project2

April 10, 2025

1 Machine Learning in Python - Project 2

Due Friday, Apr 11th by 4 pm.

Include contributors names in notebook metadata or here

2

2.1 Setup

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

[54]:
```
# Data libraries
import pandas as pd
import numpy as np
# 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
from sklearn.model_selection import train_test_split, RandomizedSearchCV,␣
 ↪StratifiedKFold
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler,␣
 ↪PowerTransformer
from sklearn.compose import ColumnTransformer
from sklearn.metrics import (
    classification_report, accuracy_score, precision_score, recall_score,␣
 ↪f1_score,
    roc_auc_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay,␣
 ↪PrecisionRecallDisplay
)
# Imbalanced-learn modules
from imblearn.pipeline import Pipeline as ImPipeline
from imblearn.over_sampling import SMOTE
```

```
# XGBoost
from xgboost import XGBClassifier
# SHAP
import shap
# Skewness
from scipy.stats import skew, probplot
# Statsmodels for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[41]:
```
# Load data in freddiemac.csv with specific dtype handling for categorical and␣
↪coded columns
d = pd.read_csv("freddiemac.csv", dtype={
    'cd_msa': str, 'cnt_units': str, 'cnt_borr': str, 'zipcode': str,␣
↪'id_loan_rr': str, 'rr_ind':str, 'property_val': str
})
```

3 Introduction

This section should include a brief introduction to the task and the data (assume this is a report you are delivering to a professional body (e.g. Freddie Mac).

Briefly outline the approaches being used and the conclusions that you are able to draw.

4 Exploratory Data Analysis and Feature Engineering

Include a detailed discussion of the data with a particular emphasis on the features of the data that are relevant for the subsequent modeling. Including visualizations of the data is strongly encouraged - all code and plots must also be described in the write up. Think carefully about whether each plot needs to be included in your final draft and the appropriate type of plot and summary for each variable type - your report should include figures but they should be as focused and impactful as possible.

You should also split your data into training and testing sets, ideally before you look to much into the features and relationships with the target

Additionally, this section should also motivate and describe any preprocessing / feature engineering of the data. Specifically, this should be any code that you use to generate new columns in the data frame `d`. Pipelines should be used and feature engineering steps that are be performed as part of an sklearn pipeline can be mentioned here but should be implemented in the following section.

All code and figures should be accompanied by text that provides an overview / context to what is being done or presented.

4.1 Filter Active Loans and Check Target Distribution

[42]:
```
# Filter out active loans (only keep 'default' and 'prepaid')
d_filtered = d[d['loan_status'].isin(['default', 'prepaid'])].copy()
```

```
d_filtered['loan_status'] = d_filtered['loan_status'].map({'default': 1,␣
 ↪'prepaid': 0})
print("Data shape after filtering active loans:", d_filtered.shape)

# Check target distribution
target_dist = d_filtered['loan_status'].value_counts(normalize=True) * 100
print("\nTarget Distribution (%):")
print(target_dist)
```

```
Data shape after filtering active loans: (126705, 33)

Target Distribution (%):
loan_status
0    99.411231
1     0.588769
Name: proportion, dtype: float64
```

4.2 Find Missing Values and Drop Features

[43]:
```
# Define not available values as missing values
missing_values = {'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'], 'loan_purpose': ['9'],
                  'program_ind': ['9'], 'property_val': ['9'], 'mi_cancel_ind':␣
 ↪['7', '9'], 'flag_sc': ['N'], 'rr_ind': ['N']
}
# Replace missing values with NaN
for col, codes in missing_values.items():
    d_filtered[col] = d_filtered[col].replace(codes, np.nan)

# Check for missing values
missing_values = d_filtered.isna().sum().sort_values(ascending=False)
missing_percent = (missing_values / len(d_filtered)) * 100
missing_df = pd.DataFrame({'Missing Count': missing_values, 'Percentage (%)':␣
 ↪missing_percent})[missing_values > 0]
print("Missing Values in Training Data:")
print(missing_df)
```

```
Missing Values in Training Data:
               Missing Count  Percentage (%)
id_loan_rr            125406       98.974784
rr_ind                125406       98.974784
flag_sc               121241       95.687621
program_ind           116496       91.942702
mi_cancel_ind          87026       68.683951
```

```
cd_msa                  11294        8.913618
dti                      1304        1.029162
property_val               94        0.074188
fico                       24        0.018942
ltv                         1        0.000789
mi_pct                      1        0.000789
cltv                        1        0.000789
```

description not available

[44]:

```
print("Columns to drop:")
# Check for columns with >90% missing values
missing_pct = d_filtered.isna().mean()
high_missing_cols = missing_pct[missing_pct > 0.9].index.tolist()
print("Columns with >90% missing values:", high_missing_cols)

# Check for columns with all same non-NaN values
constant_cols = []
for col in d_filtered.columns:
    if d_filtered[col].nunique(dropna=True) == 1:
        constant_cols.append(col)
print("Columns with constant values:", constant_cols)

# Drop identifier columns
ide_cols = ['id_loan', 'seller_name', 'servicer_name']
print("Identifier columns:", ide_cols)

# Drop date columns redundant with 'orig_loan_term' (loan term is derived from␣
↪first payment and maturity dates)
date_cols = ['dt_first_pi', 'dt_matr']
print("Date columns:", date_cols)

# Drop columns with constant values, >90% missing values, and identifier columns
cols_to_drop = list(set(constant_cols + high_missing_cols + ide_cols +␣
↪date_cols))
d_filtered = d_filtered.drop(columns=cols_to_drop, errors='ignore')
```

```
Columns to drop:
Columns with >90% missing values: ['flag_sc', 'id_loan_rr', 'program_ind',
'rr_ind']
Columns with constant values: ['ppmt_pnlty', 'prod_type', 'flag_sc', 'rr_ind',
'io_ind']
Identifier columns: ['id_loan', 'seller_name', 'servicer_name']
Date columns: ['dt_first_pi', 'dt_matr']
```

1. 90% 2. 3. id 4. orig_loan_term

4.3 Train-Test Split

[45]:
```
# Split data before EDA to avoid data leakage
X = d_filtered.drop('loan_status', axis=1)
y = d_filtered['loan_status']

# Stratified split to maintain class balance
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    shuffle= True,
    test_size=0.2,
    stratify=y,  # Maintain class distribution
    random_state=42
)
```

EDA data leakage stratify=y, defualt week6 workshop

4.4 Missing Value Imputation

[46]:
```
# Numerical columns: Median imputation
num_cols = ['fico', 'mi_pct', 'cltv', 'dti', 'ltv']
num_imputer = SimpleImputer(strategy='median')
X_train[num_cols] = num_imputer.fit_transform(X_train[num_cols])
X_test[num_cols] = num_imputer.transform(X_test[num_cols])

# Ordinal categorical: Most frequent category
ord_cols = ['property_val']
ord_imputer = SimpleImputer(strategy='most_frequent')
X_train[ord_cols] = ord_imputer.fit_transform(X_train[ord_cols])
X_test[ord_cols] = ord_imputer.transform(X_test[ord_cols])

# Nominal categorical: Fill with 'Unknown'
cat_cols = ['cd_msa', 'mi_cancel_ind']
X_train[cat_cols] = X_train[cat_cols].fillna('Unknown')
X_test[cat_cols] = X_test[cat_cols].fillna('Unknown')
```

drop feature

4.5 Exploratory Data Analysis (EDA)

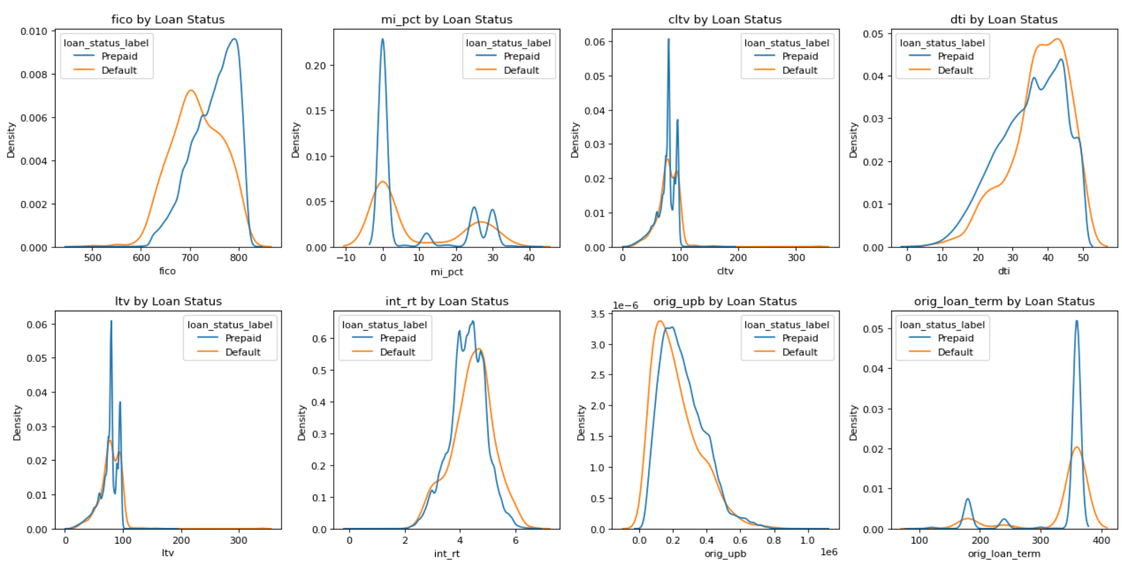
[47]:
```
# Combine training data for EDA
train_df = pd.concat([X_train, y_train], axis=1)
print("\nTraining data shape:", train_df.shape)
```

```
Training data shape: (101364, 21)
```

EDA

4.5.1 Numerical Features Analysis

[48]:
```
num_cols = ['fico', 'mi_pct', 'cltv', 'dti', 'ltv', 'int_rt', 'orig_upb', 'orig_loan_term'] # Numerical columns
n = 2 # Number of rows for subplots
m = len(num_cols) // n + len(num_cols) % n # Number of columns for subplots

# Numerical feature distribution comparison by loan status
train_df['loan_status_label'] = train_df['loan_status'].map({0: 'Prepaid', 1: 'Default'})

fig, axes = plt.subplots(n, m, figsize=(16, 8))
axes = axes.flatten()

for i, col in enumerate(num_cols):
    if col in train_df.columns:
        sns.kdeplot(data=train_df, x=col, hue='loan_status_label', ax=axes[i], common_norm=False)
        axes[i].set_title(f'{col} by Loan Status')
        axes[i].set_xlabel(col)
        axes[i].set_ylabel('Density')

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

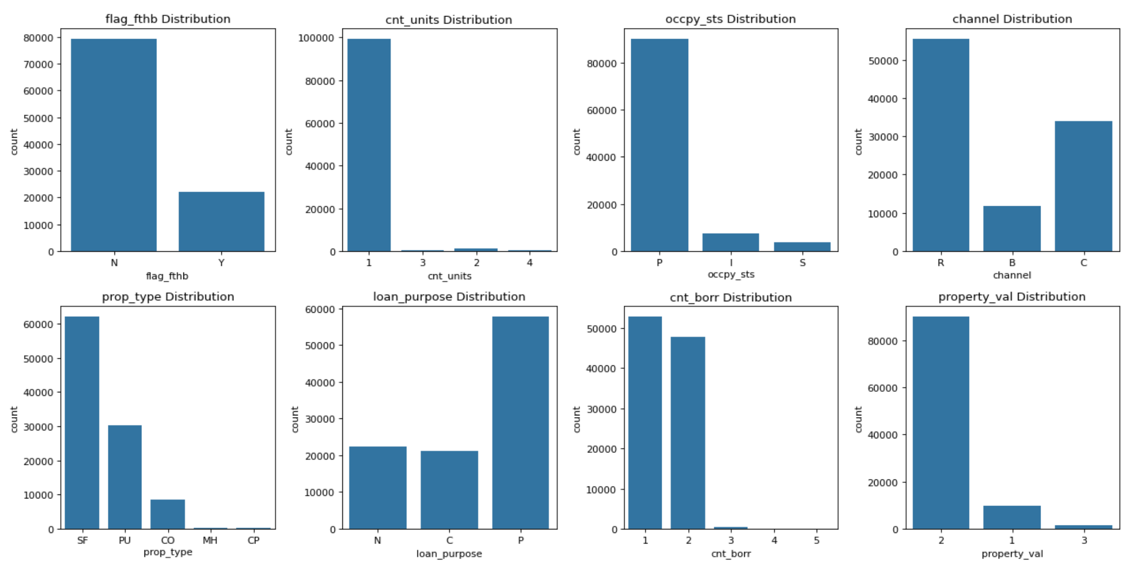


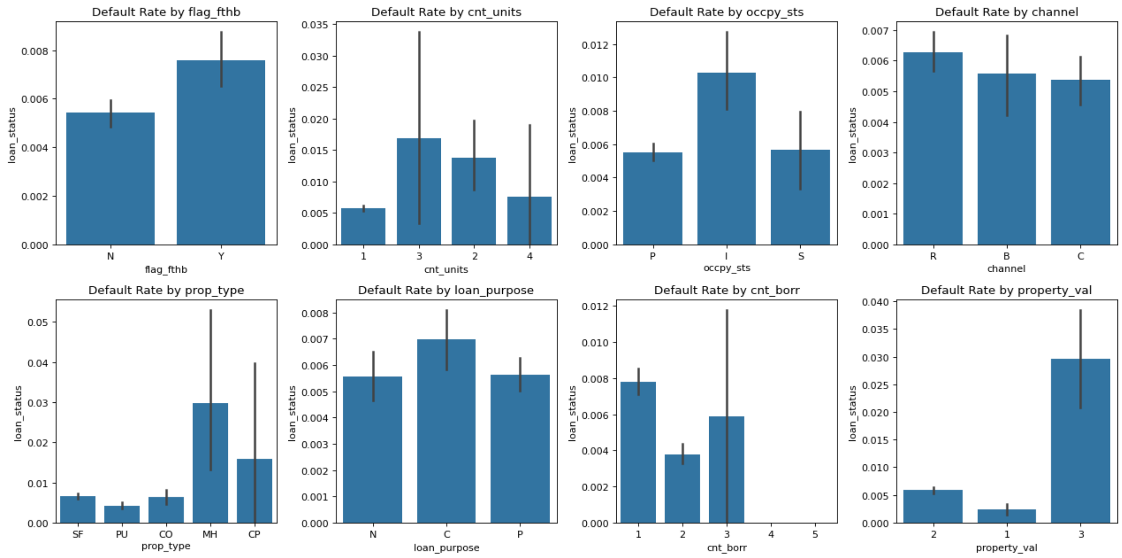
fico 700 800 prepaid fico

fico dti orig_upb FE

4.5.2 Categorical Features Analysis

[49]:
```python
cat_cols = ['flag_fthb', 'cnt_units', 'occpy_sts', 'channel', 'prop_type', 'loan_purpose', 'cnt_borr', 'property_val'] # Categorical columns
n = 2 # Number of rows for subplots
m = len(cat_cols) // n + len(cat_cols) % n # Number of columns for subplots
# Frequency plots
fig, axes = plt.subplots(n, m, figsize=(16, 8))
for i, col in enumerate(cat_cols):
    sns.countplot(x=col, data=train_df, ax=axes[i//m, i%m])
    axes[i//m, i%m].set_title(f'{col} Distribution')
    axes[i//m, i%m].tick_params(axis='x')
plt.tight_layout()
plt.show()

# Relationship with Target
fig, axes = plt.subplots(n, m, figsize=(16, 8))
for i, col in enumerate(cat_cols):
    sns.barplot(x=col, y='loan_status', data=train_df, ax=axes[i//m, i%m], estimator=np.mean)
    axes[i//m, i%m].set_title(f'Default Rate by {col}')
    axes[i//m, i%m].tick_params(axis='x')
plt.tight_layout()
plt.show()
```



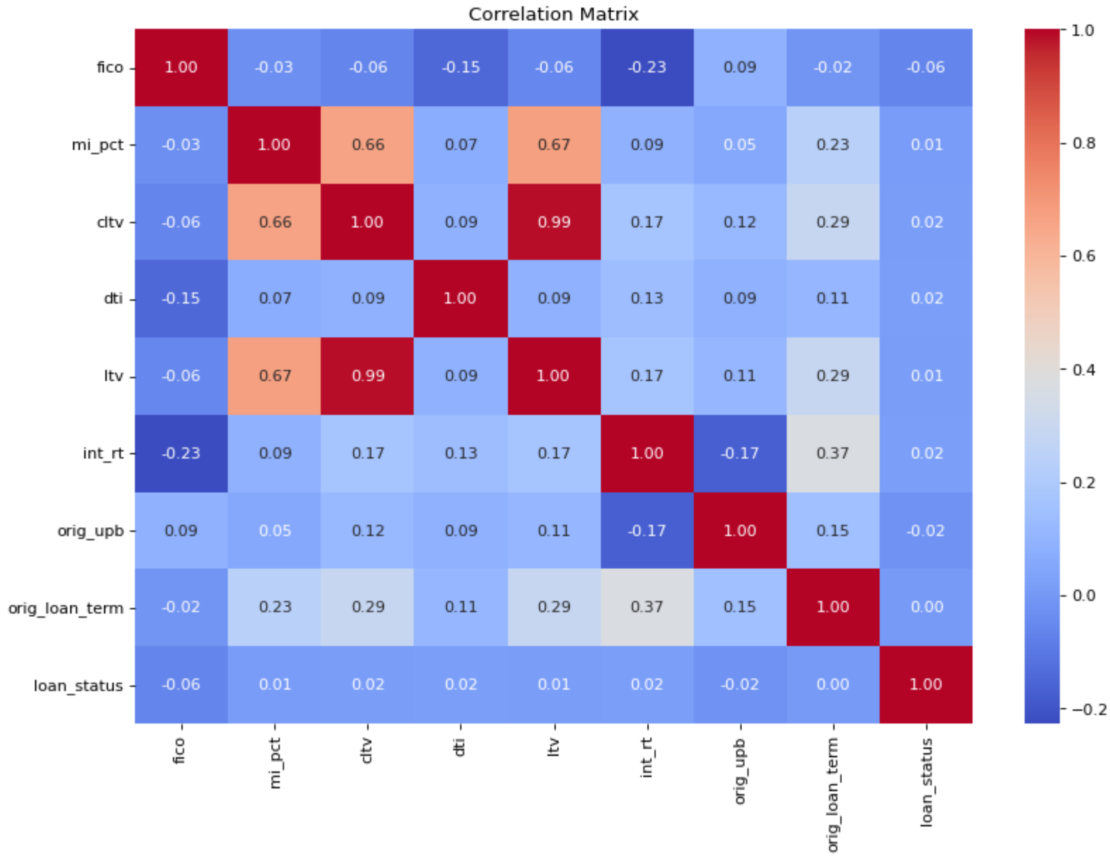


cat

occpy_sts I S P

4.5.3 Correlation Analysis

[50]:
```
corr_matrix = train_df[num_cols + ['loan_status']].corr()
plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



Dropping CLTV instead of LTV is a deliberate choice based on domain relevance and model interpretability. Here's the detailed reasoning:

1. Business Context: LTV vs. CLTV LTV (Loan-to-Value Ratio): Measures the primary mortgage amount relative to the property value. Example: A 200 k m o r t g a g e o n a 200kmortgageona250k home → LTV = 80%.

Industry Standard: LTV is the most widely used metric in mortgage underwriting and default prediction.

Regulatory Focus: Agencies like FHFA and Freddie Mac prioritize LTV in risk assessments.

CLTV (Combined Loan-to-Value Ratio): Includes all liens on the property (e.g., second mortgages, HELOCs). Example: A 200 k f i r s t m o r t g a g e + 200kfirstmortgage+50k HELOC on a $250k home → CLTV = 100%.

Redundancy: In your dataset, CLTV and LTV are nearly identical (r=0.99), meaning most loans likely have no secondary liens.

2. Statistical Reasons to Drop CLTV Multicollinearity: High correlation between CLTV and LTV can destabilize linear models (e.g., logistic regression) by inflating coefficient variances.

Feature Importance: In tree-based models (e.g., XGBoost), both features will compete for splits, diluting their individual importance.

Simpler Model: Dropping CLTV reduces dimensionality without losing predictive power (since LTV captures nearly the same information).

3. Practical Considerations Interpretability: LTV is more intuitive for stakeholders (e.g., “A 90% LTV loan is riskier than 80%”).

Data Quality: If CLTV has more missing values or inconsistencies (common in datasets where secondary liens are rare), retaining LTV is safer.

[51]:
```
X_train = X_train.drop(columns=['cltv'])
X_test = X_test.drop(columns=['cltv'])
```

[ ]:
```
# Compute VIF for features
vif_data = pd.DataFrame()
vif_data["feature"] = X_train[["mi_pct", "ltv"]].columns
vif_data["VIF"] = [variance_inflation_factor(X_train[["mi_pct", "ltv"]].values, i) for i in range(2)]
print(vif_data)
```

```
  feature      VIF
0  mi_pct  1.73093
1     ltv  1.73093
```

ltv cltv 0.99 drop drop cltv d

ltv mi_pct 0.67 VIF 5

4.5.4 EDA Findings

part FE

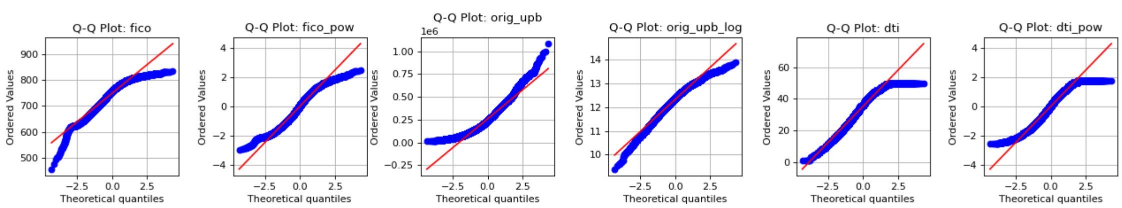
4.6 Feature Engineering

4.6.1 Transformations and Skewness Checking

[ ]:
```
# Apply log transformation on orig_upb to reduce right skewness
X_train['orig_upb_log'] = np.log1p(X_train['orig_upb'])
X_test['orig_upb_log'] = np.log1p(X_test['orig_upb'])

# Apply Yeo-Johnson transformation on fico to reduce left skewness
pt = PowerTransformer(method='yeo-johnson')
X_train['fico_pow'] = pt.fit_transform(X_train[['fico']])
X_test['fico_pow'] = pt.transform(X_test[['fico']])

# Apply Yeo-Johnson transformation on dti to reduce left skewness
X_train['dti_pow'] = pt.fit_transform(X_train[['dti']])
X_test['dti_pow'] = pt.transform(X_test[['dti']])
```

```python
# Check skewness of transformed columns
skewed_cols = ['fico', 'fico_pow', 'orig_upb', 'orig_upb_log', 'dti', 'dti_pow']
for col in skewed_cols:
    print(f"{col} skew: {skew(X_train[col]):.2f}")

# Plot Q-Q plots for skewed columns
n = 1
m = len(skewed_cols)//n + len(skewed_cols)%n
fig, axes = plt.subplots(n, m, figsize=(16, n*(16//m+1)))
axes = axes.ravel()  # Flatten axes for easy iteration
for i, col in enumerate(skewed_cols):
    probplot(X_train[col], plot=axes[i]) # Q-Q plot
    axes[i].set_title(f'Q-Q Plot: {col}')
    axes[i].grid(True)
plt.tight_layout()
plt.show()
```

```
fico skew: -0.65
fico_pow skew: -0.12
orig_upb skew: 0.78
orig_upb_log skew: -0.52
dti skew: -0.48
dti_pow skew: -0.16
```



fico dti orig_upb Yeo-Johnson log

QQ

dti dti_pow baseline roc XGBoost

4.6.2 Interaction Features

[56]:
```python
# Create categorical interaction: occupancy status + property type
# Captures combined effects (e.g., investment properties in condos)
X_train['occpy_prop'] =  X_train['occpy_sts'] + '_' + X_train['prop_type']
X_test['occpy_prop'] =  X_test['occpy_sts'] + '_' + X_test['prop_type']

# Create categorical interaction: loan purpose + number of borrowers
```

```
# Identifies purpose-specific borrower patterns (e.g., refinance with multiple␣
↪borrowers)
X_train['purpose_borr'] = X_train['loan_purpose'] + '_' + X_train['cnt_borr']
X_test['purpose_borr'] = X_test['loan_purpose'] + '_' + X_test['cnt_borr']

# Create numerical interaction: LTV * DTI
# Captures synergistic risk from high leverage and high debt burden
X_train['ltv_dti'] = X_train['ltv'] * X_train['dti']
X_test['ltv_dti'] = X_test['ltv'] * X_test['dti']
```

5 Model Fitting and Tuning

In this section you should detail and motivate your choice of model and describe the process used to refine, tune, and fit that model. You are encouraged to explore different models but you should NOT include a detailed narrative or code of all of these attempts. At most this section should briefly mention the methods explored and why they were rejected - most of your effort should go into describing the final model you are using and your process for tuning and validating it.

This section should include the full implementation of your final model, including all necessary validation. As with figures, any included code must also be addressed in the text of the document.

Finally, you should also provide a comparison of your model with a baseline model of your choice on the test data but only briefly describe the baseline model considered.

5.1 Baseline model

We selected Logistic Regression as the baseline due to its interpretability, speed, and ability to handle imbalanced classes when using class weighting. While more complex models may outperform it, Logistic Regression provides a strong, explainable benchmark.

baseline

Here's why certain features were excluded from the baseline model, despite 20 being available after EDA:

1. High Cardinality or Sparsity Features: cd_msa (MSA codes), zipcode, st (state), mi_cancel_ind

Reason:

cd_msa and zipcode have thousands of unique values. Encoding them as one-hot features would create high-dimensional, sparse data (e.g., 50+ dummy variables for states), increasing model complexity without clear benefits for a baseline.

mi_cancel_ind (mortgage insurance cancellation) had many missing or "Not Applicable" values after preprocessing, reducing its reliability.

2. Redundancy Feature: orig_loan_term (original loan term in months)

Reason:

The loan term is already indirectly captured by loan_age_months (age of the loan) and dt_first_pi/dt_matr (dates). Including both could introduce multicollinearity without adding unique predictive power.

3. Risk of Data Leakage Feature: property_val (property appraisal method)

Reason:

This variable might reflect post-origination actions (e.g., a property reappraisal after default). Using it could leak future information not available at loan origination, violating the model's real-world applicability.

4. Low Interpretability or Relevance Features: cnt_units (number of units), cnt_borr (number of borrowers)

Reason:

cnt_units (e.g., 1-unit vs. 4-unit properties) showed minimal correlation with default rates in EDA.

cnt_borr (number of borrowers) was excluded because it had low variance (e.g., 95% of loans had 1–2 borrowers).

5. Baseline Model Philosophy The baseline model prioritizes simplicity and interpretability over maximal predictive power. Including all 20 features would:

Complicate the model with marginal or noisy features (e.g., st, zipcode).

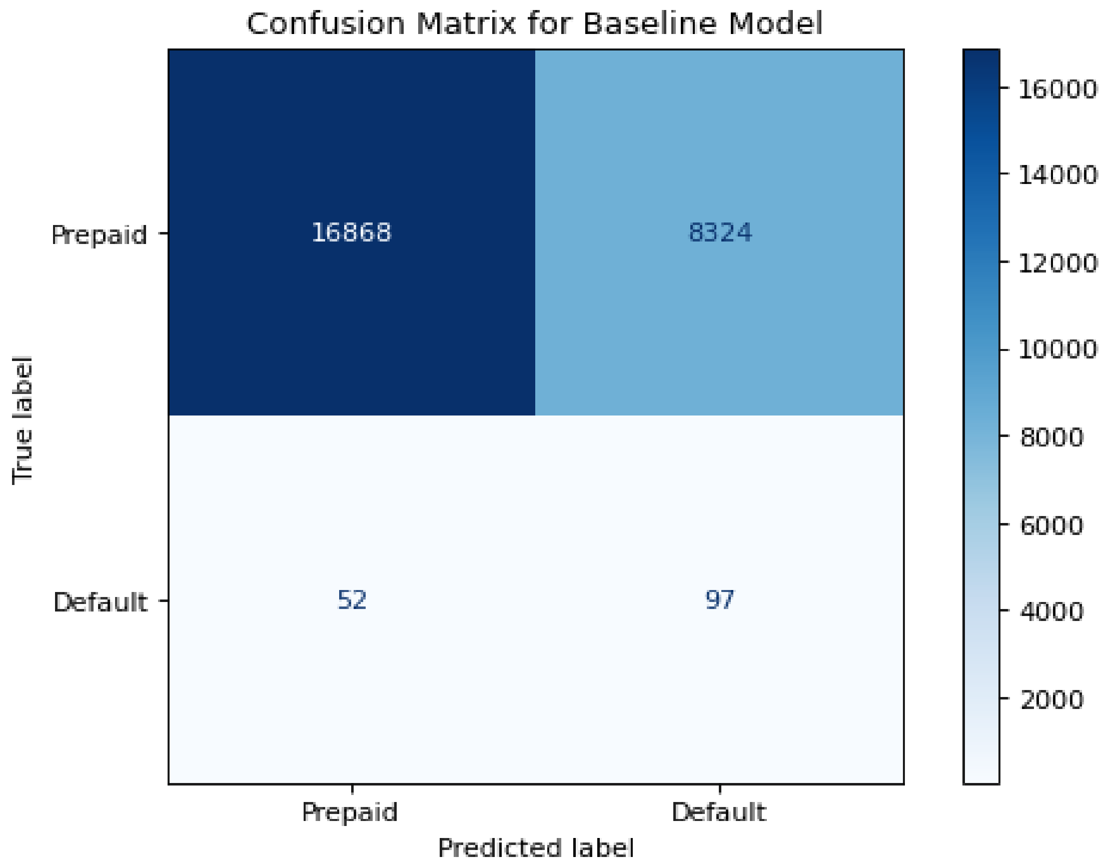
Reduce transparency, making it harder to explain coefficients to stakeholders.

Increase computational cost without guaranteeing better performance.

[ ]:
```
# Define categorical and numerical features
cat_cols = ['flag_fthb', 'occpy_sts', 'channel', 'loan_purpose', 'cnt_borr',␣
 ↪'occpy_prop', 'prop_type']
num_cols = ['fico_pow', 'mi_pct', 'ltv', 'dti_pow', 'int_rt', 'orig_upb_log',␣
 ↪'orig_loan_term', 'ltv_dti']

# Create a preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)
    ]
)

# Combine preprocessing and model into a pipeline
baseline_model = ImPipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(
        class_weight='balanced',  # Adjusts weights for imbalance
```

```
        max_iter=1000,
        random_state=42
    ))
])

# Train the model
baseline_model.fit(X_train, y_train)

# Predict on test data
y_pred = baseline_model.predict(X_test)
y_proba = baseline_model.predict_proba(X_test)[:, 1]  # Probabilities for␣
↪default

print(classification_report(y_test, y_pred))
# Evaluate performance
print("Baseline Model Performance:")
print(f"- Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"- Precision: {precision_score(y_test, y_pred):.4f}")
print(f"- Recall: {recall_score(y_test, y_pred):.4f}")
print(f"- F1 Score: {f1_score(y_test, y_pred):.4f}")
print(f"- ROC-AUC: {roc_auc_score(y_test, y_proba):.4f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Prepaid',␣
↪'Default'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix for Baseline Model')
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      0.67      0.80     25192
           1       0.01      0.65      0.02       149

    accuracy                           0.67     25341
   macro avg       0.50      0.66      0.41     25341
weighted avg       0.99      0.67      0.80     25341

Baseline Model Performance:
- Accuracy: 0.6695
- Precision: 0.0115
- Recall: 0.6510
- F1 Score: 0.0226
- ROC-AUC: 0.7176
```

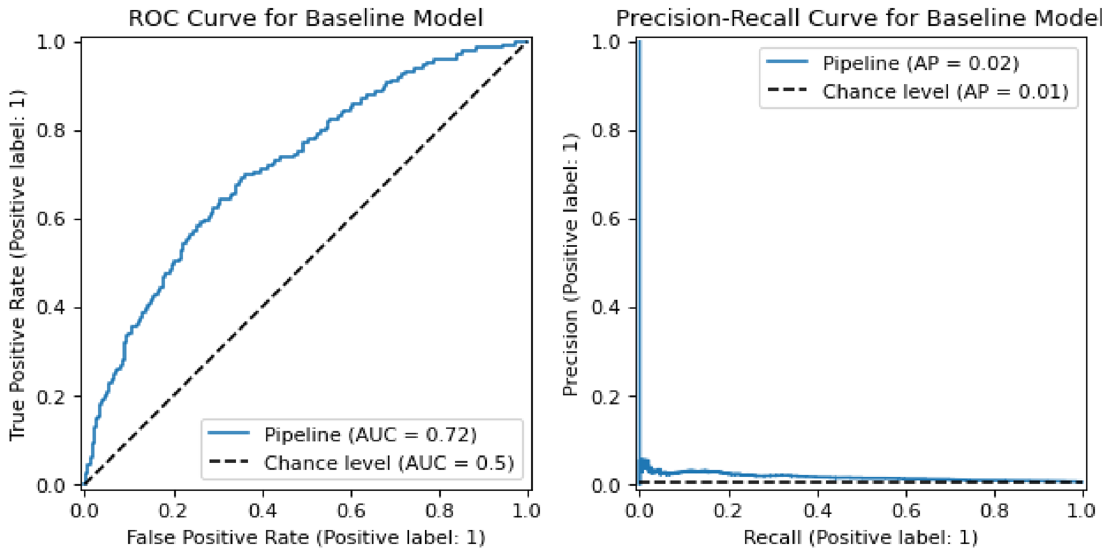


week6 workshop

class_weight='balanced' Over-sampling Under-sampling week6 workshop

[71]:
```
fig, axes = plt.subplots(1, 2)

# ROC Curve
RocCurveDisplay.from_estimator(baseline_model, X_test, y_test, ax=axes[0],␣
    ↪plot_chance_level=True)
axes[0].set_title('ROC Curve for Baseline Model')

# Precision-Recall Curve
PrecisionRecallDisplay.from_estimator(baseline_model, X_test, y_test,␣
    ↪ax=axes[1], plot_chance_level=True)
axes[1].set_title('Precision-Recall Curve for Baseline Model')
axes[1].legend(loc='upper right')

plt.tight_layout()
```

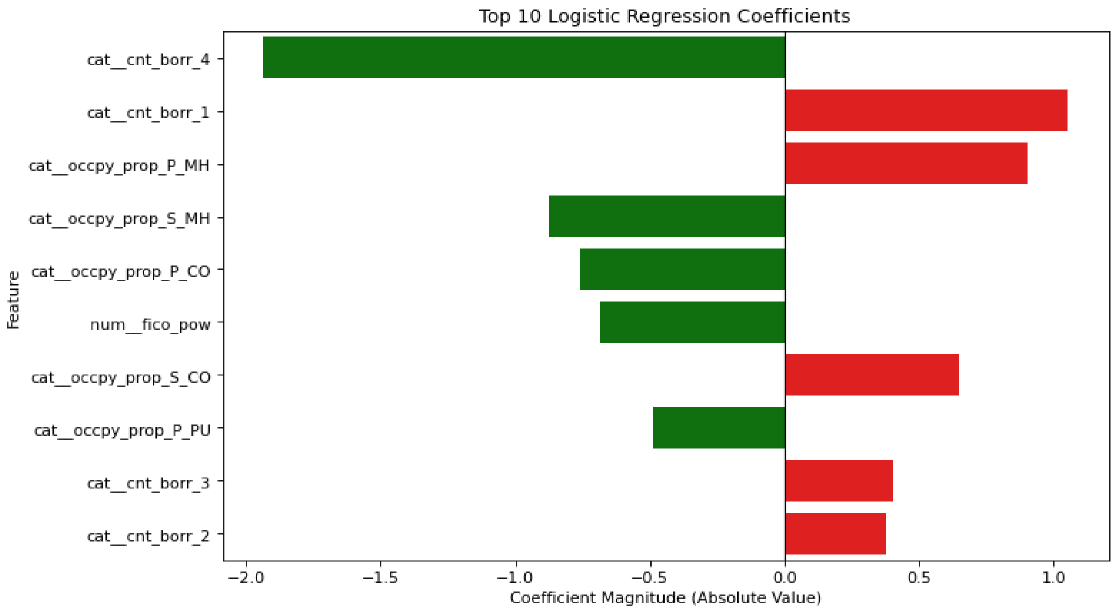
```
plt.show()
```



week6 workshop

[72]:
```
# Get feature names from the preprocessor
feature_names = baseline_model.named_steps['preprocessor'].
↪get_feature_names_out()

# Extract coefficients from logistic regression
coefficients = baseline_model.named_steps['classifier'].coef_[0]

# Create a DataFrame for interpretation
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients,
    'Abs_Coefficient': np.abs(coefficients)
}).sort_values(by='Abs_Coefficient', ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(
    x='Coefficient',
    y='Feature',
    hue='Feature',
    data=coef_df.head(10),
    palette=['Red' if x > 0 else 'Green' for x in coef_df.
↪head(10)['Coefficient']]
    )
```

```
plt.axvline(0, color='black', linestyle='-', linewidth=1)
plt.title('Top 10 Logistic Regression Coefficients')
plt.xlabel('Coefficient Magnitude (Absolute Value)')
plt.show()
```



5.2 XGBoost

[73]:
```
# Define full categorical and numerical features
cat_cols = ['flag_fthb', 'occpy_sts', 'channel', 'prop_type', 'loan_purpose',␣
↪'mi_cancel_ind', 'cnt_borr',
            'cnt_units', 'property_val', 'cd_msa', 'zipcode', 'st',␣
↪'occpy_prop', 'purpose_borr'
]

num_cols = ['fico', 'fico_pow', 'mi_pct', 'dti', 'dti_pow', 'ltv', 'int_rt',
    'orig_upb', 'orig_upb_log', 'orig_loan_term', 'ltv_dti'
]

# Preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)
    ],
```

```
    remainder='passthrough',  # Pass numerical features unchanged
)

# Calculate scale_pos_weight for XGBoost
neg = len(y_train[y_train == 0])
pos = len(y_train[y_train == 1])
scale_pos_weight = neg / pos
print(f"scale_pos_weight: {scale_pos_weight:.2f}")
```

```
scale_pos_weight: 168.79
```

FE

cat target encoder onehot

scale_pos_weight

[75]:
```
'''
# Define parameter grid
param_grid = {
    'classifier__subsample': [0.95, 1.0], # [0.5, 1.0]
    'classifier__n_estimators': [175, 200, 225, 250], # [50, 2000]
    'classifier__max_depth': [2, 3, 4],  # [1, 20]
    'classifier__learning_rate': [0.075, 0.1, 0.125, 0.15], # [0.01, 0.3]
    'classifier__colsample_bytree': [0.9, 0.95, 1.0] # [0.5, 1.0]
    }

# Create pipeline
xgb_model = ImPipeline([
    ('preprocessor', preprocessor),
    ('smote', SMOTE(sampling_strategy=0.3, random_state=42)),  # Optional: add␣
 ↪SMOTE for oversampling
    ('classifier', XGBClassifier(
        objective='binary:logistic',
        scale_pos_weight=scale_pos_weight,  # Adjust for class imbalance
        random_state=42,
        eval_metric='auc'
    ))
])

# RandomizedSearchCV with stratified K-fold
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
search = RandomizedSearchCV(
    xgb_model,
    param_grid,
    n_iter=20,
    scoring='roc_auc', # Aim for AUC
    cv=cv,
    n_jobs=-1
```

```
)

# Fit model
search.fit(X_train, y_train)
# Get best parameters without the pipeline prefix
best_params = {k.replace('classifier__', ''): v for k, v in search.best_params_.items()}
'''
#
best_params = {
    'subsample': 0.95,
    'n_estimators': 200,
    'max_depth': 2,
    'learning_rate': 0.15,
    'colsample_bytree': 1.0
}

# Best parameters
print("Best Parameters:", best_params)
```

```
Best Parameters: {'subsample': 0.95, 'n_estimators': 200, 'max_depth': 2,
'learning_rate': 0.15, 'colsample_bytree': 1.0}
```

RandomizedSearchCV roc_auc

SMOTE

gamma gamma

[76]:
```
# Initialize final model with tuned parameters
final_model = ImPipeline([
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(
        **best_params,
        objective='binary:logistic',
        scale_pos_weight=scale_pos_weight,
        random_state=42
    ))
])

# Train
final_model.fit(X_train, y_train)
print("Final Model Trained")
```

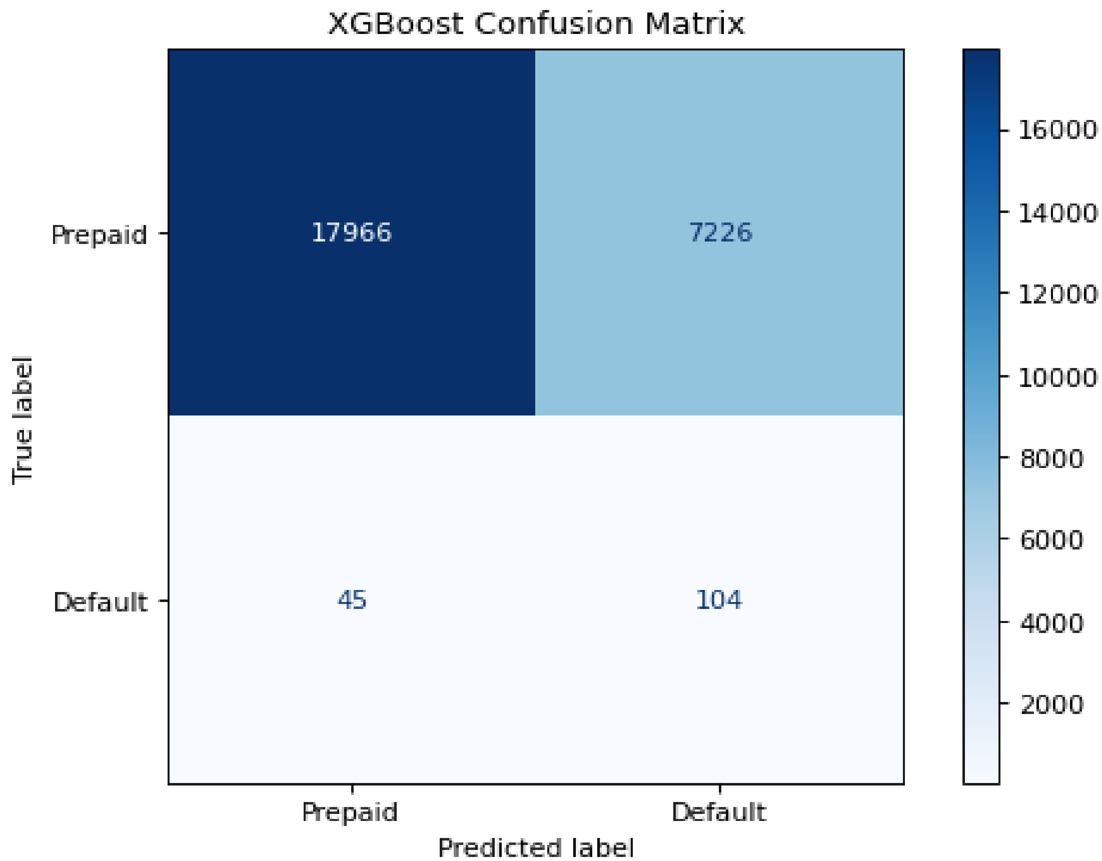
```
Final Model Trained
```

SMOTE GPT

[77]:

```python
# Predictions
y_pred_xgb = final_model.predict(X_test)
y_proba_xgb = final_model.predict_proba(X_test)[:, 1]

threshold = 0.45  # Adjust based on PR curve analysis
y_pred_xgb = (y_proba_xgb >= threshold).astype(int)
print(classification_report(y_test, y_pred_xgb))
# Performance metrics
print("XGBoost Performance:")
print(f"- Accuracy: {accuracy_score(y_test, y_pred_xgb):.4f}")
print(f"- Precision: {precision_score(y_test, y_pred_xgb):.4f}")
print(f"- Recall: {recall_score(y_test, y_pred_xgb):.4f}")
print(f"- F1 Score: {f1_score(y_test, y_pred_xgb):.4f}")
print(f"- ROC-AUC: {roc_auc_score(y_test, y_proba_xgb):.4f}")

# Confusion Matrix
cm_xgb = confusion_matrix(y_test, y_pred_xgb)
disp_xgb = ConfusionMatrixDisplay(confusion_matrix=cm_xgb,␣
 ↪display_labels=['Prepaid', 'Default'])
disp_xgb.plot(cmap='Blues')
plt.title('XGBoost Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      0.71      0.83     25192
           1       0.01      0.70      0.03       149

    accuracy                           0.71     25341
   macro avg       0.51      0.71      0.43     25341
weighted avg       0.99      0.71      0.83     25341

XGBoost Performance:
- Accuracy: 0.7131
- Precision: 0.0142
- Recall: 0.6980
- F1 Score: 0.0278
- ROC-AUC: 0.7656
```

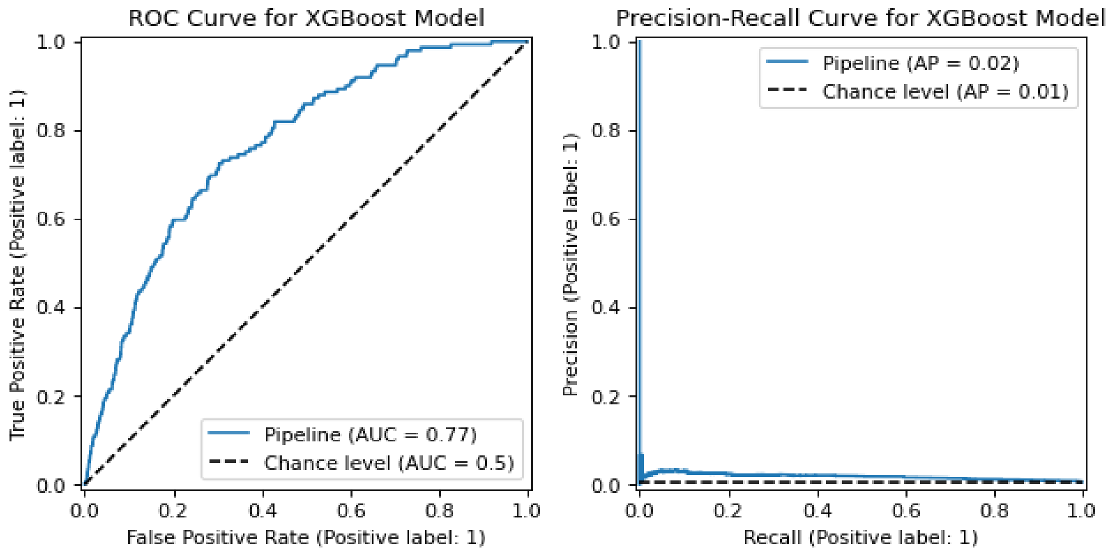


threshold cross threshold defualt

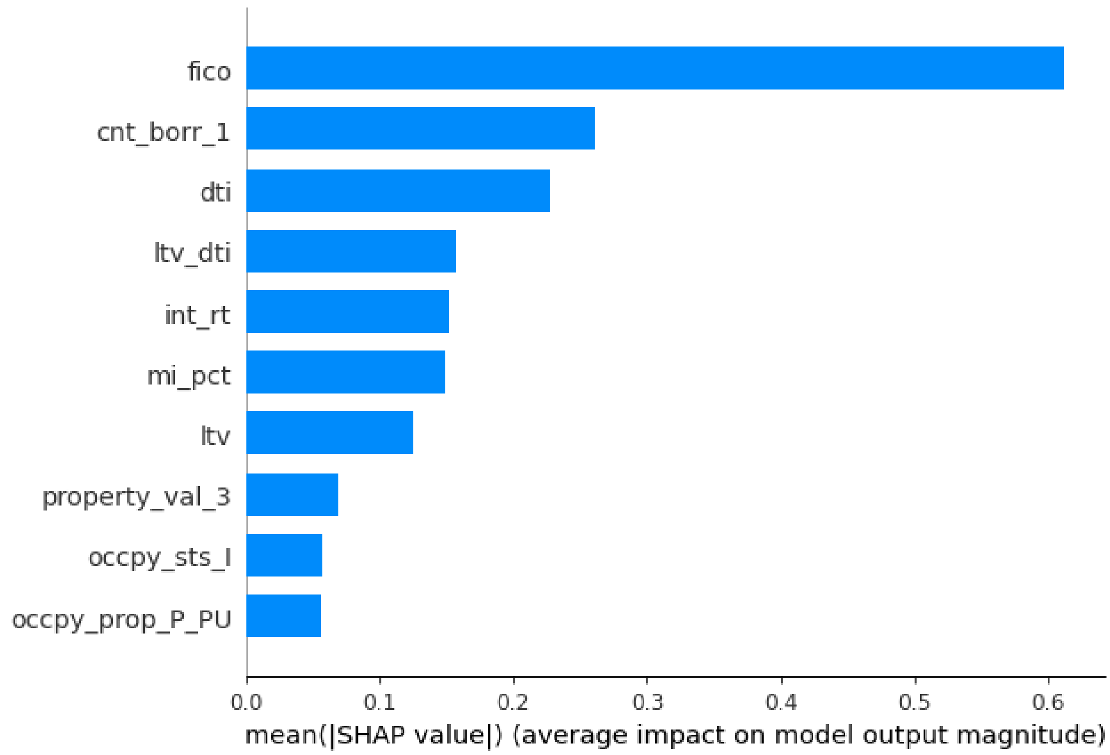
threshold

[64]:
```
fig, axes = plt.subplots(1, 2)

# ROC Curve
RocCurveDisplay.from_estimator(final_model, X_test, y_test, ax=axes[0],␣
 ↪plot_chance_level=True)
axes[0].set_title('ROC Curve for XGBoost Model')

# Precision-Recall Curve
PrecisionRecallDisplay.from_estimator(final_model, X_test, y_test, ax=axes[1],␣
 ↪plot_chance_level=True)
axes[1].set_title('Precision-Recall Curve for XGBoost Model')
axes[1].legend(loc='upper right')

plt.tight_layout()
```

```
plt.show()
```



[65]:
```
# Extract feature names after one-hot encoding
encoder = final_model.named_steps['preprocessor'].named_transformers_['cat']
cat_features = encoder.get_feature_names_out(cat_cols)
all_features = np.concatenate([cat_features, num_cols])

# Get preprocessed data
X_processed = final_model.named_steps['preprocessor'].transform(X_train)

# SHAP summary plot
explainer = shap.TreeExplainer(final_model.named_steps['classifier'])
shap_values = explainer.shap_values(X_processed)
shap.summary_plot(shap_values, X_processed, feature_names=all_features,␣
↪plot_type='bar', max_display=10)
```



SHAP SHAP workshop week11

shap value value

[68]:
```
print("Baseline vs XGBoost Comparison:")
print(f"Metric\t\tBaseline\tXGBoost")
print(f"Accuracy\t{accuracy_score(y_test, y_pred):.
  ↪4f}\t\t{accuracy_score(y_test, y_pred_xgb):.4f}")
print(f"Precision\t{precision_score(y_test, y_pred):.
  ↪4f}\t\t{precision_score(y_test, y_pred_xgb):.4f}")
print(f"Recall\t\t{recall_score(y_test, y_pred):.4f}\t\t{recall_score(y_test,␣
  ↪y_pred_xgb):.4f}")
print(f"F1 Score\t{f1_score(y_test, y_pred):.4f}\t\t{f1_score(y_test,␣
  ↪y_pred_xgb):.4f}")
print(f"ROC-AUC\t\t{roc_auc_score(y_test, y_proba):.
  ↪4f}\t\t{roc_auc_score(y_test, y_proba_xgb):.4f}")
```

```
Baseline vs XGBoost Comparison:
Metric          Baseline        XGBoost
Accuracy        0.6697          0.7131
Precision       0.0115          0.0142
Recall          0.6510          0.6980
```

```
F1 Score         0.0227          0.0278
ROC-AUC          0.7166          0.7656
```

6 Discussion & Conclusions

In this section you should provide a general overview of your final model, its performance, and reliability. You should discuss what the implications of your model are in terms of the included features, estimated parameters and relationships, predictive performance, and anything else you think is relevant.

This should be written with a target audience of a banking official, who is understands the issues associated with mortgage defaults but may only have university level mathematics (not necessarily postgraduate statistics or machine learning). Your goal should be to highlight to this audience how your model can useful. You should also discuss potential limitations or directions of future improvement of your model.

Finally, you should include recommendations on factors that may increase the risk of default, which may be useful for the companies to improve their understanding of mortgage defaults, and also to explain their decisions to clients and regulatory bodies. You should also use your model to inform the company of any active loans that are at risk of default.

Keep in mind that a negative result, i.e. a model that does not work well predictively, that is well explained and justified in terms of why it failed will likely receive higher marks than a model with strong predictive performance but with poor or incorrect explinations / justifications.

7 Generative AI statement

Include a statement on how generative AI was used in the project and report.

8 References

Include references if any

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