebuyer indicator.
- **occpy_sts**: Occupancy status of the property.
- **channel**: Loan origination channel.
- **st**: State where the property is located.
- **prop_type**: Property type.
- **loan_purpose**: Purpose of the loan.

- **seller_name**: Seller of the loan.
- **servicer_name**: Entity servicing the loan.
- **flag_sc**: Super conforming loan indicator.
- **program_ind**: Loan program indicator.
- **rr_ind**: Re-performing refinance indicator.
- **cd_msa**: Metro area indicator.
- **cnt_borr**: Number of borrowers.
- **property_val**: Property valuation method.
- **mi_cancel_ind**: Mortgage insurance cancellation status.
- **cnt_units**: Number of housing units.

- We excluded non-feature columns such as **ID** and **loan_status**, as they are either identifiers or already used to define the target variable.
- We dropped the columns **ppnt_pnlty**, **prod_type**, and **io_ind** because each of them contains only a single unique value across all loans, providing no variance or predictive power for modeling.
- The **zipcode** column was also discarded. Although it could represent geographic information, we already included the state abbreviation (**st**) as a regional indicator. Using **zipcode** as a numeric variable is not meaningful, and treating it as a categorical variable would introduce too many levels, which is inappropriate for most machine learning models.

1.3.2 Test-Training Spilt

To ensure a fair evaluation and prevent data leakage, we first split the dataset into training and testing subsets before performing any data preprocessing or exploratory analysis. In this case, we retained only the loans with status 'default' and 'prepaid', and then constructed a binary target variable where 1 indicates default and 0 indicates prepaid.

We used an 80/20 train-test split with stratification on the target variable to preserve the proportion of default cases in both sets. The stratified split is particularly important due to the imbalanced nature of the dataset. This early split ensures that the test set remains completely unseen throughout the entire feature engineering and model training process, avoiding any leakage of information that could lead to over-optimistic model performance.

By working only with the training set during preprocessing and EDA, we maintain a clean pipeline that mirrors real-world deployment scenarios and results in more trustworthy evaluation metrics.

[6]:

```
# classify categorical and numerical varaiables
categorical_columns = [
    'flag_fthb', 'occpy_sts', 'channel', 'st',
    'prop_type', 'loan_purpose', 'seller_name',
    'servicer_name', 'flag_sc', 'program_ind', 'rr_ind',
    'cd_msa', 'cnt_borr',
    'property_val', 'mi_cancel_ind','cnt_units'
]

numerical_columns = [
    'fico', 'dt_first_pi', 'dt_matr',  'mi_pct',
     'cltv', 'dti', 'orig_upb', 'ltv',
    'int_rt', 'orig_loan_term'
]

feature_columns = categorical_columns + numerical_columns

data_labeled = d[d['loan_status'].isin(['default', 'prepaid'])]

data_labeled['target'] = (data_labeled['loan_status'] == 'default').astype(int)

X = data_labeled[feature_columns]
y = data_labeled["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪test_size=0.2, random_state=42)

eda_data = X_train.copy()
eda_data['loan_status'] = y_train
```

1.3.3 Handing Missing Data

In this dataset, missing values are not consistently represented as standard `NaN`, but are instead encoded using placeholder values such as `9`, `99`, `999`, or `9999`, depending on the column.

To facilitate consistent missing value handling in the preprocessing stage, we first created a mapping (`missing_value_map`) that specifies which placeholder values correspond to missing data for each relevant column.

We then applied this mapping across the dataset to replace all these special codes with proper `NaN` values using the `.replace()` function. This ensures that later stages of analysis and imputation can recognize and treat these entries as standard missing data.

[7]:
```
# define missing value map
missing_value_map = {
    'fico': [9999],
    'mi_pct': [999],
    'cltv': [999],
    'dti': [999],
    'ltv': [999],
    'flag_fthb': ['9'],
    'occpy_sts': ['9'],
    'channel': ['9'],
    'prop_type': ['99'],
    'loan_purpose': ['9'],
    'program_ind': ['9'],
    'rr_ind': ['9'],
    'cnt_borr': ['9'],
    'property_val': ['9'],
    'mi_cancel_ind': ['9'],
    'cnt_units': ['99']
}

# replace
for col, missing_vals in missing_value_map.items():
    eda_data[col] = eda_data[col].replace(missing_vals, np.nan)
```

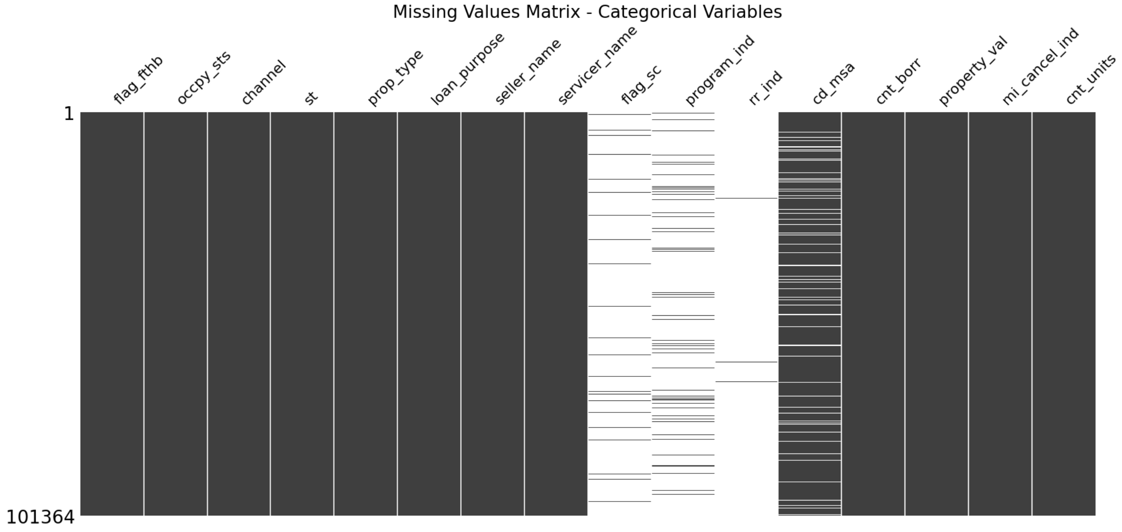
[8]:

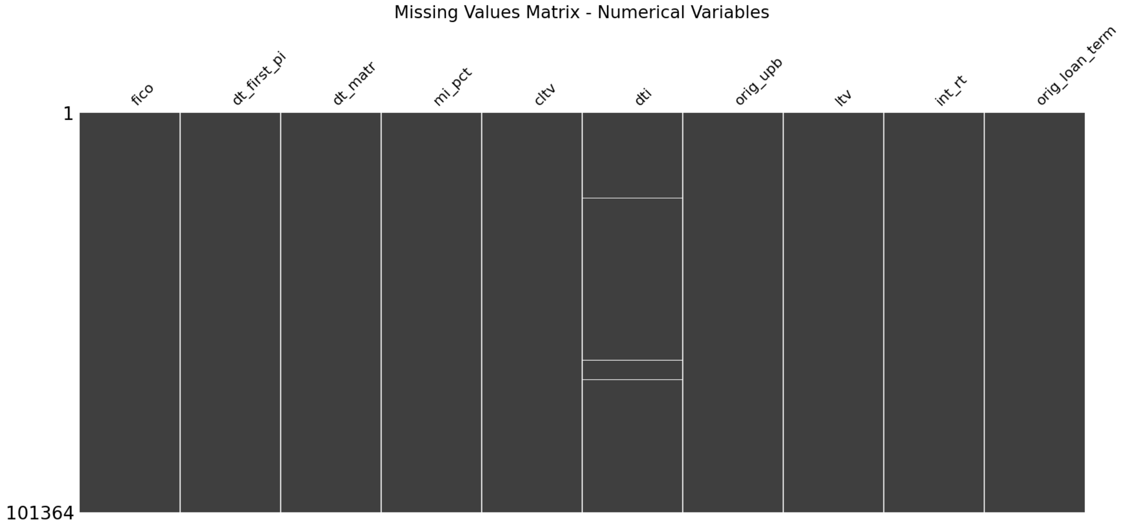
```
# have a overview of dataset and missing value
eda_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 101364 entries, 96029 to 137892
Data columns (total 27 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   flag_fthb       101364 non-null  object
 1   occpy_sts       101364 non-null  object
 2   channel         101364 non-null  object
 3   st              101364 non-null  object
 4   prop_type       101364 non-null  object
 5   loan_purpose    101364 non-null  object
 6   seller_name     101364 non-null  object
 7   servicer_name   101364 non-null  object
 8   flag_sc         4389 non-null    object
 9   program_ind     8166 non-null    object
 10  rr_ind          1066 non-null    object
 11  cd_msa          92419 non-null   float64
 12  cnt_borr        101364 non-null  int64
 13  property_val    101364 non-null  int64
 14  mi_cancel_ind   101364 non-null  object
 15  cnt_units       101364 non-null  int64
 16  fico            101346 non-null  float64
 17  dt_first_pi     101364 non-null  int64
 18  dt_matr         101364 non-null  int64
 19  mi_pct          101363 non-null  float64
 20  cltv            101363 non-null  float64
 21  dti             100293 non-null  float64
 22  orig_upb        101364 non-null  int64
 23  ltv              101363 non-null  float64
 24  int_rt           101364 non-null  float64
 25  orig_loan_term   101364 non-null  int64
 26  loan_status      101364 non-null  int64
dtypes: float64(7), int64(8), object(12)
memory usage: 21.7+ MB
```

Before deciding how to handle missing values in our mortgage dataset, we first assessed the extent and nature of the missingness. To understand whether dropping or imputing missing values is appropriate, we visualized the missing value matrix. This step helped us identify patterns and determine if the missingness appears random or systematic, which informs our later decision-making on imputation strategies or variable removal.

[9]:
```python
# Convert categorical columns to 'object' dtype for consistency
eda_data[categorical_columns] = eda_data[categorical_columns].astype("object")

# plot missing values
msno.matrix(eda_data[categorical_columns], fontsize=20, sparkline=False)
plt.title("Missing Values Matrix - Categorical Variables", fontsize = 24)
plt.show()

msno.matrix(eda_data[numerical_columns], fontsize=20, sparkline=False)
plt.title("Missing Values Matrix - Numerical Variables", fontsize = 24)
plt.show()
```





The dataset contains a small number of missing values across both categorical and numerical variables. As shown in the missing value matrices above:

- Categorical variables such as flag_sc, program_ind, rr_ind, and cd_msa have visible missing patterns.
- Numerical variable dti also have some missing entries.

Most other variables are fully observed. Since the proportion of missing values for numerical varaiables is relatively low, simple imputation (e.g., median for skewed numerical values and constant for categorical features) is feasible.

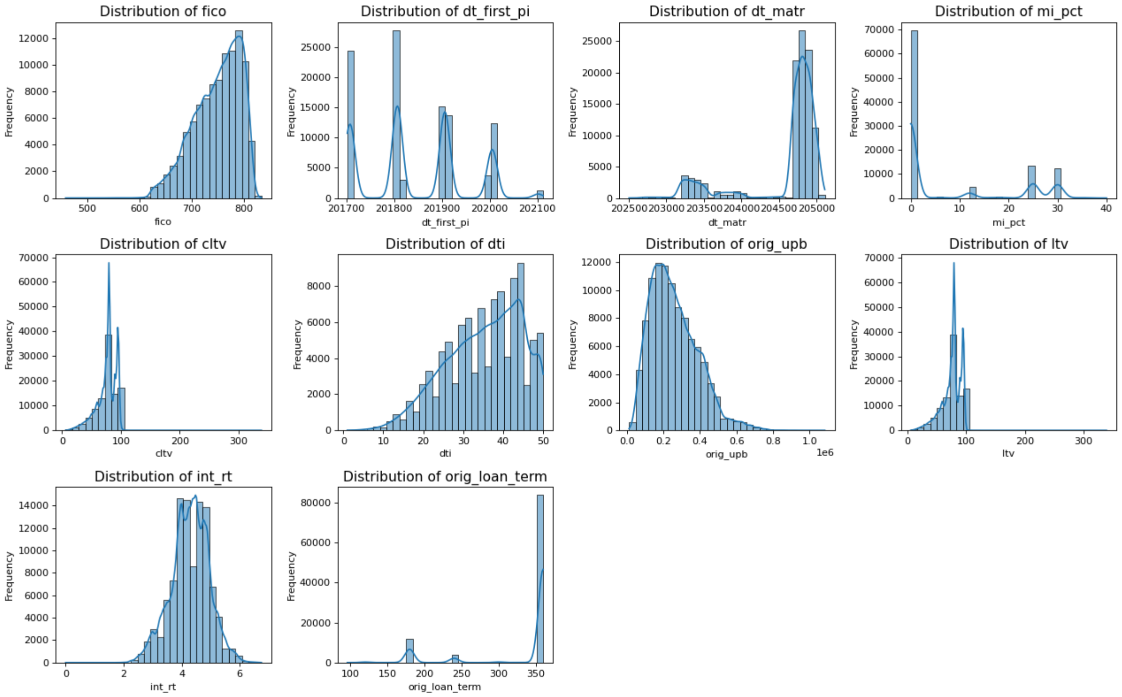
Through our analysis, we found that the majority of features are complete and do not require imputation. However, for those that do contain missing values, we adopt the following logic to handle them:

- **Drop entire columns** if a variable has too many missing values, making imputation unreliable.
- **Use the mean** to fill in missing values for numerical variables with approximately normal distributions.
- **Use the median** when the variable is skewed and the number of missing values is low.
- **Use the mode** (most frequent category) to impute categorical variables with limited missingness and no special meaning attached to the missing value.
- **Introduce a special category** such as ‘missing’ for categorical variables where missingness may carry meaningful or structural information.

This strategy ensures that we retain as much useful data as possible while minimizing the risk of introducing bias or data leakage during preprocessing.

[10]:

```python
# Then we need to find out the distribution of numerical varaiables
# define plot set
n_cols = 4
n_rows = math.ceil(len(numerical_columns) / n_cols)
fig, axes = plt.subplots(n_rows, n_cols,figsize=(16,10))
axes = axes.flatten()

# plot
for i, col in enumerate(numerical_columns):
    sns.histplot(eda_data[col], bins=30, kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {col}', fontsize=14)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



Distribution Analysis of Numerical Features The figure above displays the distributions of key numerical variables used in the model. Several important observations are summarized as follows:

- **fico** and **dti** show strong right-skewed distributions, indicating a large proportion of borrowers have relatively high FICO scores and low debt-to-income ratios. Due to this skewness, imputing missing values using the median is more appropriate than using the mean.

- **cltv** and **ltv** exhibit very similar distribution shapes, both showing a sharp concentration around 80–100. This suggests potential redundancy between the two features. Further correlation analysis will be conducted later to assess their multicollinearity and decide whether both should be retained in the final model.
- **mi_pct** (mortgage insurance percentage) displays an extremely irregular and skewed distribution, with a majority of values at zero and scattered peaks at specific insurance bands (e.g., 20%, 30%). This pattern may reflect business rules rather than organic variability.
- **dt_first_pi** and **dt_matr** (first payment date and maturity date) show patterns aligned with calendar cycles (e.g., spikes at specific months/years), which may not contribute to predictive power unless transformed into derived features such as loan duration.
- **int_rt** (interest rate) appears fairly normally distributed, which is ideal for many models and unlikely to need transformation.

Missing Value Imputation Strategy Based on the visualizations above and careful inspection of each variable, we applied the following tailored imputation strategies:

fico: Since the distribution is skewed, we filled missing values using the median.

cd_msa: Missing values may indicate loans from non-metropolitan areas, so we introduced a new category ‘missing’ to represent them.

mi_pct: This column has only one missing value, which we handled by directly dropping the corresponding row.

cltv: With only six missing values, we also dropped those rows instead of imputing.

dti: The variable is skewed, so we filled in missing values with the median.

ltv: There are only two missing values, so we dropped those rows.

flag_sc: Missing values are interpreted as “No”, and we replaced them with the category ‘N’.

id_loan_rr: This variable is not used in downstream modeling and was therefore ignored.

program_ind: Missing values might carry meaning, so we created a new category ‘missing’ to capture them.

rr_ind: Similar to flag_sc, we interpreted missingness as “No” and replaced with ‘N’.

This mixed strategy balances domain logic, data quality, and statistical distribution, ensuring robust preparation before modeling.

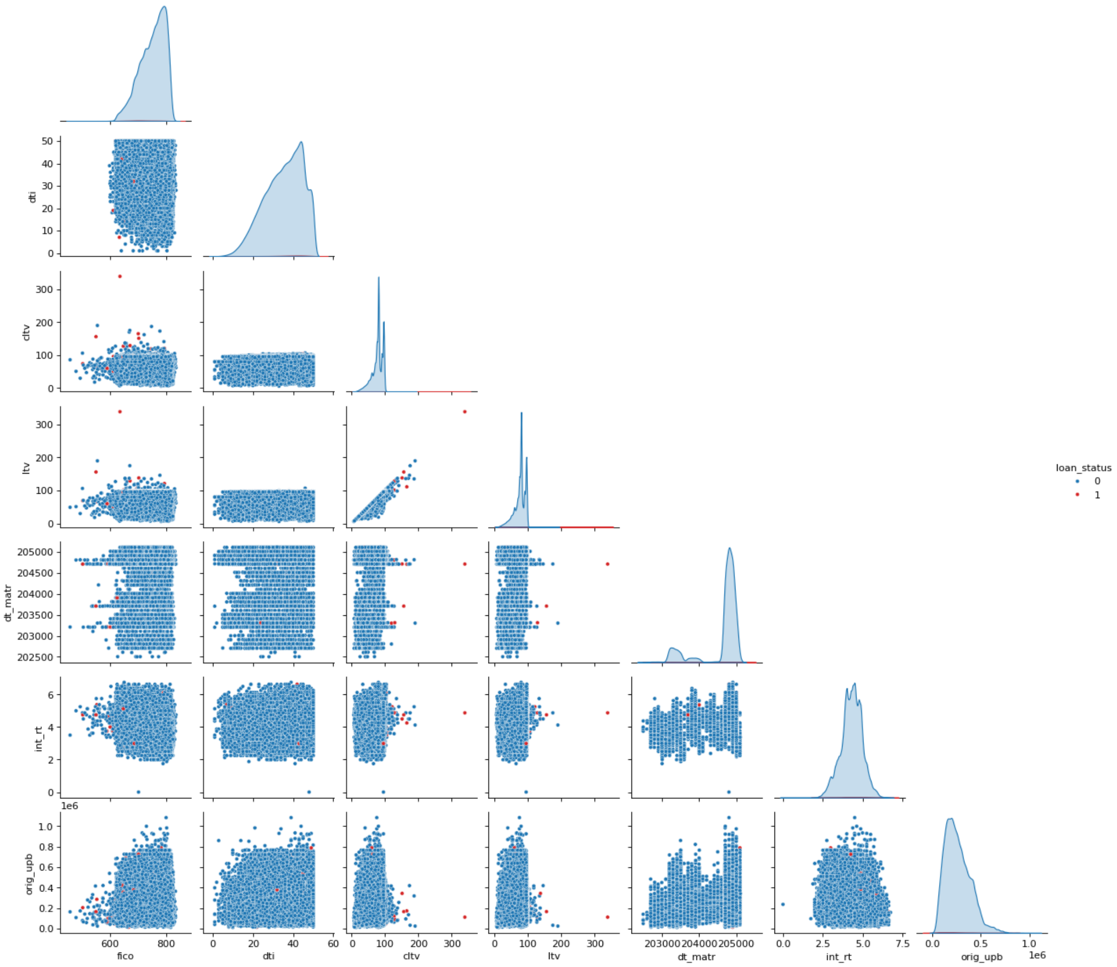
[11]:

```
# Missing value imputation
def clean_data(df):
    df = df.copy()

    # drop na
    df = df[df['mi_pct'].notna()]
    df = df[df['cltv'].notna()]
    df = df[df['ltv'].notna()]
    # add new group
    df['flag_sc'] = df['flag_sc'].fillna('N')
    df['program_ind'] = df['program_ind'].fillna('missing')
    df['rr_ind'] = df['rr_ind'].fillna('N')
    df['cd_msa'] = df['cd_msa'].apply(lambda x: 'Not Available' if pd.isna(x)␣
 ↪or x == 'Not Available' else 'Available')

    return df

X_train_clean = clean_data(X_train)
X_test_clean = clean_data(X_test)
eda_data=clean_data(eda_data)
```

1.3.4 Investigating Relationships

[12]:

```
# we now create a pairplot to investigate the relationships between the␣
 ↪variables
# set variables to plot
pair_vars = ['fico', 'dti', 'cltv', 'ltv' ,'dt_matr' , 'int_rt', 'orig_upb']

sns.pairplot(
    data=eda_data[pair_vars + ['loan_status']],
    hue='loan_status',
    palette=['#1f77b4','#d62728'],
    corner=True,
    plot_kws={'alpha': 1, 's': 15}
).fig.set_size_inches(16, 14)
```



Feature Pairwise Relationships with Target Overlay The pair plots above visualize the pairwise relationships between key numerical features (fico, dti, cltv, ltv, dt_matr, int_rt, and orig_upb) and their distributions, with the loan outcome (loan_status) overlaid using color. In this context:

- Blue points represent loans that were prepaid (loan_status = 0)
- Red points represent loans that defaulted (loan_status = 1)

This color overlay helps to intuitively observe how the distribution of feature values differs between the two outcome groups.

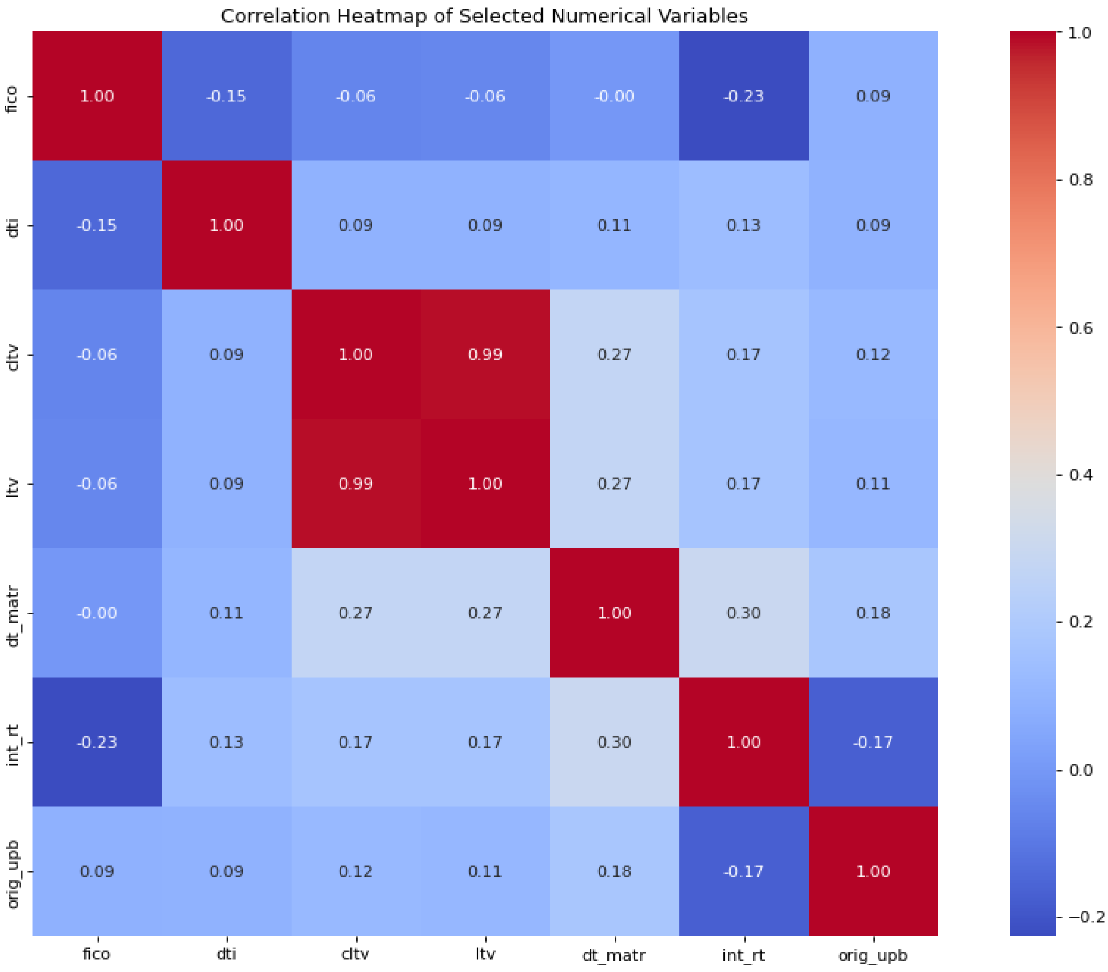
Most red points (defaults) are scattered sparsely, suggesting that defaults are rare and the data is highly imbalanced.

cltv and ltv exhibit very similar distributions, reinforcing our earlier observation. A strong correlation between them is likely and should be formally checked to avoid multicollinearity in modeling.

fico and dti are right-skewed (positively skewed), with a high density of observations at lower fico scores and higher dti ratios among defaulters. These features may benefit from transformation or binning, especially since they carry strong domain relevance.

[13]:

```python
# plot heatmap
corr_matrix = eda_data[pair_vars].corr()
plt.figure(figsize=(16, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title("Correlation Heatmap of Selected Numerical Variables")
plt.show()
```

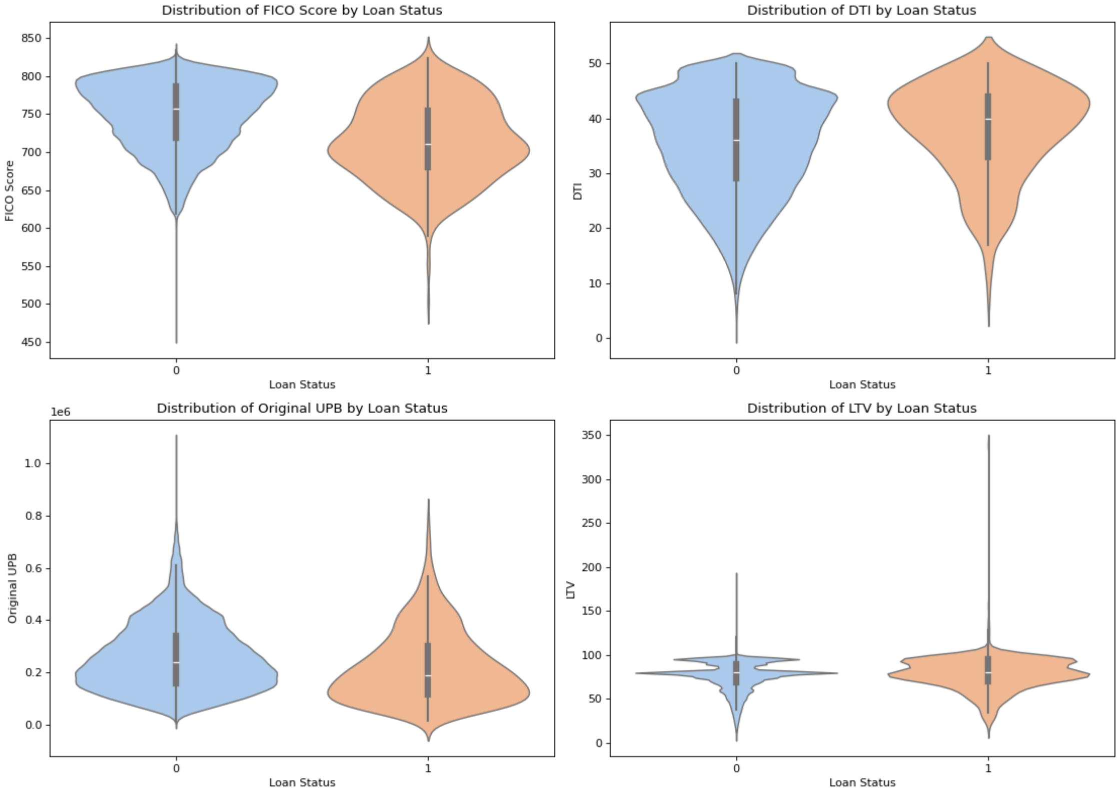


From this figure we can find that the correlation between **ltv** and **cltv** is extremely strong. Although the features **cltv** and **ltv** are highly correlated, we decided to retain both in the dataset. This is because our modeling approach includes LASSO regularized logistic regression and Random Forest, both of which are capable of handling multicollinearity. Therefore, we did not remove either feature to preserve potentially useful information.

[14]:

```python
# Create a 2x2 subplot for four violin plots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))  # 2 rows, 2 columns

# Plot 1: FICO distribution by loan status
```

```python
sns.violinplot(x='loan_status', y='fico', data=eda_data, palette='pastel', ax=axes[0, 0])
axes[0, 0].set_title("Distribution of FICO Score by Loan Status")
axes[0, 0].set_xlabel("Loan Status")
axes[0, 0].set_ylabel("FICO Score")

# Plot 2: DTI distribution by loan status
sns.violinplot(x='loan_status', y='dti', data=eda_data, palette='pastel', ax=axes[0, 1])
axes[0, 1].set_title("Distribution of DTI by Loan Status")
axes[0, 1].set_xlabel("Loan Status")
axes[0, 1].set_ylabel("DTI")

# Plot 3: Original UPB distribution by loan status
sns.violinplot(x='loan_status', y='orig_upb', data=eda_data, palette='pastel', ax=axes[1, 0])
axes[1, 0].set_title("Distribution of Original UPB by Loan Status")
axes[1, 0].set_xlabel("Loan Status")
axes[1, 0].set_ylabel("Original UPB")

# Plot 4: LTV distribution by loan status
sns.violinplot(x='loan_status', y='ltv', data=eda_data, palette='pastel', ax=axes[1, 1])
axes[1, 1].set_title("Distribution of LTV by Loan Status")
axes[1, 1].set_xlabel("Loan Status")
axes[1, 1].set_ylabel("LTV")

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```



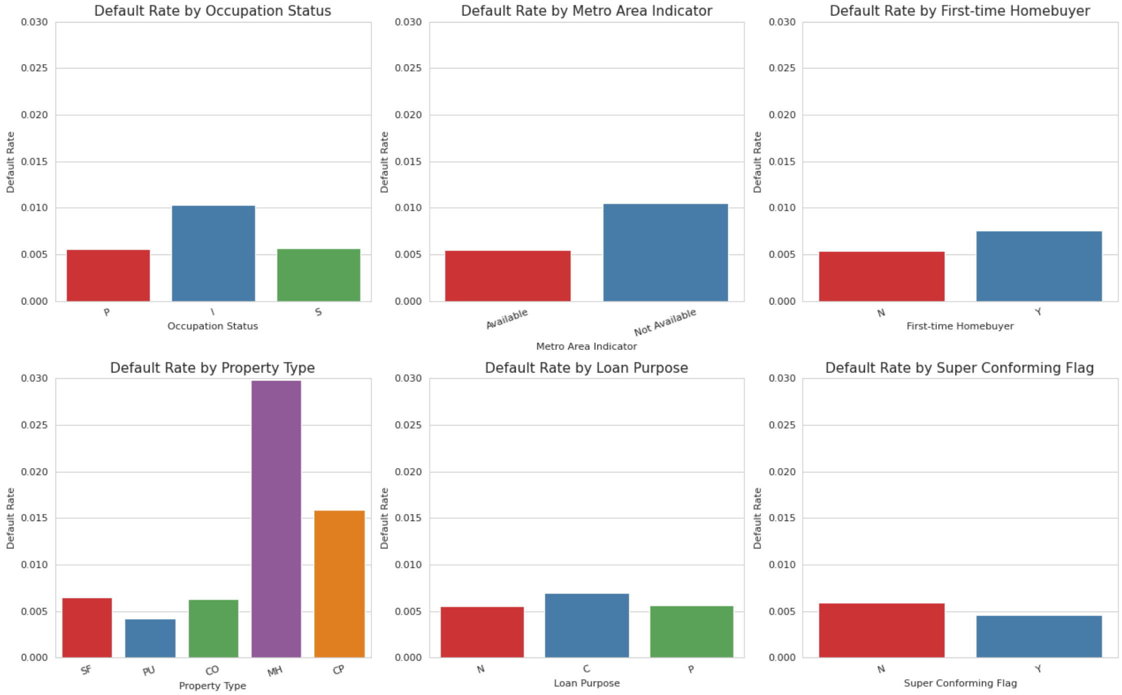
Feature Distributions by Loan Status The violin plots illustrate the distribution of key numerical variables (FICO, DTI, Original UPB, and LTV) across different loan statuses (0: prepaid, 1: default):

- **FICO Score**: Borrowers who defaulted (loan_status = 1) generally have lower FICO scores than those who prepaid, with the distribution noticeably shifted to the left.
- **DTI (Debt-to-Income Ratio)**: Defaulted borrowers tend to have slightly higher DTI values on average, suggesting more financial burden.
- **Original UPB**: The original unpaid principal balance (orig_upb) is more spread out in the prepaid group, while defaulted loans tend to cluster around lower UPB values.
- **LTV (Loan-to-Value Ratio)**: There is a clear difference in LTV distribution, with defaults showing a heavier concentration of high LTV loans, indicating higher risk lending.

These patterns suggest that FICO, DTI, and LTV are all potentially predictive of loan default behavior and worth including in our modeling.

[15]:
```
# Set plot style and resolution
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (4, 5)
plt.rcParams['figure.dpi'] = 80

# Define categorical variables and their display titles
```

```python
categorical_vars = ['occpy_sts', 'cd_msa', 'flag_fthb', 'prop_type', 'loan_purpose', 'flag_sc']
titles = [
    "Occupation Status",
    "Metro Area Indicator",
    "First-time Homebuyer",
    "Property Type",
    "Loan Purpose",
    "Super Conforming Flag"
]

# Create a 2-row, 3-column grid for subplots
fig, axes = plt.subplots(2, 3, figsize=(16, 10))
axes = axes.flatten()

# Use a vivid, high-contrast color palette
contrast_palette = sns.color_palette("Set1")

# Draw bar plots for each categorical variable
for i, var in enumerate(categorical_vars):
    sns.barplot(
        x=var, y='loan_status', data=eda_data,
        palette=contrast_palette, ax=axes[i], ci=None
    )
    axes[i].set_title(f"Default Rate by {titles[i]}", fontsize=14)
    axes[i].set_xlabel(titles[i])
    axes[i].set_ylabel("Default Rate")
    axes[i].set_ylim(0, 0.03)  # Standardize Y axis scale for better comparison
    axes[i].tick_params(axis='x', rotation=20)

# Adjust layout to avoid overlap
plt.tight_layout()
plt.show()
```



Categorical Features and Default Rate The bar plots above show the default rate across several categorical variables. Each subplot represents one feature, grouped by its categories, with the corresponding default rate on the y-axis.

Several patterns emerge from the plots:

- **Property Type** shows the most striking difference: loans for Manufactured Homes (MH) have significantly higher default rates than other property types.
- **Metro Area Availability** and **Occupation Status** also exhibit some differences—loans from areas marked Not Available or with certain occupation codes tend to have slightly higher risk.
- Other variables such as **Loan Purpose**, **First-time Homebuyer Flag**, and **Super Conforming Flag** show only modest variation between categories.

These insights help identify potentially risky segments and guide feature selection or transformation in downstream modeling.

1.3.5 Feature Engineering

To enhance model performance, we applied several feature engineering techniques. Categorical variables were transformed using One-Hot Encoding to make them compatible with machine learning models while preserving important group information. For missing values, we used tailored strategies including median imputation for numerical variables and assigning special categories for categorical ones. In addition, we carefully removed low-variance or redundant features based on domain knowledge to reduce noise and improve interpretability. These preprocessing steps ensured the model received clean and informative inputs for training. This is described further in the next section.

1.4 Model Fitting and Tuning

In this task, one of the most critical challenges is the severe **class imbalance** between default and prepaid loans. To address this, we explored a variety of **machine learning models** including `Logistic Regression`, `LASSO`, `Random Forest`, and `XGBoost`. Each of these models brings unique advantages: `Logistic Regression` offers high interpretability and serves as a solid baseline; `LASSO` enhances this by performing feature selection via regularization, reducing overfitting; `Random Forest` captures nonlinear interactions and is robust to noise; and `XGBoost` provides powerful boosting capabilities and often yields strong predictive performance in imbalanced settings.

To mitigate the imbalance issue, we also experimented with both **oversampling** and **undersampling** strategies, such as `SMOTE` and `Random Undersampling`, as well as hybrid techniques like `SMOTETomek` and `SMOTEENN`.

To establish a performance baseline, we first fitted a standard logistic regression model without any resampling. This allowed us to evaluate the added benefit of more complex models and sampling techniques. Then we fit different models with different sampling strategies.

After fitting all candidate models, we moved to the model selection phase. Our choice of the final model was guided by two primary criteria: **interpretability** and **predictive accuracy**. Interpretability was important because the client requested not only a prediction tool but also insights into which features contribute to default risk. At the same time, we aimed to maximize prediction performance.

In real-world financial applications, the cost of wrongly approving a high-risk loan (a false negative) is typically much higher than that of incorrectly rejecting a low-risk applicant (a false positive). While we do not have a quantifiable cost function to reflect this trade-off, we adjusted our evaluation focus accordingly. As such, we prioritized `recall`—the proportion of actual defaults correctly identified by the model—as a key metric. A higher recall indicates fewer missed defaulters, which is critical for minimizing financial losses. In addition to recall, we also considered `ROC-AUC` and `accuracy` to ensure overall model robustness and balance.

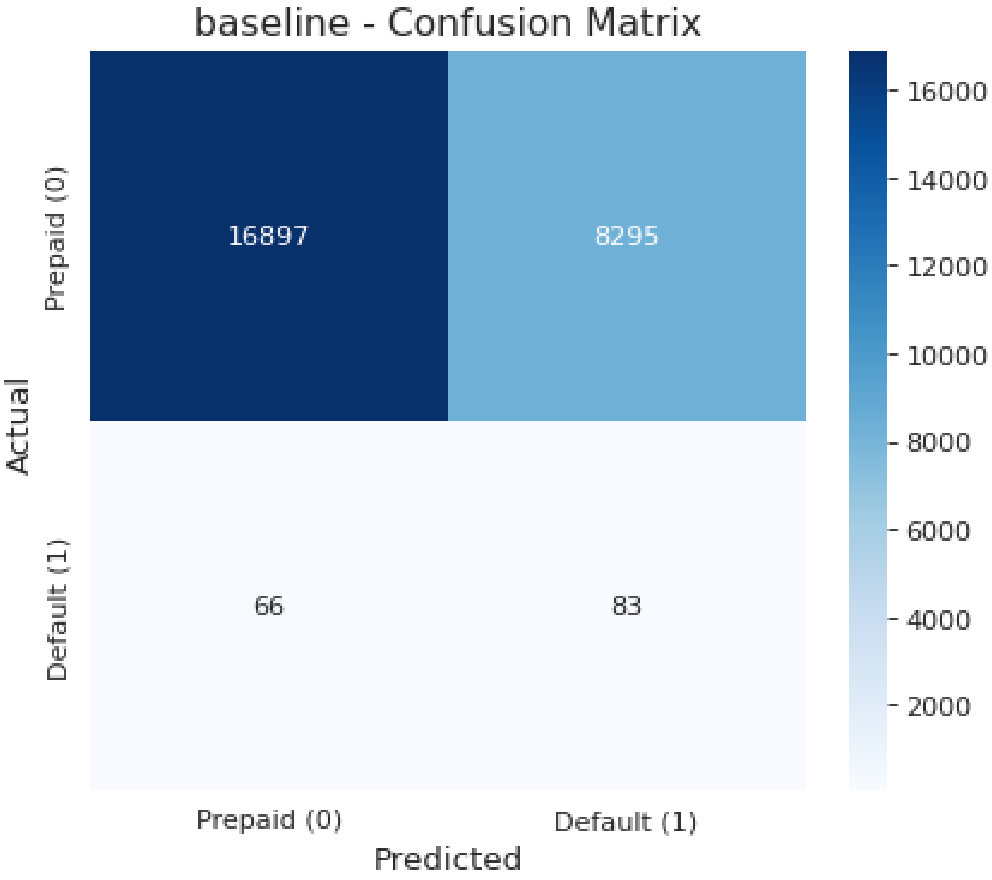
Among all models tested, we ultimately selected the **random forest model** combined with undersampling as our **final model**, due to its strong performance across multiple metrics—especially recall and ROC AUC—while maintaining a relatively straightforward structure that supports interpretation of feature importance.

1.4.1 Baseline Model

We used a **Logistic Regression model** as our **baseline**. Before fitting the model, we built a preprocessing **pipeline** to handle missing values and categorical variables. For numeric features like fico and dti, missing values were filled with the median. For categorical variables, we used constant value filling and one-hot encoding. The model was trained with balanced class weights to address class imbalance. This baseline helps us compare the performance of more advanced models later.

[16]:
```
# data imputation
numerical_imputer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median'))
])

# fill na and on-hot coding for categorical varaiables
categorical_imputer_encoder = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('encoder', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_imputer, ['fico', 'dti']),
    ('cat', categorical_imputer_encoder, categorical_columns)
], remainder='passthrough')

pipe = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('model', LogisticRegression(class_weight='balanced', max_iter=1000))
])

pipe.fit(X_train_clean, y_train)

classification_model_fit(pipe,X_test_clean,y_test,model_name="baseline")
```

```
 Evaluation Metrics (baseline)
------------------------------
Accuracy       : 0.6701
Precision      : 0.0099
Recall         : 0.5570
F1 Score       : 0.0195
ROC AUC Score  : 0.6656
```



```
[16]: {'accuracy': 0.6700603764650171,
       'precision': 0.009906899021246121,
       'recall': 0.5570469798657718,
       'f1': 0.019467573589773666,
       'roc_auc': np.float64(0.6656307744442148),
       'confusion_matrix': array([[16897,  8295],
              [   66,    83]])}
```

1.4.2 Final Model: Random Forest with Undersampling

To address the class imbalance between default and prepaid loans, we implemented a **Random Forest** classifier combined with **random undersampling**. The model was built using a pipeline that first applies preprocessing (missing value imputation and One-Hot Encoding), then performs random undersampling to balance the training data, and finally fits a Random Forest classifier with class-weight adjustment to handle any remaining imbalance.

To evaluate model stability, we conducted 5-fold stratified **cross-validation** using recall as the primary scoring metric, which is particularly relevant for this task as we aim to reduce false negatives (i.e., failing to detect high-risk defaults). The cross-validated recall scores ranged from 0.68 to 0.74, with a mean recall of 0.72, indicating consistent performance across folds.

On the hold-out test set, the model achieved a recall of 0.7181, a ROC AUC of 0.7876, and an overall accuracy of 0.7363. The confusion matrix shows that the model was able to correctly identify 107 defaults, missing only 42, which is crucial in a context where failing to detect defaults can lead to significant financial losses.

We selected this model as the final choice due to its strong performance on recall, interpretability through feature importance, and robustness achieved through ensemble learning. Moreover, the undersampling strategy helped mitigate the bias toward the majority class without requiring synthetic data generation.

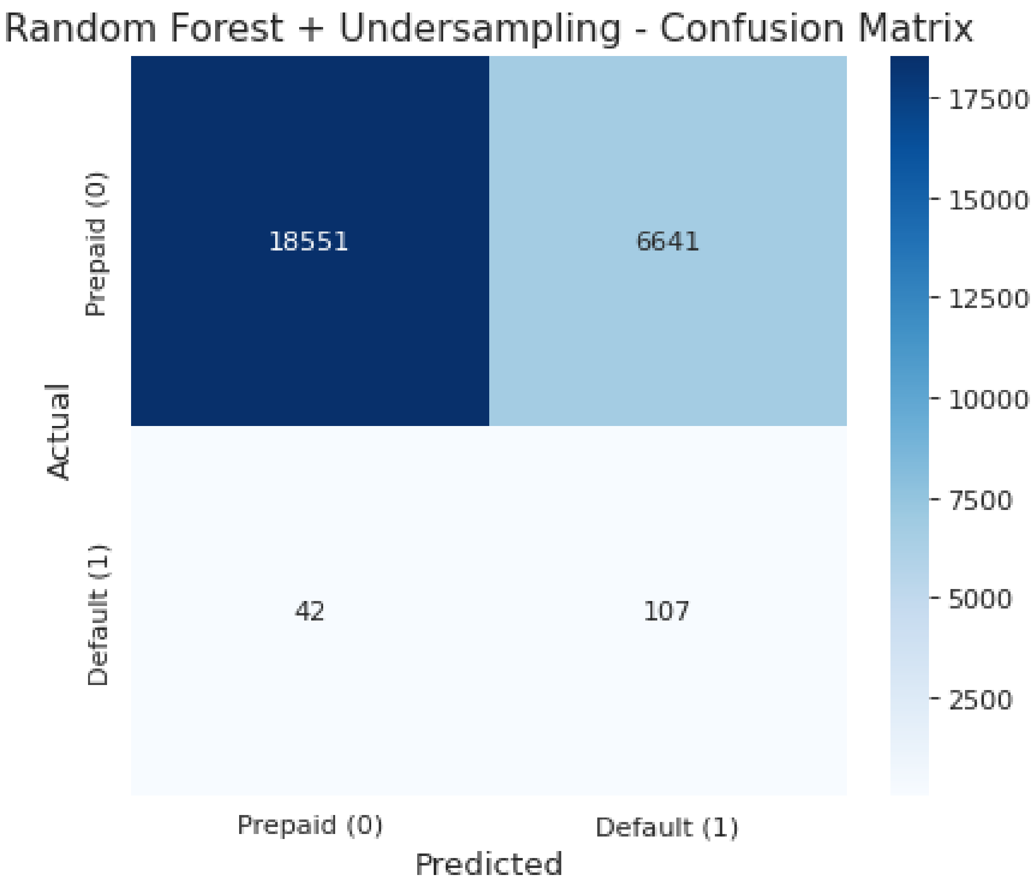
Other models were excluded due to either poor predictive performance or limited interpretability. Lasso regression, while interpretable, showed low recall (max 0.65), making it unreliable for identifying defaults. XGBoost achieved high recall and AUC, but it is difficult to interpret — a key requirement for this task. Some models like SMOTE-only logistic regression had recall below 0.52, and others like plain Random Forest had zero recall, failing to detect any defaults. Therefore, these models did not offer a better balance between accuracy and interpretability compared to our chosen model.

[17]:

```
# Construct pipeline with undersampling and Random Forest, and add␣
 ↪preprocessing (OneHot + Imputation)
rf_under_pipeline = ImbPipeline(steps=[
    ('preprocess', preprocessor),  # Apply preprocessing (e.g., OneHotEncoder +␣
 ↪missing value imputation)
    ('sampler', RandomUnderSampler(random_state=42)),  # Perform random␣
 ↪undersampling to balance classes
    ('classifier', RandomForestClassifier(
        n_estimators=100,
        class_weight='balanced',
        random_state=42
    ))  # Train Random Forest with class_weight balancing
])

# Define Stratified K-Fold Cross-Validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Optional: use recall as primary evaluation metric
recall_scorer = make_scorer(recall_score)

# Perform cross-validation with recall as the main scoring metric
cv_scores = cross_val_score(
    rf_under_pipeline,
    X_train_clean,
    y_train,
    scoring=recall_scorer,  # You can also use 'roc_auc', 'accuracy', etc.
    cv=cv,
    n_jobs=-1  # Use all cores for parallel processing
)
```

```python
# Print cross-validated recall scores
print("Cross-validated Recall Scores:", cv_scores)
print("Mean Recall:", cv_scores.mean())

# Final fit on the full training set (for test evaluation)
rf_under_pipeline.fit(X_train_clean, y_train)

# Evaluate model on test set using custom function
classification_model_fit(
    rf_under_pipeline,
    X_test_clean,
    y_test,
    model_name="Random Forest + Undersampling"
)
```

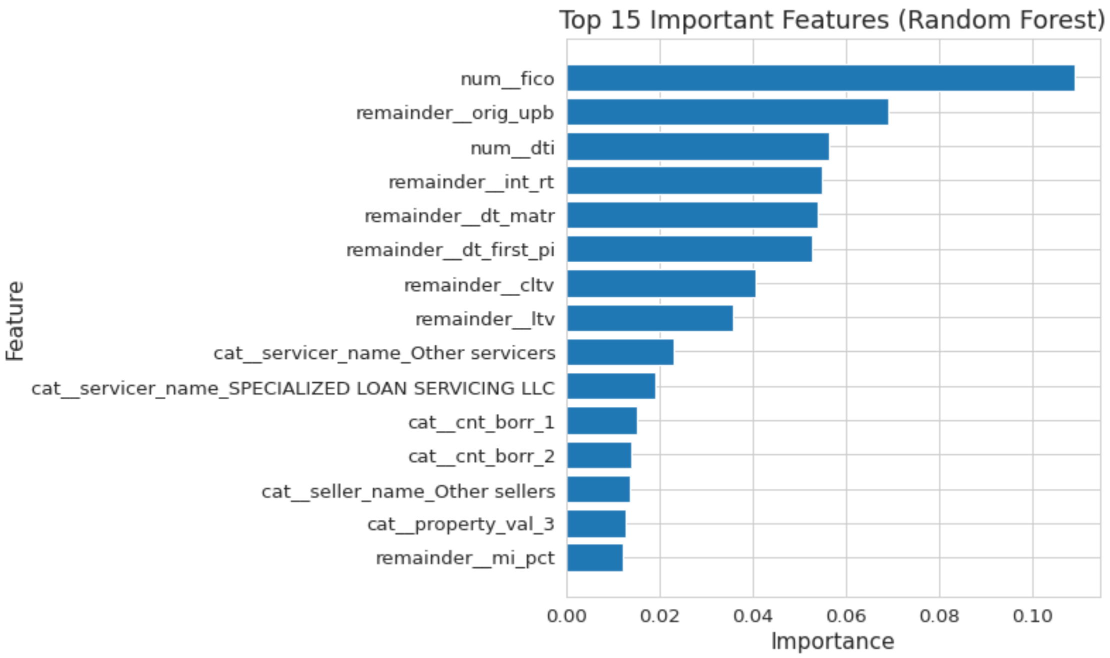
```
Cross-validated Recall Scores: [0.78151261 0.68067227 0.73333333 0.74166667
0.67226891]
Mean Recall: 0.721890756302521
 Evaluation Metrics (Random Forest + Undersampling)
------------------------------
Accuracy       : 0.7363
Precision      : 0.0159
Recall         : 0.7181
F1 Score       : 0.0310
ROC AUC Score  : 0.7876
```



[17]:
```
{'accuracy': 0.7362771792746932,
 'precision': 0.015856550088915233,
 'recall': 0.7181208053691275,
 'f1': 0.03102798318109323,
 'roc_auc': np.float64(0.7875561326595638),
 'confusion_matrix': array([[18551,  6641],
        [   42,   107]])}
```

[18]:
```
# extract classifier
rf_model = rf_under_pipeline.named_steps['classifier']
feature_names = rf_under_pipeline.named_steps['preprocess'].
 ↪get_feature_names_out()
importances = rf_model.feature_importances_

# build DataFrame
feat_imp_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪importances})
feat_imp_df = feat_imp_df.sort_values(by='Importance', ascending=False)
```

```
# plot
plt.figure(figsize=(10, 6))
plt.barh(feat_imp_df['Feature'][:15][::-1], feat_imp_df['Importance'][:15][::
    ↪-1])
plt.xlabel('Importance', fontsize=14)
plt.ylabel('Feature', fontsize=14)
plt.title('Top 15 Important Features (Random Forest)', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```



1.4.3 Identifying High-Risk Factors

To identify key drivers of default risk, we utilized the built-in feature importance attribute of the Random Forest classifier. This method evaluates how frequently and effectively each variable is used in decision tree splits across the ensemble, thus highlighting which features contribute most to the model's predictions. Specifically, we fitted our final model (Random Forest with undersampling and preprocessing) and extracted the top 15 most important features based on the feature_importances_ attribute.

- The bar chart above displays the relative importance of each feature. As observed, **fico** score stands out as the most influential predictor, with an importance score exceeding 0.10. This confirms expectations, as a lower **FICO** score is generally indicative of poor creditworthiness and higher default risk. Other high-impact variables include:
- **orig_upb (original unpaid balance):** Larger loan amounts may correspond to higher risk

due to increased financial burden.

- **dti (debt-to-income ratio):** A higher DTI signals financial strain, which increases the chance of default.
- **int_rt (interest rate):** Higher interest rates imply larger monthly payments and a greater chance of delinquency.
- **ltv / cltv (loan-to-value ratios):** High values suggest less equity in the property, reducing collateral security for lenders.

Additionally, some categorical features such as **servicer_name, seller_name**, and **number of borrowers** also appear, indicating that institutional factors and borrower composition can influence default risk.

Potential High-Risk Profiles Based on these insights, we can preliminarily define potential high-risk individuals as those who:

- Have a **low FICO score**,
- **Carry high loan amounts** with limited down payments (high LTV/CLTV),
- Face **high monthly payment obligations** (due to high DTI or interest rate),
- And possibly originate from **certain servicers or sellers** with historically higher default patterns.

This information can be valuable not only for predictive modeling but also for future risk control strategies and lending policy adjustments.

1.5 Discussion & Conclusions

Our final model—**a Random Forest classifier with undersampling**—demonstrated strong performance in identifying high-risk mortgage defaults. With a `recall` of approximately 0.72, the model effectively captures the majority of actual defaults, making it highly suitable for real-world applications where missing a high-risk loan could result in significant financial loss. Additionally, the model maintains a reasonable `AUC score` of 0.79, indicating solid overall classification ability.

Using feature importance analysis, we identified **FICO score**, **original unpaid balance (UPB)**, **debt-to-income ratio (DTI)**, and **interest rate** as the most influential variables. This provides valuable insight into the key drivers of default risk, and supports targeted risk management strategies for lending institutions. These results are interpretable and actionable, enabling banks to prioritize customers with low credit scores, high loan amounts, and less favorable financial profiles.

While the model performs well, one major limitation lies in the absence of a defined cost or loss function. In practice, the cost of misclassifying a defaulter (false negative) is much higher than wrongly rejecting a safe borrower (false positive). However, our data does not quantify these costs, so we are unable to fully incorporate cost-sensitive learning or precisely evaluate the financial consequences of prediction errors.

Despite this, the high recall ensures that most potential defaults are captured, aligning with the institution's need to minimize financial risk. Looking ahead, integrating actual financial loss values and applying cost-sensitive training could further enhance decision-making accuracy. Additionally, the model could be adapted to flag active loans at risk, allowing timely interventions such as restructured repayment plans or enhanced monitoring.

In summary, our model balances predictive performance with interpretability, and serves as a practical tool for identifying high-risk loans and supporting credit risk assessment in the mortgage lending process.

1.6 Generative AI statement

Generative AI tools (specifically, ChatGPT) were used throughout the project to assist in several areas. These included identifying and resolving coding errors during model development, and improving the clarity and fluency of the written report.

1.7 References

[19]:

```
# Run the following to render to PDF
!jupyter nbconvert --to pdf project2.ipynb
```

```
[NbConvertApp] Converting notebook project2.ipynb to pdf
[NbConvertApp] ERROR | Error while converting 'project2.ipynb'
Traceback (most recent call last):
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/nbconvertapp.py", line 487, in export_single_notebook
    output, resources = self.exporter.from_filename(
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/exporters/templateexporter.py", line 390, in from_filename
    return super().from_filename(filename, resources, **kw)  #
type:ignore[return-value]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/exporters/exporter.py", line 201, in from_filename
    return self.from_file(f, resources=resources, **kw)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/exporters/templateexporter.py", line 396, in from_file
    return super().from_file(file_stream, resources, **kw)  #
type:ignore[return-value]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/exporters/exporter.py", line 220, in from_file
    return self.from_notebook_node(
           ^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/exporters/pdf.py", line 184, in from_notebook_node
    latex, resources = super().from_notebook_node(nb, resources=resources, **kw)
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/exporters/latex.py", line 92, in from_notebook_node
```

```
    return super().from_notebook_node(nb, resources, **kw)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/exporters/templateexporter.py", line 429, in
from_notebook_node
    output = self.template.render(nb=nb_copy, resources=resources)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/codespace/.local/lib/python3.12/site-
packages/jinja2/environment.py", line 1295, in render
    self.environment.handle_exception()
  File "/home/codespace/.local/lib/python3.12/site-
packages/jinja2/environment.py", line 942, in handle_exception
    raise rewrite_traceback_stack(source=source)
  File
"/home/codespace/.local/share/jupyter/nbconvert/templates/latex/index.tex.j2",
line 8, in top-level template code
    ((* extends cell_style *))
    ^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/codespace/.local/share/jupyter/nbconvert/templates/latex/style_jup
yter.tex.j2", line 176, in top-level template code
    \prompt{(((prompt)))}{(((prompt_color)))}{(((execution_count)))}{(((extra_sp
ace)))}
    ^^^^^^^^^^^^^^^^^^^^^^^^^

  File
"/home/codespace/.local/share/jupyter/nbconvert/templates/latex/base.tex.j2",
line 7, in top-level template code
    ((*- extends 'document_contents.tex.j2' -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/codespace/.local/share/jupyter/nbconvert/templates/latex/document_
contents.tex.j2", line 51, in top-level template code
    ((*- block figure scoped -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/codespace/.local/share/jupyter/nbconvert/templates/latex/display_p
riority.j2", line 5, in top-level template code
    ((*- extends 'null.j2' -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/codespace/.local/share/jupyter/nbconvert/templates/latex/null.j2",
line 30, in top-level template code
    ((*- block body -*))
  File
"/home/codespace/.local/share/jupyter/nbconvert/templates/latex/base.tex.j2",
line 241, in block 'body'
    ((( super() )))
  File "/home/codespace/.local/share/jupyter/nbconvert/templates/latex/null.j2",
line 32, in block 'body'
    ((*- block any_cell scoped -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/home/codespace/.local/share/jupyter/nbconvert/templates/latex/null.j2",
line 85, in block 'any_cell'
    ((*- block markdowncell scoped-*)) ((*- endblock markdowncell -*))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/share/jupyter/nbconvert/templates/latex/document_
contents.tex.j2", line 68, in block 'markdowncell'
    ((( cell.source | citation2latex | strip_files_prefix |
convert_pandoc('markdown+tex_math_double_backslash', 'json',extra_args=[]) |
resolve_references | convert_explicitly_relative_paths |
convert_pandoc('json','latex'))))
    ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/filters/pandoc.py", line 36, in convert_pandoc
    return pandoc(source, from_format, to_format, extra_args=extra_args)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/utils/pandoc.py", line 50, in pandoc
    check_pandoc_version()
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/utils/pandoc.py", line 98, in check_pandoc_version
    v = get_pandoc_version()
        ^^^^^^^^^^^^^^^^^^^^
  File "/home/codespace/.local/lib/python3.12/site-
packages/nbconvert/utils/pandoc.py", line 75, in get_pandoc_version
    raise PandocMissing()
nbconvert.utils.pandoc.PandocMissing: Pandoc wasn't found.
Please check that pandoc is installed:
https://pandoc.org/installing.html
```