Kaggle Playground

Problem Statement / Real World Implementations

In [1]:
```python
# --- 1. Importing Libraries ---
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from scipy.stats import rankdata

# Preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score

# Models
import lightgbm as lgb
import xgboost as xgb
from catboost import CatBoostClassifier

# Notebook settings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
```

In [2]:
```python
# --- 2. Configuration ---
class Config:
    """Configuration class for hyperparameters and settings"""
    N_SPLITS = 5
    SEED = 42
    TARGET = 'loan_paid_back'

config = Config()

print("Version 11: 3-Model CV Ensemble (LGBM+XGB+CAT) with Advanced FE")
print(f"--- 1. Loading Data ---")
```

```
Version 11: 3-Model CV Ensemble (LGBM+XGB+CAT) with Advanced FE
--- 1. Loading Data ---
```

In [3]:
```python
# Define file paths
TRAIN_PATH = "/kaggle/input/playground-series-s5e11/train.csv"
TEST_PATH = "/kaggle/input/playground-series-s5e11/test.csv"
SUBMISSION_PATH = "/kaggle/input/playground-series-s5e11/sample_submission.csv"

# Load the datasets
train = pd.read_csv(TRAIN_PATH)
test = pd.read_csv(TEST_PATH)
sample_submission = pd.read_csv(SUBMISSION_PATH)
```

```
print(f"Train shape: {train.shape}, Test shape: {test.shape}")
print("--- 2. Defining Preprocessing & Feature Engineering ---")
```

```
Train shape: (593994, 13), Test shape: (254569, 12)
--- 2. Defining Preprocessing & Feature Engineering ---
```

In [4]:
```
def complete_feature_engineering(df):
    """
    Comprehensive feature engineering pipeline for loan prediction
    """
    df = df.copy()

    # 1. FINANCIAL RATIOS
    df['loan_to_income_ratio'] = df['loan_amount'] / (df['annual_income'] + 1)
    df['monthly_income'] = df['annual_income'] / 12
    # Simplified approximation from source
    df['monthly_payment_estimate'] = (df['loan_amount'] * df['interest_rate']) / 10
    df['payment_to_income_ratio'] = df['monthly_payment_estimate'] / (df['monthly_i
    df['current_debt_amount'] = df['debt_to_income_ratio'] * df['annual_income']
    df['total_debt_with_loan'] = df['current_debt_amount'] + df['loan_amount']
    df['new_debt_to_income'] = df['total_debt_with_loan'] / (df['annual_income'] +
    df['debt_increase_ratio'] = df['new_debt_to_income'] / (df['debt_to_income_rati
    df['disposable_income'] = df['annual_income'] - df['current_debt_amount']
    df['disposable_income_ratio'] = df['disposable_income'] / (df['annual_income']
    df['loan_to_disposable_income'] = df['loan_amount'] / (df['disposable_income']
    df['monthly_disposable_income'] = df['disposable_income'] / 12
    df['payment_to_disposable_ratio'] = df['monthly_payment_estimate'] / (df['month
    df['annual_payment_burden'] = df['monthly_payment_estimate'] * 12
    df['payment_burden_ratio'] = df['annual_payment_burden'] / (df['annual_income']

    # 2. CREDIT SCORE FEATURES
    df['credit_score_normalized'] = df['credit_score'] / 850
    df['credit_risk_score'] = 1 - df['credit_score_normalized']
    df['credit_score_squared'] = df['credit_score'] ** 2
    df['credit_score_log'] = np.log1p(df['credit_score'])
    df['credit_category'] = pd.cut(df['credit_score'], bins=[0, 580, 670, 740, 800,
                                   labels=['poor', 'fair', 'good', 'very_good', 'exce
    df['credit_income_interaction'] = df['credit_score'] * df['annual_income']
    df['credit_times_dti'] = df['credit_score'] * df['debt_to_income_ratio']
    df['credit_loan_interaction'] = df['credit_score'] * df['loan_amount']

    # 3. INTEREST RATE FEATURES
    df['high_interest_flag'] = (df['interest_rate'] > df['interest_rate'].median())
    df['very_high_interest'] = (df['interest_rate'] > df['interest_rate'].quantile(
    df['low_interest_flag'] = (df['interest_rate'] < df['interest_rate'].quantile(0
    df['total_interest_cost'] = df['loan_amount'] * df['interest_rate'] / 100
    df['interest_burden'] = df['total_interest_cost'] / (df['annual_income'] + 1)
    df['interest_credit_mismatch'] = df['interest_rate'] * (1 - df['credit_score_no
    df['interest_credit_ratio'] = df['interest_rate'] / (df['credit_score'] / 100)
    df['interest_rate_squared'] = df['interest_rate'] ** 2

    # 4. RISK SCORES
    df['risk_score_v1'] = (df['debt_to_income_ratio'] * 0.25 + df['loan_to_income_r
                           df['credit_risk_score'] * 0.30 + (df['interest_rate'] /
    df['risk_score_v2'] = (df['payment_to_income_ratio'] * 0.40 + df['new_debt_to_i
                           df['interest_burden'] * 0.25)
```

```python
    df['affordability_score'] = (df['credit_score_normalized'] * 0.40 +
                                 (1 - df['debt_to_income_ratio']) * 0.30 +
                                 df['disposable_income_ratio'] * 0.30)
    df['financial_health_score'] = df['affordability_score'] * 0.60 - df['risk_scor

    # 5. LOAN AMOUNT FEATURES
    df['loan_size'] = pd.cut(df['loan_amount'], bins=[0, 10000, 20000, 30000, np.in
                             labels=['small', 'medium', 'large', 'very_large'])
    df['loan_amount_squared'] = df['loan_amount'] ** 2
    df['loan_amount_log'] = np.log1p(df['loan_amount'])
    df['annual_income_log'] = np.log1p(df['annual_income'])
    df['loan_amount_sqrt'] = np.sqrt(df['loan_amount'])

    # 6. BINNING FEATURES
    df['income_decile'] = pd.qcut(df['annual_income'], q=10, labels=False, duplicat
    df['credit_decile'] = pd.qcut(df['credit_score'], q=10, labels=False, duplicate
    df['loan_decile'] = pd.qcut(df['loan_amount'], q=10, labels=False, duplicates='
    df['dti_decile'] = pd.qcut(df['debt_to_income_ratio'], q=10, labels=False, dupl
    df['interest_decile'] = pd.qcut(df['interest_rate'], q=10, labels=False, duplic

    # 7. INTERACTION FEATURES
    df['income_x_credit'] = df['annual_income'] * df['credit_score']
    df['dti_x_interest'] = df['debt_to_income_ratio'] * df['interest_rate']
    df['loan_x_interest'] = df['loan_amount'] * df['interest_rate']
    df['income_x_dti'] = df['annual_income'] * df['debt_to_income_ratio']
    df['income_credit_loan'] = (df['annual_income'] * df['credit_score']) / (df['lo
    df['dti_interest_credit'] = (df['debt_to_income_ratio'] * df['interest_rate'])

    # 8. GRADE FEATURES
    df['grade'] = df['grade_subgrade'].str[0]
    df['subgrade_num'] = pd.to_numeric(df['grade_subgrade'].str[1:], errors='coerce
    grade_map = {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7}
    df['grade_numeric'] = df['grade'].map(grade_map)
    df['full_grade_score'] = df['grade_numeric'] * 10 + df['subgrade_num']
    df['grade_credit_ratio'] = df['full_grade_score'] / (df['credit_score'] / 100)

    # 9. STATISTICAL AGGREGATIONS
    financial_metrics = ['debt_to_income_ratio', 'loan_to_income_ratio', 'payment_t
    df['mean_financial_metrics'] = df[financial_metrics].mean(axis=1)
    df['max_financial_burden'] = df[financial_metrics].max(axis=1)
    df['min_financial_burden'] = df[financial_metrics].min(axis=1)
    df['std_financial_metrics'] = df[financial_metrics].std(axis=1)

    # 10. CATEGORICAL COMBINATIONS
    df['gender_marital'] = df['gender'] + '_' + df['marital_status']
    df['education_employment'] = df['education_level'] + '_' + df['employment_statu
    df['gender_education'] = df['gender'] + '_' + df['education_level']
    df['marital_employment'] = df['marital_status'] + '_' + df['employment_status']
    df['purpose_grade'] = df['loan_purpose'] + '_' + df['grade']
    df['employment_purpose'] = df['employment_status'] + '_' + df['loan_purpose']

    # 11. ANOMALY FLAGS
    df['extreme_dti'] = (df['debt_to_income_ratio'] > df['debt_to_income_ratio'].qu
    df['low_income'] = (df['annual_income'] < df['annual_income'].quantile(0.25)).a
    df['large_loan'] = (df['loan_amount'] > df['loan_amount'].quantile(0.75)).astyp
    df['risky_combo_1'] = ((df['debt_to_income_ratio'] > 0.4) & (df['credit_score']
    df['risky_combo_2'] = ((df['loan_to_income_ratio'] > 0.5) & (df['interest_rate'
    df['safe_combo'] = ((df['credit_score'] > 750) & (df['debt_to_income_ratio'] <
    df['high_risk_all'] = (df['extreme_dti'] & df['risky_combo_1']).astype(int)

    return df
```

In [5]:
```
print("--- 3. Applying Feature Engineering & Encoding ---")

# Apply feature engineering
train_fe = complete_feature_engineering(train)
test_fe = complete_feature_engineering(test)

# Encode categorical features
print("\nENCODING CATEGORICAL FEATURES")
categorical_features = train_fe.select_dtypes(include=['object', 'category']).colum
le_dict = {}
for col in categorical_features:
    le = LabelEncoder()
    # Combine train and test for a full fit, ensuring all categories are known
    all_values = pd.concat([train_fe[col].astype(str), test_fe[col].astype(str)])
    le.fit(all_values)

    train_fe[col] = le.transform(train_fe[col].astype(str))
    test_fe[col] = le.transform(test_fe[col].astype(str))
    le_dict[col] = le
    print(f"✓ {col}: {len(le.classes_)} classes")

# Prepare final datasets
print("\nFINAL DATA READY")
feature_cols = [col for col in train_fe.columns if col not in ['id', config.TARGET]

# Align columns - crucial if FE created different columns
train_cols = set(train_fe.columns)
test_cols = set(test_fe.columns)

missing_in_test = list(train_cols - test_cols - {'id', config.TARGET})
for col in missing_in_test:
    if col in feature_cols:
        test_fe[col] = 0

missing_in_train = list(test_cols - train_cols - {'id'})
for col in missing_in_train:
    if col in feature_cols:
        train_fe[col] = 0

# Ensure final feature list is identical
feature_cols = [col for col in feature_cols if col in test_fe.columns]
X = train_fe[feature_cols]
y = train_fe[config.TARGET]
X_test = test_fe[feature_cols]
test_ids = test_fe['id']

# Fill any remaining NaNs from FE (e.g., from ratios)
X = X.fillna(-1)
X_test = X_test.fillna(-1)
```

```
print(f"X: {X.shape}")
print(f"y: {y.shape}")
print(f"X_test: {X_test.shape}")
print(f"Features: {len(feature_cols)}")
print("\n--- 4. Model Training (LGBM) ---")
```

```
--- 3. Applying Feature Engineering & Encoding ---

ENCODING CATEGORICAL FEATURES
✓ gender: 3 classes
✓ marital_status: 4 classes
✓ education_level: 5 classes
✓ employment_status: 5 classes
✓ loan_purpose: 8 classes
✓ grade_subgrade: 30 classes
✓ credit_category: 5 classes
✓ loan_size: 4 classes
✓ grade: 6 classes
✓ gender_marital: 12 classes
✓ education_employment: 25 classes
✓ gender_education: 15 classes
✓ marital_employment: 20 classes
✓ purpose_grade: 48 classes
✓ employment_purpose: 40 classes

FINAL DATA READY
X: (593994, 84)
y: (593994,)
X_test: (254569, 84)
Features: 84

--- 4. Model Training (LGBM) ---
```

In [6]:

```
def train_lightgbm(X, y, X_test, n_splits=5):
    """ Trains LightGBM model using StratifiedKFold """
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=config.SEED
    oof_preds = np.zeros(len(X))
    test_preds = np.zeros(len(X_test))
    feature_importance = pd.DataFrame()

    # Parameters from source notebook
    params = {
        'objective': 'binary',
        'metric': 'auc',
        'boosting_type': 'gbdt',
        'num_leaves': 31,
        'learning_rate': 0.05,
        'feature_fraction': 0.8,
        'bagging_fraction': 0.8,
        'bagging_freq': 5,
        'max_depth': -1,
        'min_child_samples': 20,
        'reg_alpha': 0.1,
        'reg_lambda': 0.1,
        'random_state': config.SEED,
        'verbose': -1,
        'n_jobs': -1
    }

    fold_scores = []

    for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
        print(f"\n--- Fold {fold + 1}/{n_splits} ---")
        X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

        train_data = lgb.Dataset(X_train, label=y_train)
        val_data = lgb.Dataset(X_val, label=y_val, reference=train_data)

        model = lgb.train(
            params,
            train_data,
            num_boost_round=2000,
            valid_sets=[train_data, val_data],
            callbacks=[lgb.early_stopping(100), lgb.log_evaluation(200)]
        )

        val_preds = model.predict(X_val, num_iteration=model.best_iteration)
        oof_preds[val_idx] = val_preds
        test_preds += model.predict(X_test, num_iteration=model.best_iteration) / n

        score = roc_auc_score(y_val, val_preds)
        fold_scores.append(score)
        print(f"Fold {fold + 1} AUC: {score:.6f}")

        fold_importance_df = pd.DataFrame()
        fold_importance_df["feature"] = X.columns
        fold_importance_df["importance"] = model.feature_importance(importance_type
        fold_importance_df["fold"] = fold + 1
        feature_importance = pd.concat([feature_importance, fold_importance_df], ax

    overall_score = roc_auc_score(y, oof_preds)
    print(f"\nLightGBM OOF AUC: {overall_score:.6f}")
    print(f"Mean: {np.mean(fold_scores):.6f} (+/- {np.std(fold_scores):.6f})")
    return oof_preds, test_preds, feature_importance, overall_score
```

In [7]:
```
print("\nTraining LightGBM...")
lgb_oof, lgb_test, lgb_importance, lgb_score = train_lightgbm(X, y, X_test, n_split
print("\n--- 5. Model Training (XGBoost) ---")
```

```
Training LightGBM...

--- Fold 1/5 ---
Training until validation scores don't improve for 100 rounds
[200]   training's auc: 0.922442        valid_1's auc: 0.920568
[400]   training's auc: 0.927888        valid_1's auc: 0.921743
[600]   training's auc: 0.932566        valid_1's auc: 0.922435
[800]   training's auc: 0.936837        valid_1's auc: 0.922586
[1000]  training's auc: 0.940291        valid_1's auc: 0.922635
Early stopping, best iteration is:
[938]   training's auc: 0.939342        valid_1's auc: 0.922691
Fold 1 AUC: 0.922691

--- Fold 2/5 ---
Training until validation scores don't improve for 100 rounds
[200]   training's auc: 0.922359        valid_1's auc: 0.91966
[400]   training's auc: 0.928167        valid_1's auc: 0.921101
[600]   training's auc: 0.932599        valid_1's auc: 0.921701
Early stopping, best iteration is:
[631]   training's auc: 0.933185        valid_1's auc: 0.921781
Fold 2 AUC: 0.921781

--- Fold 3/5 ---
Training until validation scores don't improve for 100 rounds
[200]   training's auc: 0.923076        valid_1's auc: 0.918779
[400]   training's auc: 0.928423        valid_1's auc: 0.919877
[600]   training's auc: 0.932904        valid_1's auc: 0.920312
[800]   training's auc: 0.936921        valid_1's auc: 0.920498
Early stopping, best iteration is:
[747]   training's auc: 0.935897        valid_1's auc: 0.920619
Fold 3 AUC: 0.920619

--- Fold 4/5 ---
Training until validation scores don't improve for 100 rounds
[200]   training's auc: 0.922797        valid_1's auc: 0.919481
[400]   training's auc: 0.928375        valid_1's auc: 0.920635
[600]   training's auc: 0.932944        valid_1's auc: 0.921098
[800]   training's auc: 0.936856        valid_1's auc: 0.921253
Early stopping, best iteration is:
[835]   training's auc: 0.937522        valid_1's auc: 0.9213
Fold 4 AUC: 0.921300

--- Fold 5/5 ---
Training until validation scores don't improve for 100 rounds
[200]   training's auc: 0.922951        valid_1's auc: 0.919204
[400]   training's auc: 0.928501        valid_1's auc: 0.920553
[600]   training's auc: 0.932888        valid_1's auc: 0.92081
Early stopping, best iteration is:
[680]   training's auc: 0.934556        valid_1's auc: 0.920939
Fold 5 AUC: 0.920939

LightGBM OOF AUC: 0.921461
Mean: 0.921466 (+/- 0.000724)

--- 5. Model Training (XGBoost) ---
```

```
In [8]: def train_xgboost(X, y, X_test, n_splits=5):
            """ Trains XGBoost model using StratifiedKFold"""
            skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=config.SEED
            oof_preds = np.zeros(len(X))
            test_preds = np.zeros(len(X_test))

            # Parameters from source notebook
            params = {
                'objective': 'binary:logistic',
                'eval_metric': 'auc',
                'max_depth': 6,
                'learning_rate': 0.05,
                'subsample': 0.8,
                'colsample_bytree': 0.8,
                'min_child_weight': 1,
                'reg_alpha': 0.1,
                'reg_lambda': 0.1,
                'random_state': config.SEED,
                'tree_method': 'hist',
                'n_jobs': -1
            }

            fold_scores = []

            for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
                print(f"\n--- Fold {fold + 1}/{n_splits} ---")
                X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
                y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

                model = xgb.XGBClassifier(**params, n_estimators=2000)
                model.fit(
                    X_train, y_train,
                    eval_set=[(X_val, y_val)],
                    early_stopping_rounds=100,
                    verbose=200
                )

                val_preds = model.predict_proba(X_val)[:, 1]
                oof_preds[val_idx] = val_preds
                test_preds += model.predict_proba(X_test)[:, 1] / n_splits

                score = roc_auc_score(y_val, val_preds)
                fold_scores.append(score)
                print(f"Fold {fold + 1} AUC: {score:.6f}")

            overall_score = roc_auc_score(y, oof_preds)
            print(f"\nXGBoost OOF AUC: {overall_score:.6f}")
            print(f"Mean: {np.mean(fold_scores):.6f} (+/- {np.std(fold_scores):.6f})")
            return oof_preds, test_preds, overall_score
```

```
In [9]: print("\nTraining XGBoost...")
        xgb_oof, xgb_test, xgb_score = train_xgboost(X, y, X_test, n_splits=config.N_SPLITS
        print("\n--- 6. Model Training (CatBoost) ---")
```

```
Training XGBoost...

--- Fold 1/5 ---
[0]	validation_0-auc:0.90833
[200]	validation_0-auc:0.91925
[400]	validation_0-auc:0.92142
[600]	validation_0-auc:0.92198
[776]	validation_0-auc:0.92207
Fold 1 AUC: 0.922122

--- Fold 2/5 ---
[0]	validation_0-auc:0.90679
[200]	validation_0-auc:0.91852
[400]	validation_0-auc:0.92065
[600]	validation_0-auc:0.92140
[772]	validation_0-auc:0.92146
Fold 2 AUC: 0.921505

--- Fold 3/5 ---
[0]	validation_0-auc:0.90673
[200]	validation_0-auc:0.91692
[400]	validation_0-auc:0.91900
[600]	validation_0-auc:0.91959
[800]	validation_0-auc:0.91970
[965]	validation_0-auc:0.91966
Fold 3 AUC: 0.919772

--- Fold 4/5 ---
[0]	validation_0-auc:0.90693
[200]	validation_0-auc:0.91814
[400]	validation_0-auc:0.92006
[600]	validation_0-auc:0.92067
[800]	validation_0-auc:0.92088
[950]	validation_0-auc:0.92083
Fold 4 AUC: 0.920934

--- Fold 5/5 ---
[0]	validation_0-auc:0.90723
[200]	validation_0-auc:0.91757
[400]	validation_0-auc:0.91968
[600]	validation_0-auc:0.92023
[800]	validation_0-auc:0.92026
[919]	validation_0-auc:0.92027
Fold 5 AUC: 0.920333

XGBoost OOF AUC: 0.920922
Mean: 0.920933 (+/- 0.000830)

--- 6. Model Training (CatBoost) ---
```

In [10]:

```python
def train_catboost(X, y, X_test, n_splits=5):
    """ Trains CatBoost model using StratifiedKFold"""
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=config.SEED
    oof_preds = np.zeros(len(X))
    test_preds = np.zeros(len(X_test))
    # Parameters from source notebook
    params = {
        'iterations': 2000,
        'learning_rate': 0.05,
        'depth': 6,
        'l2_leaf_reg': 3,
        'random_seed': config.SEED,
        'loss_function': 'Logloss',
        'eval_metric': 'AUC',
        'early_stopping_rounds': 100,
        'verbose': 200,
        'task_type': 'CPU' # Source notebook uses CPU
    }

    fold_scores = []

    for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
        print(f"\n--- Fold {fold + 1}/{n_splits} ---")
        X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

        model = CatBoostClassifier(**params)
        model.fit(X_train, y_train, eval_set=(X_val, y_val), use_best_model=True)

        val_preds = model.predict_proba(X_val)[:, 1]
        oof_preds[val_idx] = val_preds
        test_preds += model.predict_proba(X_test)[:, 1] / n_splits

        score = roc_auc_score(y_val, val_preds)
        fold_scores.append(score)
        print(f"Fold {fold + 1} AUC: {score:.6f}")

    overall_score = roc_auc_score(y, oof_preds)
    print(f"\nCatBoost OOF AUC: {overall_score:.6f}")
    print(f"Mean: {np.mean(fold_scores):.6f} (+/- {np.std(fold_scores):.6f})")
    return oof_preds, test_preds, overall_score
```

In [11]:
```
print("\nTraining CatBoost...")
cat_oof, cat_test, cat_score = train_catboost(X, y, X_test, n_splits=config.N_SPLIT
print("\n--- 7. Model Evaluation & Ensemble ---")
```

```
Training CatBoost...

--- Fold 1/5 ---
0:      test: 0.9022510 best: 0.9022510 (0)     total: 178ms    remaining: 5m 56s
200:    test: 0.9166465 best: 0.9166465 (200)   total: 21.1s    remaining: 3m 8s
400:    test: 0.9190007 best: 0.9190007 (400)   total: 41.2s    remaining: 2m 44s
600:    test: 0.9202509 best: 0.9202509 (600)   total: 1m 1s    remaining: 2m 23s
800:    test: 0.9209301 best: 0.9209302 (798)   total: 1m 22s   remaining: 2m 2s
1000:   test: 0.9215918 best: 0.9215918 (1000)  total: 1m 42s   remaining: 1m 42s
1200:   test: 0.9219187 best: 0.9219188 (1198)  total: 2m 3s    remaining: 1m 21s
1400:   test: 0.9221681 best: 0.9221732 (1393)  total: 2m 23s   remaining: 1m 1s
1600:   test: 0.9224090 best: 0.9224090 (1600)  total: 2m 44s   remaining: 40.9s
1800:   test: 0.9225697 best: 0.9225697 (1800)  total: 3m 4s    remaining: 20.4s
1999:   test: 0.9226961 best: 0.9226968 (1997)  total: 3m 25s   remaining: 0us

bestTest = 0.9226967803
bestIteration = 1997

Shrink model to first 1998 iterations.
Fold 1 AUC: 0.922697

--- Fold 2/5 ---
0:      test: 0.9023162 best: 0.9023162 (0)     total: 108ms    remaining: 3m 35s
200:    test: 0.9160644 best: 0.9160644 (200)   total: 21s      remaining: 3m 8s
400:    test: 0.9184948 best: 0.9184949 (399)   total: 41.4s    remaining: 2m 44s
600:    test: 0.9196654 best: 0.9196654 (600)   total: 1m 1s    remaining: 2m 23s
800:    test: 0.9205471 best: 0.9205471 (800)   total: 1m 21s   remaining: 2m 2s
1000:   test: 0.9209918 best: 0.9209922 (999)   total: 1m 42s   remaining: 1m 42s
1200:   test: 0.9215138 best: 0.9215138 (1200)  total: 2m 2s    remaining: 1m 21s
1400:   test: 0.9218013 best: 0.9218013 (1400)  total: 2m 23s   remaining: 1m 1s
1600:   test: 0.9220076 best: 0.9220076 (1600)  total: 2m 44s   remaining: 41s
1800:   test: 0.9221596 best: 0.9221596 (1800)  total: 3m 4s    remaining: 20.4s
1999:   test: 0.9223127 best: 0.9223127 (1999)  total: 3m 25s   remaining: 0us

bestTest = 0.922312679
bestIteration = 1999

Fold 2 AUC: 0.922313

--- Fold 3/5 ---
0:      test: 0.9001268 best: 0.9001268 (0)     total: 106ms    remaining: 3m 31s
200:    test: 0.9147979 best: 0.9147979 (200)   total: 21.1s    remaining: 3m 8s
400:    test: 0.9170988 best: 0.9170988 (400)   total: 41.2s    remaining: 2m 44s
600:    test: 0.9182407 best: 0.9182407 (600)   total: 1m 1s    remaining: 2m 24s
800:    test: 0.9190087 best: 0.9190092 (799)   total: 1m 22s   remaining: 2m 3s
1000:   test: 0.9194972 best: 0.9194972 (1000)  total: 1m 43s   remaining: 1m 43s
1200:   test: 0.9199853 best: 0.9199918 (1190)  total: 2m 4s    remaining: 1m 22s
1400:   test: 0.9202397 best: 0.9202403 (1399)  total: 2m 25s   remaining: 1m 2s
1600:   test: 0.9204282 best: 0.9204390 (1591)  total: 2m 46s   remaining: 41.5s
1800:   test: 0.9205645 best: 0.9205698 (1794)  total: 3m 7s    remaining: 20.8s
1999:   test: 0.9206759 best: 0.9206772 (1997)  total: 3m 28s   remaining: 0us

bestTest = 0.9206772067
bestIteration = 1997

Shrink model to first 1998 iterations.
```

```
Fold 3 AUC: 0.920677

--- Fold 4/5 ---
0:      test: 0.9006072 best: 0.9006072 (0)     total: 107ms    remaining: 3m 33s
200:    test: 0.9155320 best: 0.9155320 (200)   total: 22.2s    remaining: 3m 18s
400:    test: 0.9180183 best: 0.9180183 (400)   total: 45.6s    remaining: 3m 2s
600:    test: 0.9191428 best: 0.9191446 (599)   total: 1m 7s    remaining: 2m 38s
800:    test: 0.9200291 best: 0.9200298 (795)   total: 1m 30s   remaining: 2m 16s
1000:   test: 0.9204769 best: 0.9204769 (1000)  total: 1m 55s   remaining: 1m 55s
1200:   test: 0.9208387 best: 0.9208412 (1193)  total: 2m 18s   remaining: 1m 32s
1400:   test: 0.9210745 best: 0.9210775 (1388)  total: 2m 40s   remaining: 1m 8s
1600:   test: 0.9212991 best: 0.9212991 (1600)  total: 3m 4s    remaining: 45.9s
1800:   test: 0.9214484 best: 0.9214515 (1793)  total: 3m 27s   remaining: 22.9s
1999:   test: 0.9215817 best: 0.9215839 (1960)  total: 3m 50s   remaining: 0us

bestTest = 0.9215838764
bestIteration = 1960

Shrink model to first 1961 iterations.
Fold 4 AUC: 0.921584

--- Fold 5/5 ---
0:      test: 0.9010101 best: 0.9010101 (0)     total: 119ms    remaining: 3m 58s
200:    test: 0.9152413 best: 0.9152413 (200)   total: 22s      remaining: 3m 17s
400:    test: 0.9175739 best: 0.9175739 (400)   total: 43.1s    remaining: 2m 52s
600:    test: 0.9187326 best: 0.9187326 (600)   total: 1m 4s    remaining: 2m 30s
800:    test: 0.9194385 best: 0.9194398 (799)   total: 1m 26s   remaining: 2m 9s
1000:   test: 0.9199161 best: 0.9199212 (993)   total: 1m 49s   remaining: 1m 49s
1200:   test: 0.9202181 best: 0.9202221 (1194)  total: 2m 11s   remaining: 1m 27s
1400:   test: 0.9204521 best: 0.9204531 (1398)  total: 2m 34s   remaining: 1m 5s
1600:   test: 0.9206681 best: 0.9206790 (1592)  total: 2m 55s   remaining: 43.6s
1800:   test: 0.9207619 best: 0.9207637 (1798)  total: 3m 16s   remaining: 21.7s
1999:   test: 0.9208691 best: 0.9208691 (1999)  total: 3m 39s   remaining: 0us

bestTest = 0.9208690996
bestIteration = 1999

Fold 5 AUC: 0.920869

CatBoost OOF AUC: 0.921624
Mean: 0.921628 (+/- 0.000787)

--- 7. Model Evaluation & Ensemble ---
```

In [12]:

```
# Model Comparison
print("\nMODEL COMPARISON")
comparison = pd.DataFrame({
    'Model': ['LightGBM', 'XGBoost', 'CatBoost'],
    'OOF AUC': [lgb_score, xgb_score, cat_score]
}).sort_values('OOF AUC', ascending=False)
print(comparison)

# Create Ensemble
print("\nCREATING ENSEMBLE")

# 1. Simple Average
```

```
simple_oof = (lgb_oof + xgb_oof + cat_oof) / 3
simple_test = (lgb_test + xgb_test + cat_test) / 3
simple_score = roc_auc_score(y, simple_oof)

# 2. Weighted Average
total_auc = lgb_score + xgb_score + cat_score
w_lgb = lgb_score / total_auc
w_xgb = xgb_score / total_auc
w_cat = cat_score / total_auc
weighted_oof = (lgb_oof * w_lgb) + (xgb_oof * w_xgb) + (cat_oof * w_cat)
weighted_test = (lgb_test * w_lgb) + (xgb_test * w_xgb) + (cat_test * w_cat)
weighted_score = roc_auc_score(y, weighted_oof)

# 3. Rank Average
rank_oof = (rankdata(lgb_oof) + rankdata(xgb_oof) + rankdata(cat_oof)) / (3 * len(l
rank_test = (rankdata(lgb_test) + rankdata(xgb_test) + rankdata(cat_test)) / (3 * l
rank_score = roc_auc_score(y, rank_oof)

# Ensemble Results
ensemble_results = pd.DataFrame({
    'Ensemble': ['Simple Average', 'Weighted Average', 'Rank Average'],
    'OOF AUC': [simple_score, weighted_score, rank_score]
}).sort_values('OOF AUC', ascending=False)

print("\nEnsemble Results:")
print(ensemble_results)
print(f"\nWeights: LGB={w_lgb:.3f}, XGB={w_xgb:.3f}, CAT={w_cat:.3f}")

# Choose best
best_idx = ensemble_results['OOF AUC'].idxmax()
best_name = ensemble_results.loc[best_idx, 'Ensemble']
best_score = ensemble_results.loc[best_idx, 'OOF AUC']

if best_name == 'Simple Average':
    final_preds = simple_test
elif best_name == 'Weighted Average':
    final_preds = weighted_test
else:
    final_preds = rank_test

print(f"\nBest Ensemble: {best_name} (AUC: {best_score:.6f})")
print("\n--- 8. Submission ---")
```

```
MODEL COMPARISON
      Model   OOF AUC
2  CatBoost  0.921624
0  LightGBM  0.921461
1   XGBoost  0.920922

CREATING ENSEMBLE

Ensemble Results:
           Ensemble   OOF AUC
2      Rank Average  0.921893
1  Weighted Average  0.921865
0    Simple Average  0.921865

Weights: LGB=0.333, XGB=0.333, CAT=0.333

Best Ensemble: Rank Average (AUC: 0.921893)

--- 8. Submission ---
```

In [13]:
```python
# Create submission
# (Uncommented from source to generate the file as per 'Version' format)
submission = pd.DataFrame({
    'id': test_ids,
    config.TARGET: final_preds
})

submission.to_csv('submission.csv', index=False)
print("SUBMISSION CREATED")
print(f"File: submission.csv")
print(f"Shape: {submission.shape}")
print(f"\nPreview:")
display(submission.head(10))
```

```
SUBMISSION CREATED
File: submission.csv
Shape: (254569, 2)

Preview:
```

	id	loan_paid_back
0	593994	0.499309
1	593995	0.788466
2	593996	0.128066
3	593997	0.506112
4	593998	0.623759
5	593999	0.727296
6	594000	0.834262
7	594001	0.661948
8	594002	0.526185
9	594003	0.018947