Loan Approval Prediction Task

Matthew Loh

Table of contents

Dataset Source : https://www.kaggle.com/competitions/playground-series-s4e10/data

```
from sklearn.metrics import roc_curve
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import mutual_info_classif
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.calibration import CalibrationDisplay
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from hillclimbers import climb_hill, partial
from category_encoders import TargetEncoder
from sklearn.pipeline import make_pipeline
from sklearn.metrics import roc_auc_score
```

```
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import pandas as pd
import numpy as np
import warnings
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import learning_curve

warnings.filterwarnings("ignore")


oof = {}
test_pred = {}
NUM_FOLD = 5
target = "loan_status"
np.random.seed(42)


# Load the data
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

# Display basic information about the dataset
print("Training set shape:", train_df.shape)
print("\nTest set shape:", test_df.shape)
print("\nTraining set info:")
train_df.info()

# Display first few rows
print("\nFirst few rows of training data:")
display(train_df.head())

# Basic statistics of numerical columns
print("\nNumerical columns statistics:")
display(train_df.describe())

# Check for missing values
```

```
print("\nMissing values in training set:")
display(train_df.isnull().sum())

# Plot distribution of target variable
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x=target)
plt.title("Distribution of Loan Status")
plt.show()

# Correlation analysis
numerical_cols = train_df.select_dtypes(include=["int64", "float64"]).columns
correlation_matrix = train_df[numerical_cols].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix of Numerical Features")
plt.show()

# Feature distributions
categorical_cols = train_df.select_dtypes(include=["object"]).columns
numerical_cols = train_df.select_dtypes(include=["int64", "float64"]).columns.drop(
    target
)

# Plot distributions for numerical features
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data=train_df, x=col, hue=target, multiple="stack")
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()

# Plot categorical features
plt.figure(figsize=(15, 10))
for i, col in enumerate(categorical_cols, 1):
    plt.subplot(3, 3, i)
    sns.countplot(data=train_df, x=col, hue=target)
    plt.xticks(rotation=45)
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()
```

```
# Preprocess data
def preprocess_data(df):
    df = df.copy()

    # Handle missing values
    for col in df.columns:
        if df[col].dtype != "object":
            df[col] = df[col].fillna(df[col].median())
        else:
            df[col] = df[col].fillna(df[col].mode()[0])

    # Encode categorical variables
    categorical_cols = df.select_dtypes(include=["object"]).columns
    for col in categorical_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col].astype(str))

    return df

# Prepare the data
X = preprocess_data(train_df.drop([target, 'id'], axis=1))
y = train_df[target]
X_test = preprocess_data(test_df.drop('id', axis=1))

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_test_scaled = scaler.transform(X_test)

# Add feature selection section
def select_features(X, y):
    # Calculate mutual information scores
    mi_scores = mutual_info_classif(X, y)

    # Create DataFrame of features and their scores
    feature_importance = pd.DataFrame({
        'Feature': X.columns,
        'MI_Score': mi_scores
    }).sort_values('MI_Score', ascending=False)

    # Plot feature importance
    plt.figure(figsize=(12, 6))
```

```
    sns.barplot(x='MI_Score', y='Feature', data=feature_importance.head(15))
    plt.title('Top 15 Features by Mutual Information Score')
    plt.tight_layout()
    plt.show()

    return feature_importance

# Perform feature selection
feature_importance = select_features(pd.DataFrame(X_scaled, columns=X.columns), y)
print("\nTop 10 Most Important Features:")
display(feature_importance.head(10))

# Initialize models
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42),
    "LightGBM": LGBMClassifier(random_state=42),
    "CatBoost": CatBoostClassifier(random_state=42, verbose=False)
}

# Train and evaluate initial models
def train_and_evaluate_classifier(model, X, y, test_data, model_name):
    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    model.fit(X_train, y_train)

    train_pred = model.predict_proba(X_train)[:, 1]
    val_pred = model.predict_proba(X_val)[:, 1]

    train_auc = roc_auc_score(y_train, train_pred)
    val_auc = roc_auc_score(y_val, val_pred)

    print(f"\n{model_name} Results:")
    print(f"Train ROC-AUC: {train_auc:.4f}")
    print(f"Validation ROC-AUC: {val_auc:.4f}")

    return model, (y_train, train_pred, y_val, val_pred)

model_results = {}
```

```
for name, model in models.items():
    print(f"\nTraining {name}...")
    model_results[name] = train_and_evaluate_classifier(
        model, X_scaled, y, X_test_scaled, name
    )

# Now tune the models
param_grids = {
    "Logistic Regression": {
        'C': [0.001, 0.01, 0.1, 1, 10],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    },
    "Random Forest": {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5]
    },
    "XGBoost": {
        'max_depth': [3, 6],
        'learning_rate': [0.01, 0.1],
        'n_estimators': [100, 200]
    },
    "LightGBM": {
        'num_leaves': [31, 127],
        'learning_rate': [0.01, 0.1],
        'n_estimators': [100, 200]
    },
    "CatBoost": {
        'depth': [6, 8],
        'learning_rate': [0.01, 0.1],
        'iterations': [100, 200]
    }
}

# Tune each model
tuned_models = {}
for name, model in models.items():
    print(f"\nTuning {name}...")
    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grids[name],
```

```
        cv=5,
        scoring='roc_auc',
        verbose=1,
        n_jobs=-1
    )
    grid_search.fit(X_scaled, y)
    tuned_models[name] = grid_search.best_estimator_
    print(f"Best parameters for {name}:")
    print(grid_search.best_params_)
    print(f"Best ROC-AUC: {grid_search.best_score_:.4f}")

# Plot ROC curves
def plot_roc_curves(models_results):
    plt.figure(figsize=(10, 8))

    for name, (model, results) in models_results.items():
        y_val, val_pred = results[2], results[3]
        fpr, tpr, _ = roc_curve(y_val, val_pred)
        auc = roc_auc_score(y_val, val_pred)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {auc:.3f})")

    plt.plot([0, 1], [0, 1], "r--")
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curves for Different Models")
    plt.legend(loc="lower right")
    plt.show()


plot_roc_curves(model_results)


# Plot confusion matrices and print classification reports
def plot_confusion_matrices(models_results):
    n_models = len(models_results)
    fig, axes = plt.subplots(2, 3, figsize=(20, 12))
    axes = axes.ravel()

    for idx, (name, (model, results)) in enumerate(models_results.items()):
        y_val, val_pred_prob = results[2], results[3]
```

```
        # Convert probabilities to binary predictions using 0.5 threshold
        val_pred = (val_pred_prob > 0.5).astype(int)

        # Calculate confusion matrix
        cm = confusion_matrix(y_val, val_pred)

        # Plot confusion matrix
        sns.heatmap(cm, annot=True, fmt="d", ax=axes[idx], cmap="Blues", cbar=False)
        axes[idx].set_title(f"{name} Confusion Matrix")
        axes[idx].set_xlabel("Predicted")
        axes[idx].set_ylabel("Actual")

        # Print classification report
        print(f"\nClassification Report for {name}:")
        print(classification_report(y_val, val_pred))

    # Remove extra subplot if any
    if n_models < len(axes):
        for idx in range(n_models, len(axes)):
            fig.delaxes(axes[idx])

    plt.tight_layout()
    plt.show()


plot_confusion_matrices(model_results)


# Additional analysis: Threshold sensitivity
def plot_threshold_sensitivity(models_results):
    plt.figure(figsize=(12, 6))

    thresholds = np.linspace(0, 1, 100)
    for name, (model, results) in models_results.items():
        y_val, val_pred_prob = results[2], results[3]

        accuracies = []
        for threshold in thresholds:
            val_pred = (val_pred_prob > threshold).astype(int)
            accuracy = (val_pred == y_val).mean()
            accuracies.append(accuracy)
```

```
        plt.plot(thresholds, accuracies, label=name)

    plt.xlabel("Classification Threshold")
    plt.ylabel("Accuracy")
    plt.title("Model Accuracy vs Classification Threshold")
    plt.legend()
    plt.grid(True)
    plt.show()


# Plot threshold sensitivity
plot_threshold_sensitivity(model_results)


# Function to find optimal threshold
def find_optimal_threshold(y_true, y_pred_prob):
    thresholds = np.linspace(0, 1, 100)
    best_threshold = 0.5
    best_accuracy = 0

    for threshold in thresholds:
        y_pred = (y_pred_prob > threshold).astype(int)
        accuracy = (y_pred == y_true).mean()

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_threshold = threshold

    return best_threshold, best_accuracy


# Find and print optimal thresholds for each model
print("\nOptimal Classification Thresholds:")
for name, (model, results) in model_results.items():
    y_val, val_pred_prob = results[2], results[3]
    opt_threshold, opt_accuracy = find_optimal_threshold(y_val, val_pred_prob)
    print(f"{name}:")
    print(f"  Optimal Threshold: {opt_threshold:.3f}")
    print(f"  Optimal Accuracy: {opt_accuracy:.3f}")

    # Print confusion matrix with optimal threshold
    val_pred_opt = (val_pred_prob > opt_threshold).astype(int)
```

```
    cm_opt = confusion_matrix(y_val, val_pred_opt)
    print("  Confusion Matrix with Optimal Threshold:")
    print(cm_opt)
    print("\n")


# First, define the parameter grids for each model
param_grids = {
    "Logistic Regression": {
        'C': [0.001, 0.01, 0.1, 1, 10],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    },
    "Random Forest": {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5]
    },
    "XGBoost": {
        'max_depth': [3, 6],
        'learning_rate': [0.01, 0.1],
        'n_estimators': [100, 200]
    },
    "LightGBM": {
        'num_leaves': [31, 127],
        'learning_rate': [0.01, 0.1],
        'n_estimators': [100, 200]
    },
    "CatBoost": {
        'depth': [6, 8],
        'learning_rate': [0.01, 0.1],
        'iterations': [100, 200]
    }
}

# Tune each model
tuned_models = {}
for name, model in models.items():
    print(f"\nTuning {name}...")
    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grids[name],
```

```
        cv=5,
        scoring='roc_auc',
        verbose=1,
        n_jobs=-1
    )
    grid_search.fit(X_scaled, y)
    tuned_models[name] = grid_search.best_estimator_
    print(f"Best parameters for {name}:")
    print(grid_search.best_params_)
    print(f"Best ROC-AUC: {grid_search.best_score_:.4f}")

# Add this function before using it
def plot_feature_importance(model, X, model_name):
    if hasattr(model, 'feature_importances_'):
        importances = model.feature_importances_
    elif hasattr(model, 'coef_'):
        importances = np.abs(model.coef_[0])
    else:
        print(f"Feature importance not available for {model_name}")
        return

    feature_imp = pd.DataFrame({
        'feature': X.columns,
        'importance': importances
    })
    feature_imp = feature_imp.sort_values('importance', ascending=False).head(20)

    plt.figure(figsize=(10, 6))
    sns.barplot(x='importance', y='feature', data=feature_imp)
    plt.title(f'Top 20 Feature Importances - {model_name}')
    plt.tight_layout()
    plt.show()

# Now we can use it for the models
for name, model in tuned_models.items():
    if name != "Logistic Regression":  # Skip logistic regression as it needs different handl
        plot_feature_importance(model, X, name)

# For Logistic Regression, plot coefficients
lr_model = tuned_models["Logistic Regression"]
lr_coef = pd.DataFrame({
    'feature': X.columns,
```

```
    'coefficient': np.abs(lr_model.coef_[0])
}).sort_values('coefficient', ascending=False).head(20)

plt.figure(figsize=(10, 6))
sns.barplot(x='coefficient', y='feature', data=lr_coef)
plt.title('Top 20 Feature Coefficients - Logistic Regression')
plt.tight_layout()
plt.show()


def plot_classification_metrics(models_results):
    # Prepare data for plotting
    metrics_data = []
    for name, (model, results) in models_results.items():
        y_val, val_pred_prob = results[2], results[3]
        val_pred = (val_pred_prob > 0.5).astype(int)

        # Get classification report as dict
        report = classification_report(y_val, val_pred, output_dict=True)

        # Extract metrics
        metrics_data.append(
            {
                "Model": name,
                "Precision": report["weighted avg"]["precision"],
                "Recall": report["weighted avg"]["recall"],
                "F1-Score": report["weighted avg"]["f1-score"],
                "Accuracy": report["accuracy"],
            }
        )

    # Convert to DataFrame
    metrics_df = pd.DataFrame(metrics_data)

    # Create subplot for each metric
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))
    metrics = ["Precision", "Recall", "F1-Score", "Accuracy"]

    for ax, metric in zip(axes.ravel(), metrics):
        sns.barplot(x="Model", y=metric, data=metrics_df, ax=ax)
        ax.set_title(f"{metric} by Model")
        ax.tick_params(axis="x", rotation=45)
```

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

    # Create a heatmap of all metrics
    plt.figure(figsize=(12, 6))
    metrics_heatmap = metrics_df.set_index("Model")
    sns.heatmap(metrics_heatmap, annot=True, fmt=".3f", cmap="YlOrRd")
    plt.title("Model Performance Metrics Comparison")
    plt.tight_layout()
    plt.show()


# Plot the metrics
plot_classification_metrics(model_results)


# Function to plot detailed per-class metrics
def plot_per_class_metrics(models_results):
    # Prepare data for plotting
    class_metrics = []

    for name, (model, results) in models_results.items():
        y_val, val_pred_prob = results[2], results[3]
        val_pred = (val_pred_prob > 0.5).astype(int)

        # Get classification report as dict
        report = classification_report(y_val, val_pred, output_dict=True)

        # Extract per-class metrics
        for class_label in ["0", "1"]:  # Binary classification
            class_metrics.append(
                {
                    "Model": name,
                    "Class": f"Class {class_label}",
                    "Precision": report[class_label]["precision"],
                    "Recall": report[class_label]["recall"],
                    "F1-Score": report[class_label]["f1-score"],
                }
            )

    # Convert to DataFrame
    metrics_df = pd.DataFrame(class_metrics)
```

```
    # Create subplot for each metric
    fig, axes = plt.subplots(1, 3, figsize=(18, 6))
    metrics = ["Precision", "Recall", "F1-Score"]

    for ax, metric in zip(axes, metrics):
        sns.barplot(x="Model", y=metric, hue="Class", data=metrics_df, ax=ax)
        ax.set_title(f"{metric} by Model and Class")
        ax.tick_params(axis="x", rotation=45)

    plt.tight_layout()
    plt.show()


# Plot per-class metrics
plot_per_class_metrics(model_results)


# Function to plot model comparison summary
def plot_model_summary(models_results):
    summary_data = []

    for name, (model, results) in models_results.items():
        y_val, val_pred_prob = results[2], results[3]
        val_pred = (val_pred_prob > 0.5).astype(int)

        # Calculate various metrics
        auc = roc_auc_score(y_val, val_pred_prob)
        accuracy = (val_pred == y_val).mean()

        # Get optimal threshold
        opt_threshold, opt_accuracy = find_optimal_threshold(y_val, val_pred_prob)

        summary_data.append(
            {
                "Model": name,
                "ROC-AUC": auc,
                "Accuracy": accuracy,
                "Optimal Threshold": opt_threshold,
                "Optimal Accuracy": opt_accuracy,
            }
        )
```

```
    # Convert to DataFrame
    summary_df = pd.DataFrame(summary_data)

    # Create summary visualization
    fig, axes = plt.subplots(2, 1, figsize=(12, 12))

    # Performance metrics
    sns.barplot(x="Model", y="ROC-AUC", data=summary_df, ax=axes[0], color="skyblue")
    axes[0].set_title("ROC-AUC Score by Model")
    axes[0].tick_params(axis="x", rotation=45)

    # Threshold comparison
    summary_df.plot(
        x="Model", y=["Accuracy", "Optimal Accuracy"], kind="bar", ax=axes[1], width=0.8
    )
    axes[1].set_title("Default vs Optimal Threshold Accuracy")
    axes[1].tick_params(axis="x", rotation=45)

    plt.tight_layout()
    plt.show()

    # Threshold heatmap
    plt.figure(figsize=(10, 4))
    threshold_df = summary_df[["Model", "Optimal Threshold", "Optimal Accuracy"]]
    sns.heatmap(threshold_df.set_index("Model"), annot=True, fmt=".3f", cmap="YlOrRd")
    plt.title("Optimal Thresholds and Corresponding Accuracies")
    plt.tight_layout()
    plt.show()


# Plot model summary
plot_model_summary(model_results)

# Add detailed EDA visualizations
def plot_detailed_eda(train_df, target):
    # Target Distribution with Percentages
    plt.figure(figsize=(10, 6))
    target_counts = train_df[target].value_counts(normalize=True) * 100
    sns.barplot(x=target_counts.index, y=target_counts.values)
    plt.title('Distribution of Loan Status (%)')
    plt.ylabel('Percentage')
    for i, v in enumerate(target_counts):
```

```
        plt.text(i, v, f'{v:.1f}%', ha='center')
    plt.show()

    # Correlation Matrix with Annotations
    plt.figure(figsize=(12, 8))
    numerical_cols = train_df.select_dtypes(include=['int64', 'float64']).columns
    corr_matrix = train_df[numerical_cols].corr()
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
    sns.heatmap(corr_matrix, mask=mask, annot=True, cmap='coolwarm', center=0)
    plt.title('Correlation Matrix of Numerical Features')
    plt.tight_layout()
    plt.show()

    # Feature Distributions by Target
    numerical_features = [col for col in numerical_cols if col != target]
    n_features = len(numerical_features)
    n_rows = (n_features + 2) // 3

    plt.figure(figsize=(15, 5*n_rows))
    for i, feature in enumerate(numerical_features, 1):
        plt.subplot(n_rows, 3, i)
        sns.boxplot(x=target, y=feature, data=train_df)
        plt.title(f'{feature} by Loan Status')
    plt.tight_layout()
    plt.show()

# Run detailed EDA
plot_detailed_eda(train_df, target)

# Add model evaluation metrics visualization
def plot_model_evaluation_metrics(models_results):
    # Prepare metrics data
    metrics_data = []
    for name, (model, results) in models_results.items():
        y_val, val_pred_prob = results[2], results[3]
        val_pred = (val_pred_prob > 0.5).astype(int)

        # Calculate metrics
        accuracy = accuracy_score(y_val, val_pred)
        precision = precision_score(y_val, val_pred, average='weighted')
        recall = recall_score(y_val, val_pred, average='weighted')
        f1 = f1_score(y_val, val_pred, average='weighted')
```

```
        auc = roc_auc_score(y_val, val_pred_prob)

        metrics_data.append({
            'Model': name,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1-Score': f1,
            'AUC-ROC': auc
        })

    metrics_df = pd.DataFrame(metrics_data)

    # Create heatmap of metrics
    plt.figure(figsize=(12, 6))
    metrics_heatmap = metrics_df.set_index('Model').round(3)
    sns.heatmap(metrics_heatmap, annot=True, cmap='YlOrRd', fmt='.3f')
    plt.title('Model Performance Metrics Comparison')
    plt.tight_layout()
    plt.show()

    return metrics_df

# Run model evaluation
metrics_comparison = plot_model_evaluation_metrics(model_results)

# Add learning curves analysis
def plot_learning_curves(model, X, y, model_name):
    train_sizes, train_scores, val_scores = learning_curve(
        model, X, y, train_sizes=np.linspace(0.1, 1.0, 10),
        cv=5, scoring='roc_auc', n_jobs=-1
    )

    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    val_mean = np.mean(val_scores, axis=1)
    val_std = np.std(val_scores, axis=1)

    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_mean, label='Training score')
    plt.plot(train_sizes, val_mean, label='Cross-validation score')
    plt.fill_between(train_sizes, train_mean - train_std,
```

```
                        train_mean + train_std, alpha=0.1)
    plt.fill_between(train_sizes, val_mean - val_std,
                        val_mean + val_std, alpha=0.1)
    plt.xlabel('Training Examples')
    plt.ylabel('ROC-AUC Score')
    plt.title(f'Learning Curves - {model_name}')
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()

# Plot learning curves for each model
for name, model in tuned_models.items():
    plot_learning_curves(model, X_scaled, y, name)
```

```
Training set shape: (58645, 13)

Test set shape: (39098, 12)

Training set info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58645 entries, 0 to 58644
Data columns (total 13 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   id                          58645 non-null  int64
 1   person_age                  58645 non-null  int64
 2   person_income               58645 non-null  int64
 3   person_home_ownership       58645 non-null  object
 4   person_emp_length           58645 non-null  float64
 5   loan_intent                 58645 non-null  object
 6   loan_grade                  58645 non-null  object
 7   loan_amnt                   58645 non-null  int64
 8   loan_int_rate               58645 non-null  float64
 9   loan_percent_income         58645 non-null  float64
 10  cb_person_default_on_file   58645 non-null  object
 11  cb_person_cred_hist_length  58645 non-null  int64
 12  loan_status                 58645 non-null  int64
dtypes: float64(3), int64(6), object(4)
memory usage: 5.8+ MB

First few rows of training data:
```

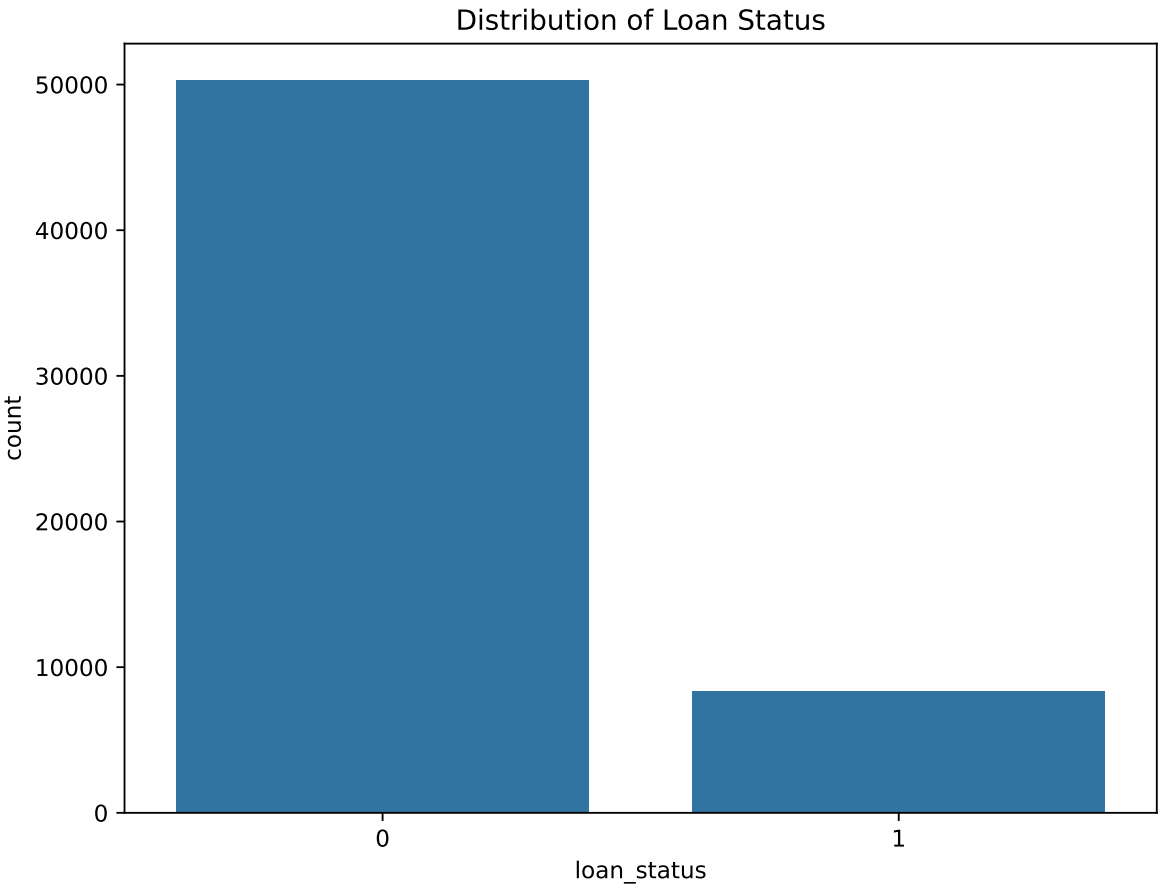
	id	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan
0	0	37	35000	RENT	0.0	EDUCATION	B
1	1	22	56000	OWN	6.0	MEDICAL	C
2	2	29	28800	OWN	8.0	PERSONAL	A
3	3	30	70000	RENT	14.0	VENTURE	B
4	4	22	60000	RENT	2.0	MEDICAL	A

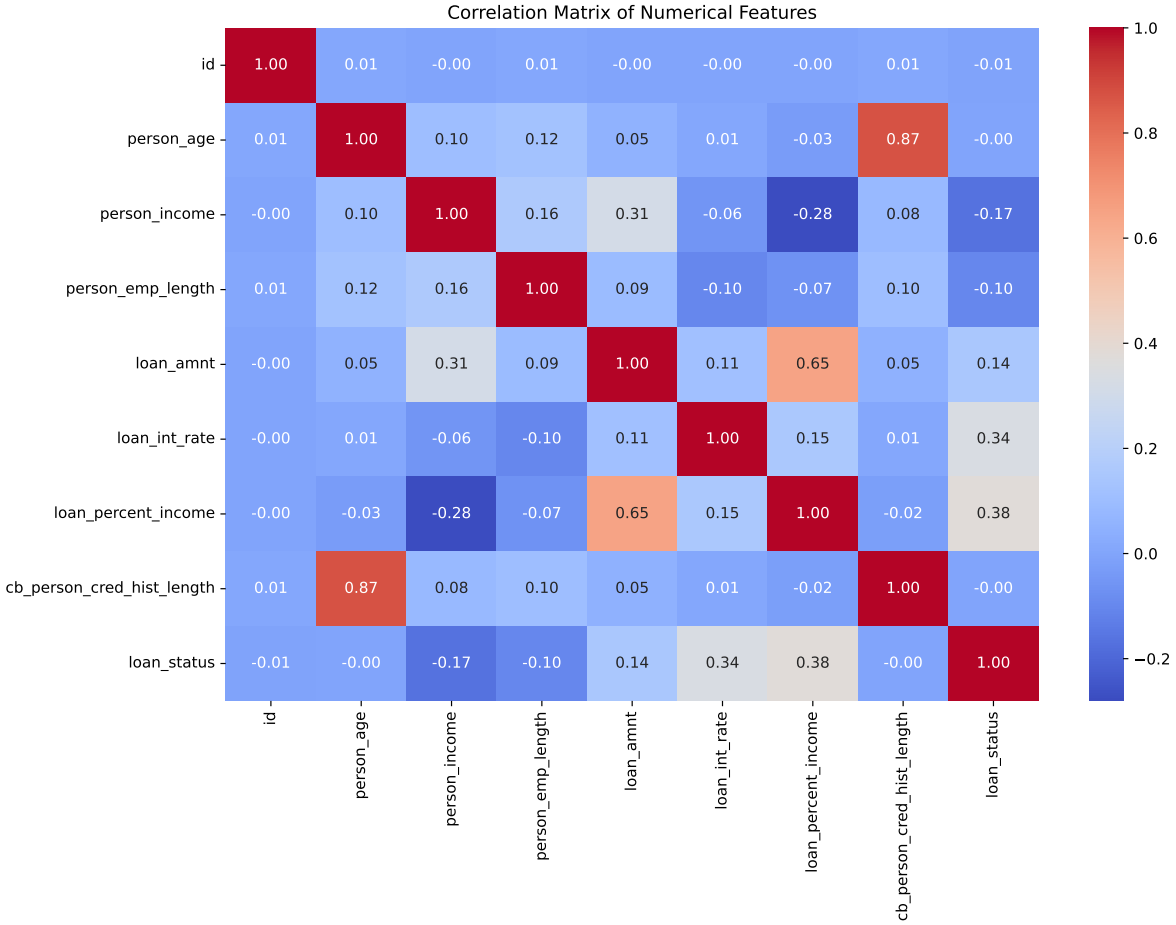
Numerical columns statistics:

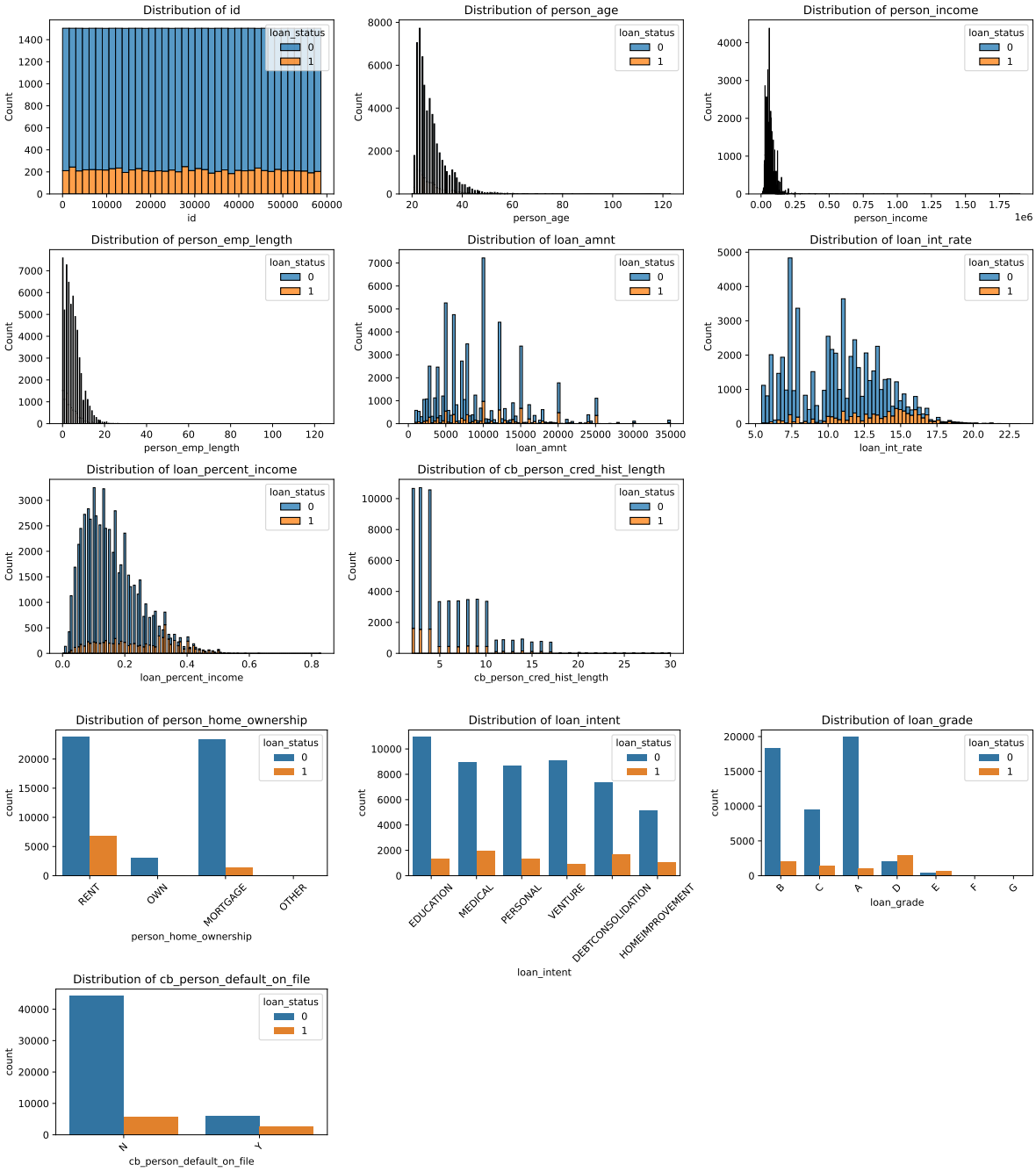
	id	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate
count	58645.000000	58645.000000	5.864500e+04	58645.000000	58645.000000	58645.000000
mean	29322.000000	27.550857	6.404617e+04	4.701015	9217.556518	10.677874
std	16929.497605	6.033216	3.793111e+04	3.959784	5563.807384	3.034697
min	0.000000	20.000000	4.200000e+03	0.000000	500.000000	5.420000
25%	14661.000000	23.000000	4.200000e+04	2.000000	5000.000000	7.880000
50%	29322.000000	26.000000	5.800000e+04	4.000000	8000.000000	10.750000
75%	43983.000000	30.000000	7.560000e+04	7.000000	12000.000000	12.990000
max	58644.000000	123.000000	1.900000e+06	123.000000	35000.000000	23.220000

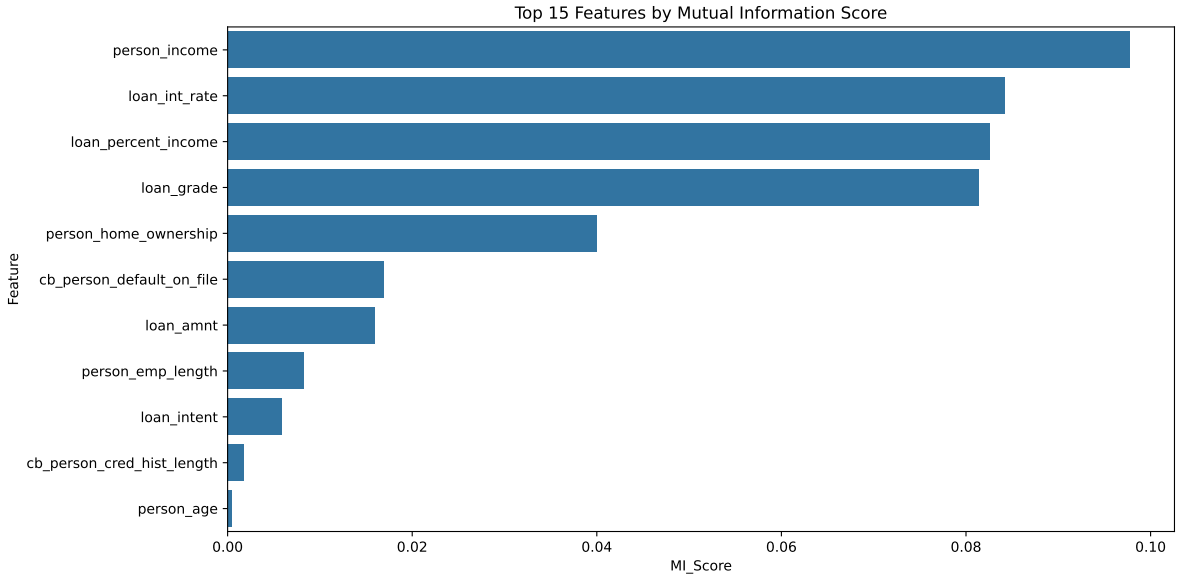
Missing values in training set:

```
id                          0
person_age                  0
person_income               0
person_home_ownership       0
person_emp_length           0
loan_intent                 0
loan_grade                  0
loan_amnt                   0
loan_int_rate               0
loan_percent_income         0
cb_person_default_on_file   0
cb_person_cred_hist_length  0
loan_status                 0
dtype: int64
```


Distribution of Loan Status
50000
40000
30000
20000
10000
0
count
0
1
loan_status


Correlation Matrix of Numerical Features
id
person_age
person_income
person_emp_length
loan_amnt
loan_int_rate
loan_percent_income
cb_person_cred_hist_length
loan_status
1.00 0.01 -0.00 0.01 -0.00 -0.00 -0.00 0.01 -0.01
0.01 1.00 0.10 0.12 0.05 0.01 -0.03 0.87 -0.00
-0.00 0.10 1.00 0.16 0.31 -0.06 -0.28 0.08 -0.17
0.01 0.12 0.16 1.00 0.09 -0.10 -0.07 0.10 -0.10
-0.00 0.05 0.31 0.09 1.00 0.11 0.65 0.05 0.14
-0.00 0.01 -0.06 -0.10 0.11 1.00 0.15 0.01 0.34
-0.00 -0.03 -0.28 -0.07 0.65 0.15 1.00 -0.02 0.38
0.01 0.87 0.08 0.10 0.05 0.01 -0.02 1.00 -0.00
-0.01 -0.00 -0.17 -0.10 0.14 0.34 0.38 -0.00 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


Distribution of id
Distribution of person_age
Distribution of person_income
Distribution of person_emp_length
Distribution of loan_amnt
Distribution of loan_int_rate
Distribution of loan_percent_income
Distribution of cb_person_cred_hist_length
Distribution of person_home_ownership
Distribution of loan_intent
Distribution of loan_grade
Distribution of cb_person_default_on_file
loan_status
0
1
Count
count
id
person_age
person_income
1e6
person_emp_length
loan_amnt
loan_int_rate
loan_percent_income
cb_person_cred_hist_length
RENT
OWN
MORTGAGE
OTHER
person_home_ownership
EDUCATION
MEDICAL
PERSONAL
VENTURE
DEBTCONSOLIDATION
HOMEIMPROVEMENT
loan_intent
B
C
A
D
E
F
G
loan_grade
N
Y
cb_person_default_on_file



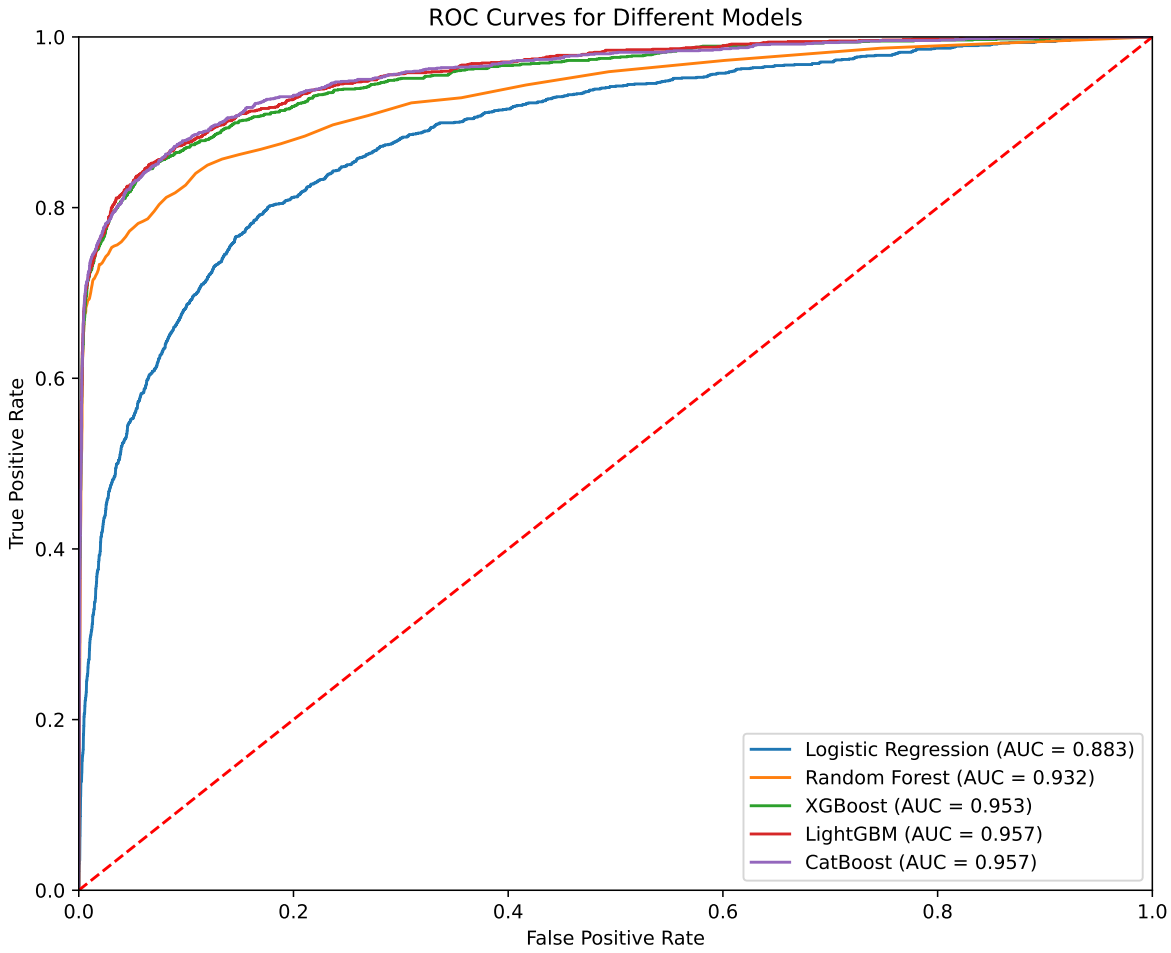

```
Top 10 Most Important Features:
```

	Feature	MI_Score
1	person_income	0.097702
7	loan_int_rate	0.084164
8	loan_percent_income	0.082553
5	loan_grade	0.081381
2	person_home_ownership	0.040029
9	cb_person_default_on_file	0.016888
6	loan_amnt	0.016007
3	person_emp_length	0.008263
4	loan_intent	0.005864
10	cb_person_cred_hist_length	0.001753

```
Training Logistic Regression...

Logistic Regression Results:
Train ROC-AUC: 0.8801
Validation ROC-AUC: 0.8833

Training Random Forest...
```

```
Random Forest Results:
Train ROC-AUC: 1.0000
Validation ROC-AUC: 0.9317

Training XGBoost...

XGBoost Results:
Train ROC-AUC: 0.9884
Validation ROC-AUC: 0.9529

Training LightGBM...
[LightGBM] [Info] Number of positive: 6680, number of negative: 40236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.0004
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 881
[LightGBM] [Info] Number of data points in the train set: 46916, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.142382 -> initscore=-1.795644
[LightGBM] [Info] Start training from score -1.795644

LightGBM Results:
Train ROC-AUC: 0.9768
Validation ROC-AUC: 0.9569

Training CatBoost...

CatBoost Results:
Train ROC-AUC: 0.9767
Validation ROC-AUC: 0.9565

Tuning Logistic Regression...
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters for Logistic Regression:
{'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
Best ROC-AUC: 0.8809

Tuning Random Forest...
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters for Random Forest:
{'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 200}
Best ROC-AUC: 0.9403
```

```
Tuning XGBoost...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for XGBoost:
{'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 200}
Best ROC-AUC: 0.9549

Tuning LightGBM...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[LightGBM] [Info] Number of positive: 8350, number of negative: 50295
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.0018
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 887
[LightGBM] [Info] Number of data points in the train set: 58645, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.142382 -> initscore=-1.795644
[LightGBM] [Info] Start training from score -1.795644
Best parameters for LightGBM:
{'learning_rate': 0.1, 'n_estimators': 200, 'num_leaves': 31}
Best ROC-AUC: 0.9571

Tuning CatBoost...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for CatBoost:
{'depth': 8, 'iterations': 200, 'learning_rate': 0.1}
Best ROC-AUC: 0.9512
```



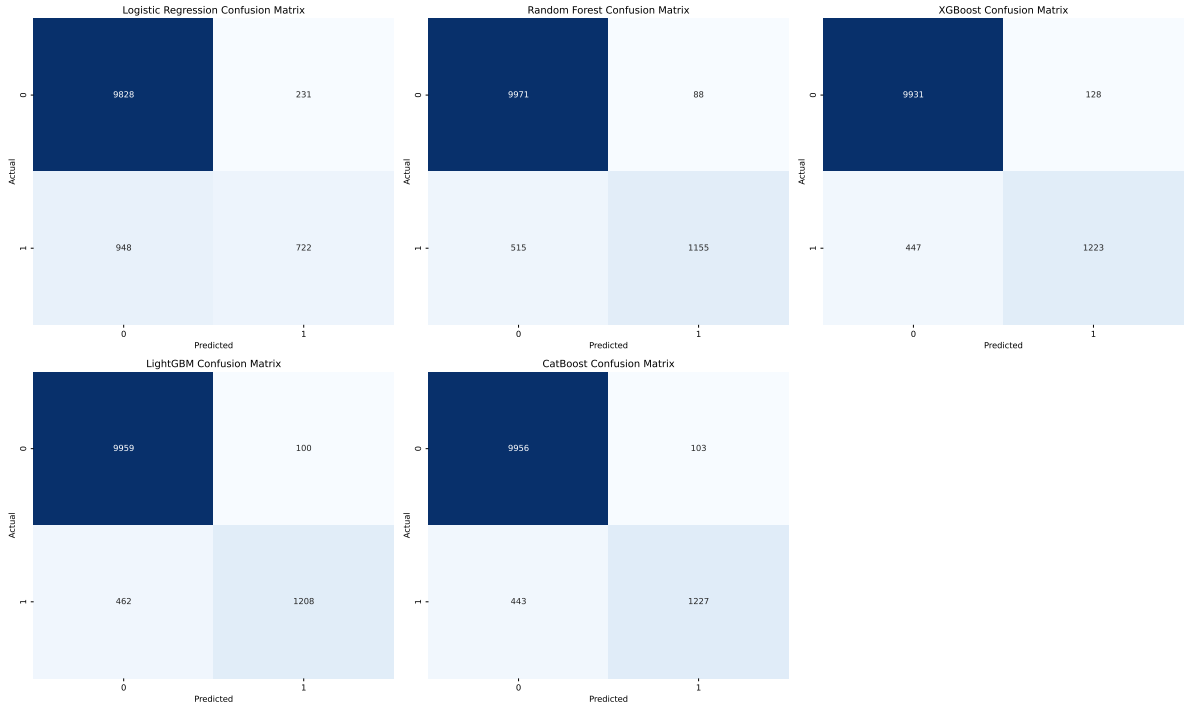
Classification Report for Logistic Regression:

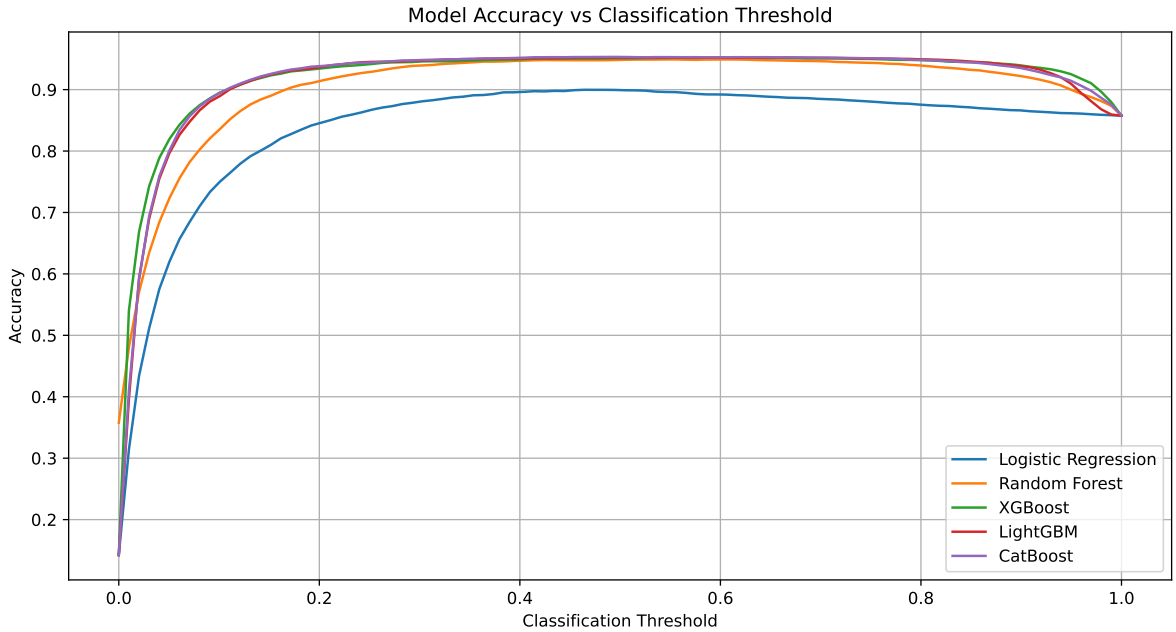
	precision	recall	f1-score	support
0	0.91	0.98	0.94	10059
1	0.76	0.43	0.55	1670
accuracy			0.90	11729
macro avg	0.83	0.70	0.75	11729
weighted avg	0.89	0.90	0.89	11729

Classification Report for Random Forest:

	precision	recall	f1-score	support

```
              0       0.95      0.99      0.97     10059
              1       0.93      0.69      0.79      1670

       accuracy                           0.95     11729
      macro avg       0.94      0.84      0.88     11729
   weighted avg       0.95      0.95      0.95     11729


Classification Report for XGBoost:
                 precision    recall  f1-score   support

              0       0.96      0.99      0.97     10059
              1       0.91      0.73      0.81      1670

       accuracy                           0.95     11729
      macro avg       0.93      0.86      0.89     11729
   weighted avg       0.95      0.95      0.95     11729


Classification Report for LightGBM:
                 precision    recall  f1-score   support

              0       0.96      0.99      0.97     10059
              1       0.92      0.72      0.81      1670

       accuracy                           0.95     11729
      macro avg       0.94      0.86      0.89     11729
   weighted avg       0.95      0.95      0.95     11729


Classification Report for CatBoost:
                 precision    recall  f1-score   support

              0       0.96      0.99      0.97     10059
              1       0.92      0.73      0.82      1670

       accuracy                           0.95     11729
      macro avg       0.94      0.86      0.90     11729
   weighted avg       0.95      0.95      0.95     11729
```

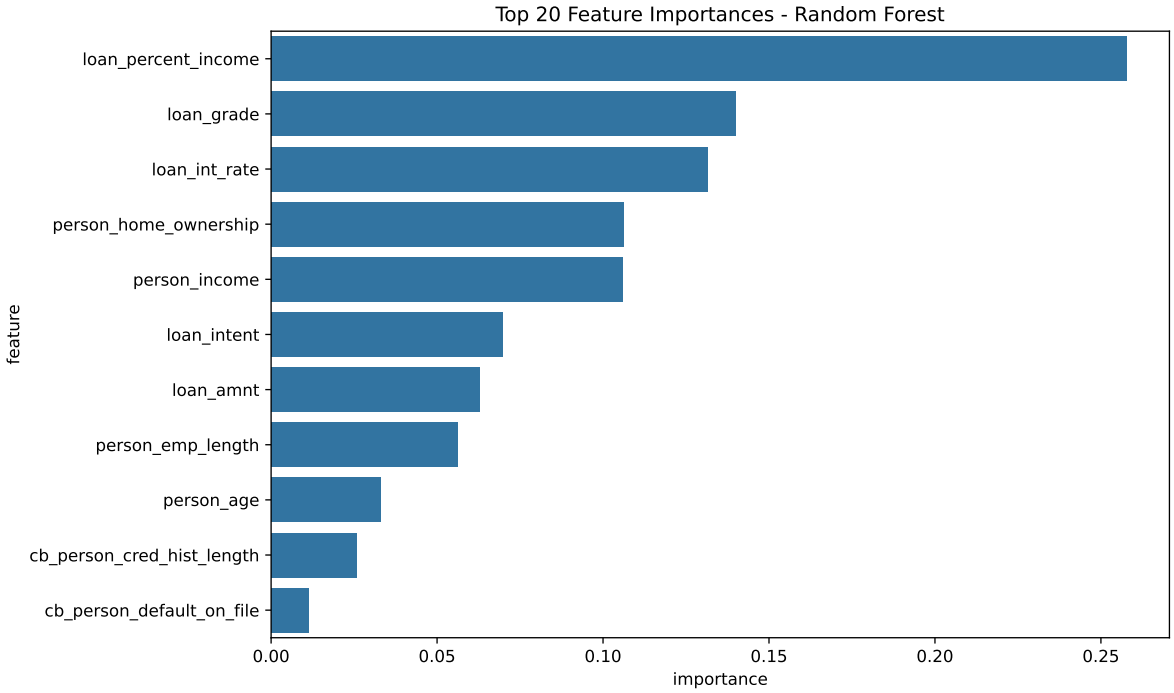


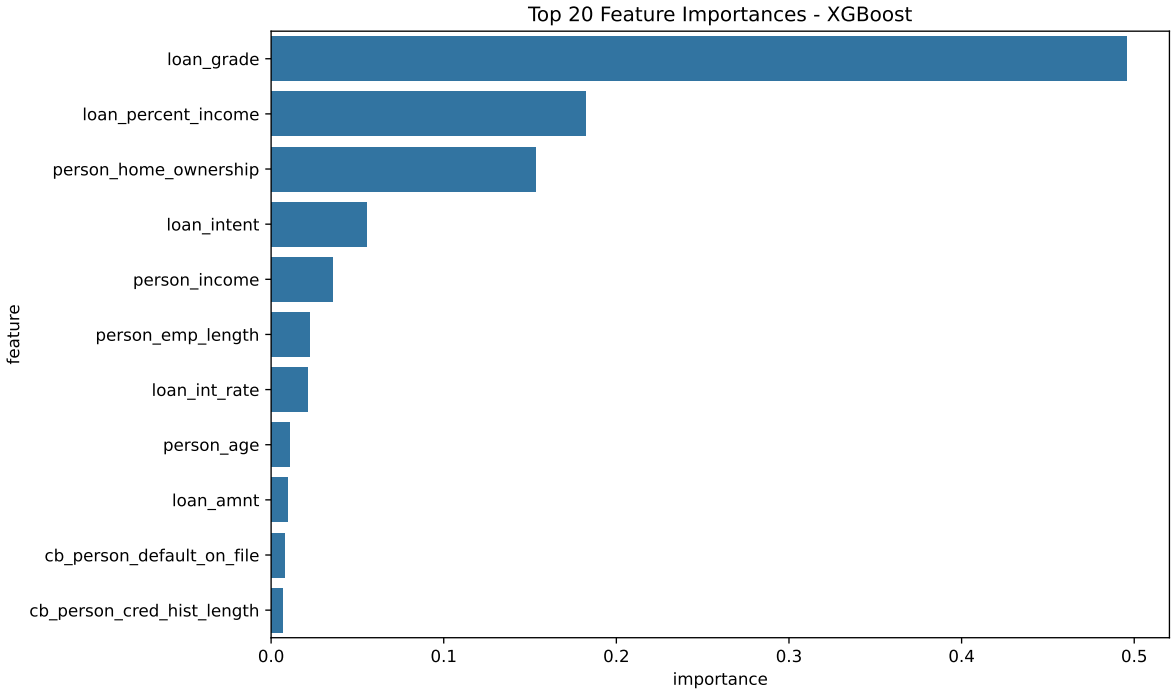


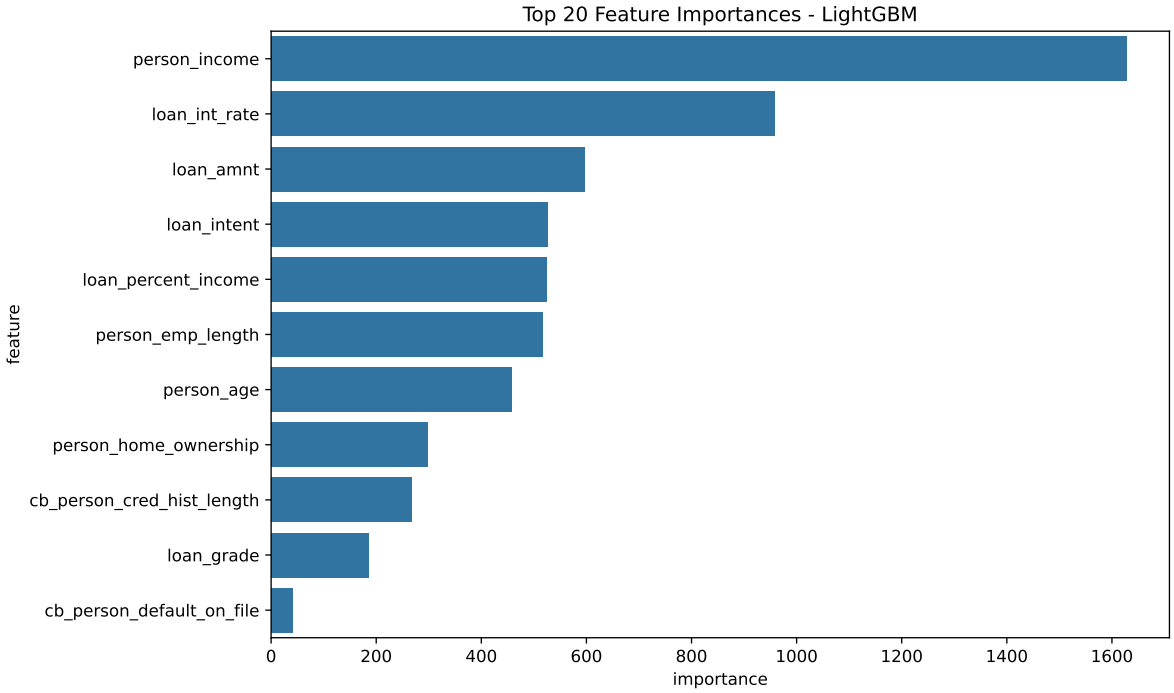
Optimal Classification Thresholds:
Logistic Regression:

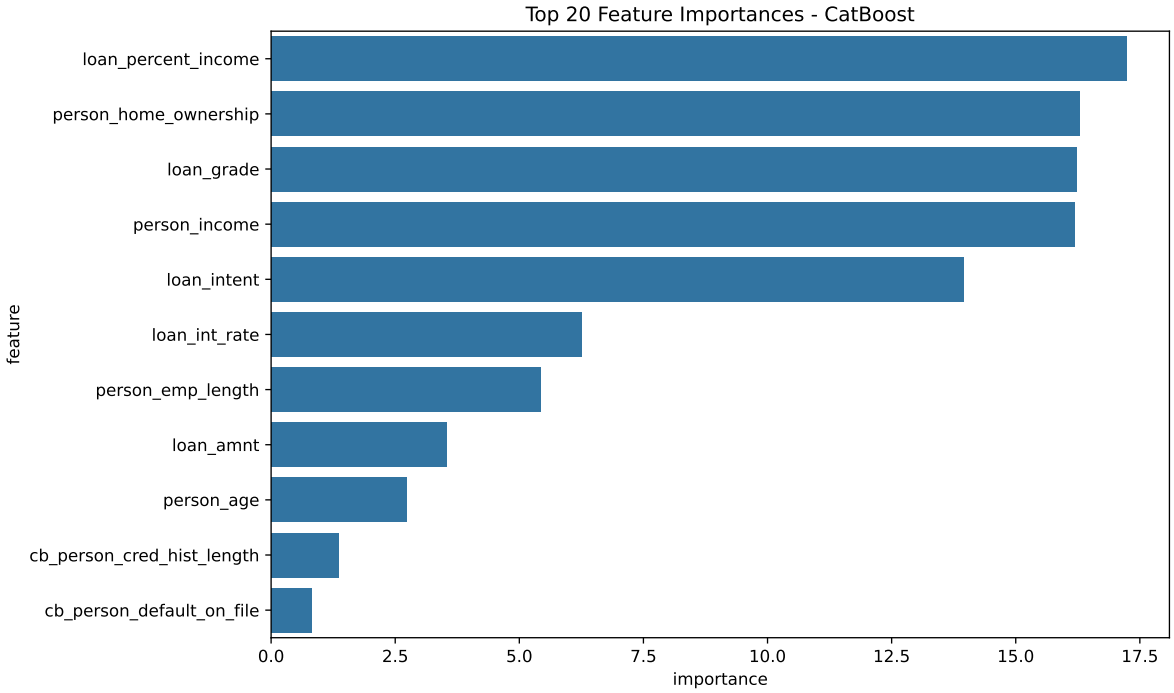
```
  Optimal Threshold: 0.465
  Optimal Accuracy: 0.900
  Confusion Matrix with Optimal Threshold:
[[9769  290]
 [ 886  784]]


Random Forest:
  Optimal Threshold: 0.545
  Optimal Accuracy: 0.949
  Confusion Matrix with Optimal Threshold:
[[9991   68]
 [ 529 1141]]


XGBoost:
  Optimal Threshold: 0.616
  Optimal Accuracy: 0.952
  Confusion Matrix with Optimal Threshold:
[[9974   85]
 [ 477 1193]]


LightGBM:
  Optimal Threshold: 0.556
  Optimal Accuracy: 0.952
  Confusion Matrix with Optimal Threshold:
[[9970   89]
 [ 470 1200]]


CatBoost:
  Optimal Threshold: 0.495
  Optimal Accuracy: 0.953
  Confusion Matrix with Optimal Threshold:
[[9952  107]
 [ 441 1229]]



Tuning Logistic Regression...
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters for Logistic Regression:
```

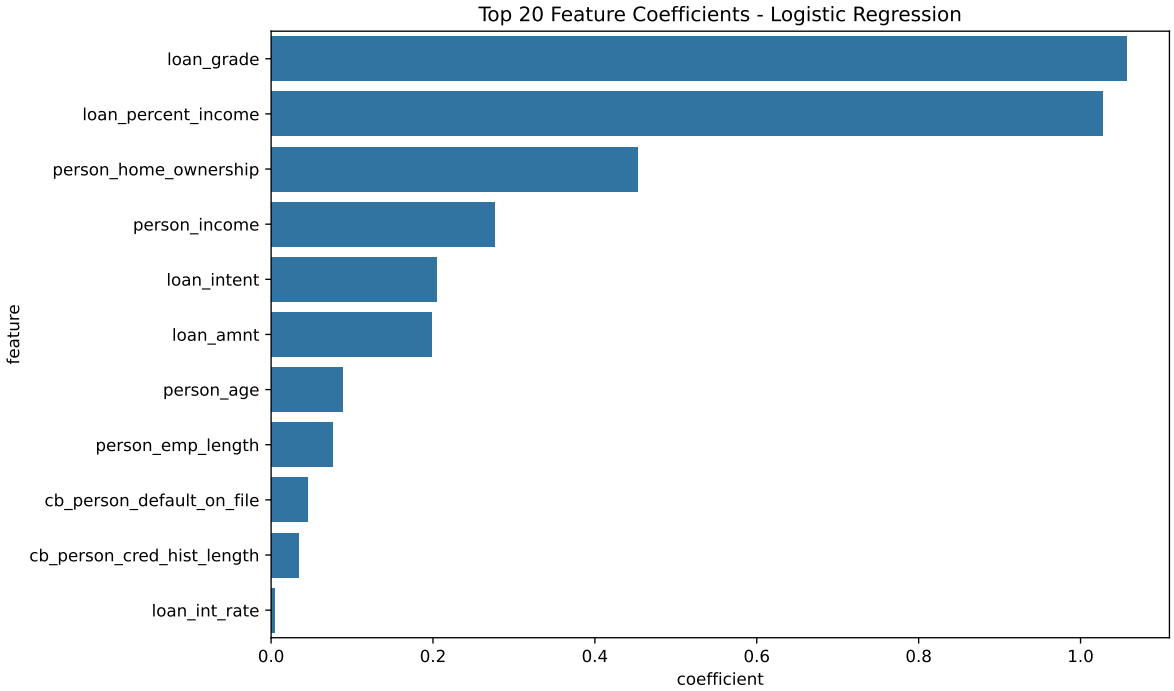
```
{'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
Best ROC-AUC: 0.8809

Tuning Random Forest...
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters for Random Forest:
{'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 200}
Best ROC-AUC: 0.9403

Tuning XGBoost...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for XGBoost:
{'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 200}
Best ROC-AUC: 0.9549

Tuning LightGBM...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[LightGBM] [Info] Number of positive: 8350, number of negative: 50295
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.0017
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 887
[LightGBM] [Info] Number of data points in the train set: 58645, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.142382 -> initscore=-1.795644
[LightGBM] [Info] Start training from score -1.795644
Best parameters for LightGBM:
{'learning_rate': 0.1, 'n_estimators': 200, 'num_leaves': 31}
Best ROC-AUC: 0.9571

Tuning CatBoost...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for CatBoost:
{'depth': 8, 'iterations': 200, 'learning_rate': 0.1}
Best ROC-AUC: 0.9512
```

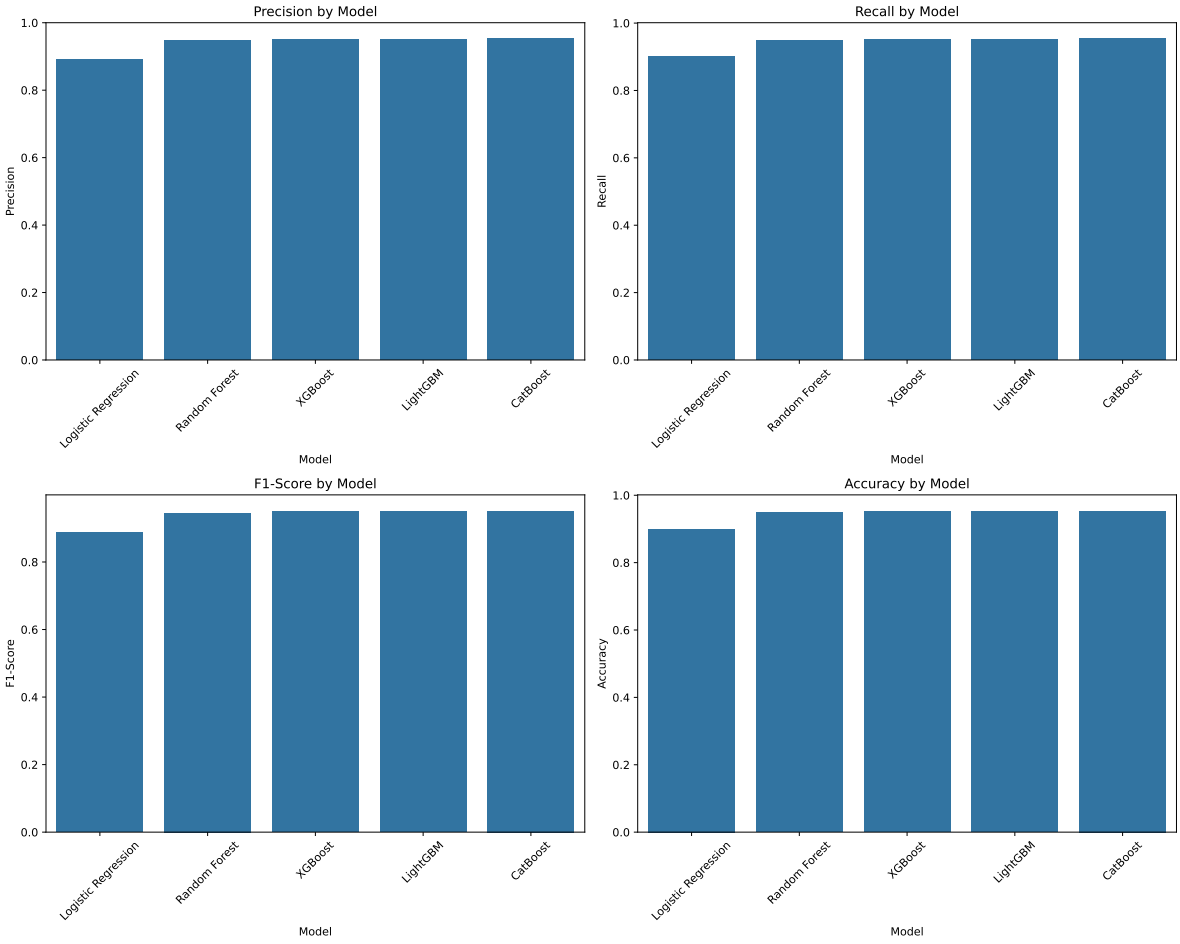

Top 20 Feature Importances - Random Forest
loan_percent_income
loan_grade
loan_int_rate
person_home_ownership
person_income
loan_intent
loan_amnt
person_emp_length
person_age
cb_person_cred_hist_length
cb_person_default_on_file
feature
0.00
0.05
0.10
0.15
0.20
0.25
importance

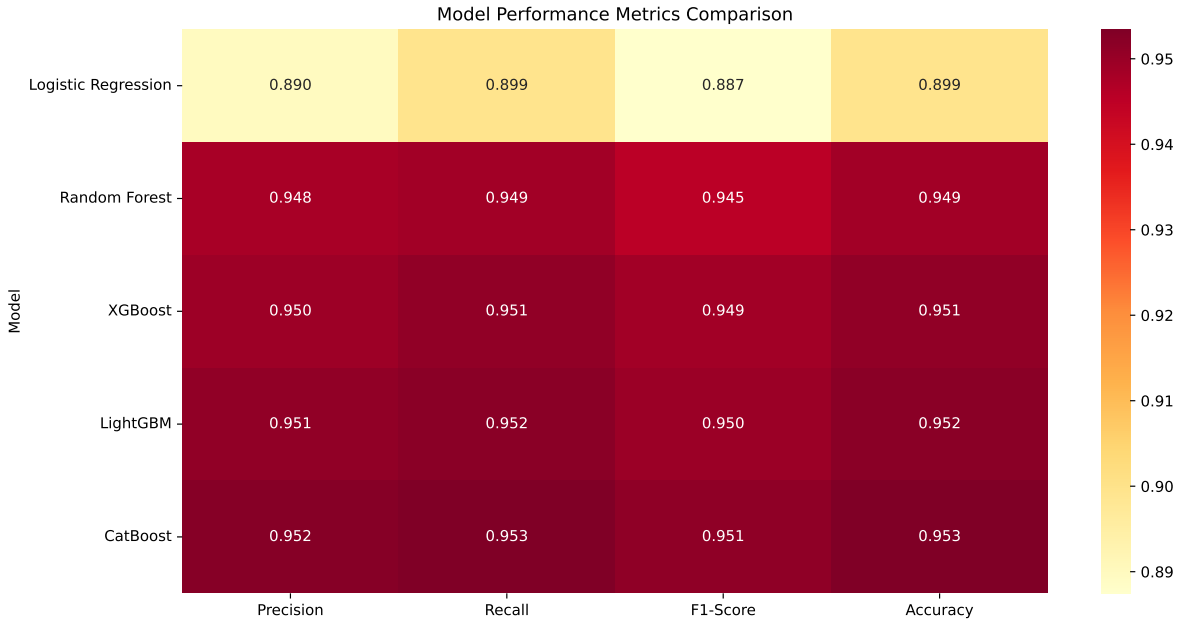

Top 20 Feature Importances - XGBoost
loan_grade
loan_percent_income
person_home_ownership
loan_intent
person_income
person_emp_length
loan_int_rate
person_age
loan_amnt
cb_person_default_on_file
cb_person_cred_hist_length
feature
0.0
0.1
0.2
0.3
0.4
0.5
importance

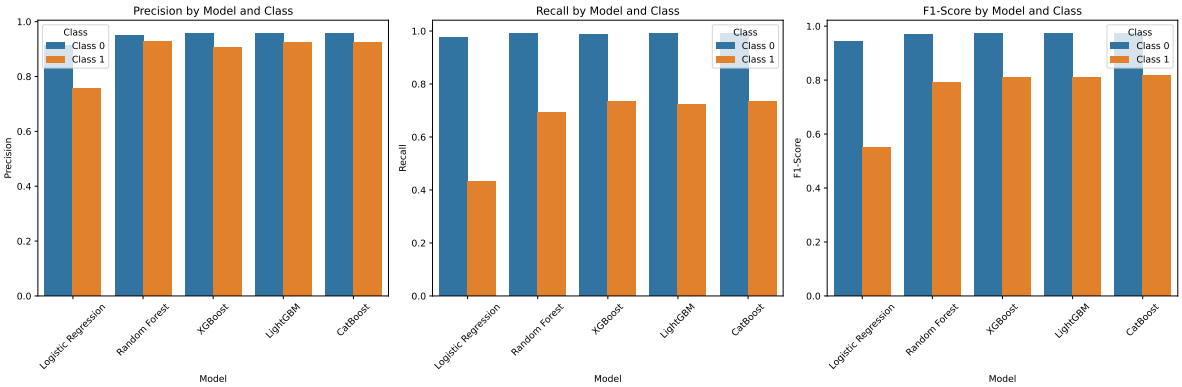

Top 20 Feature Importances - LightGBM
person_income
loan_int_rate
loan_amnt
loan_intent
loan_percent_income
person_emp_length
person_age
person_home_ownership
cb_person_cred_hist_length
loan_grade
cb_person_default_on_file
feature
0
200
400
600
800
1000
1200
1400
1600
importance

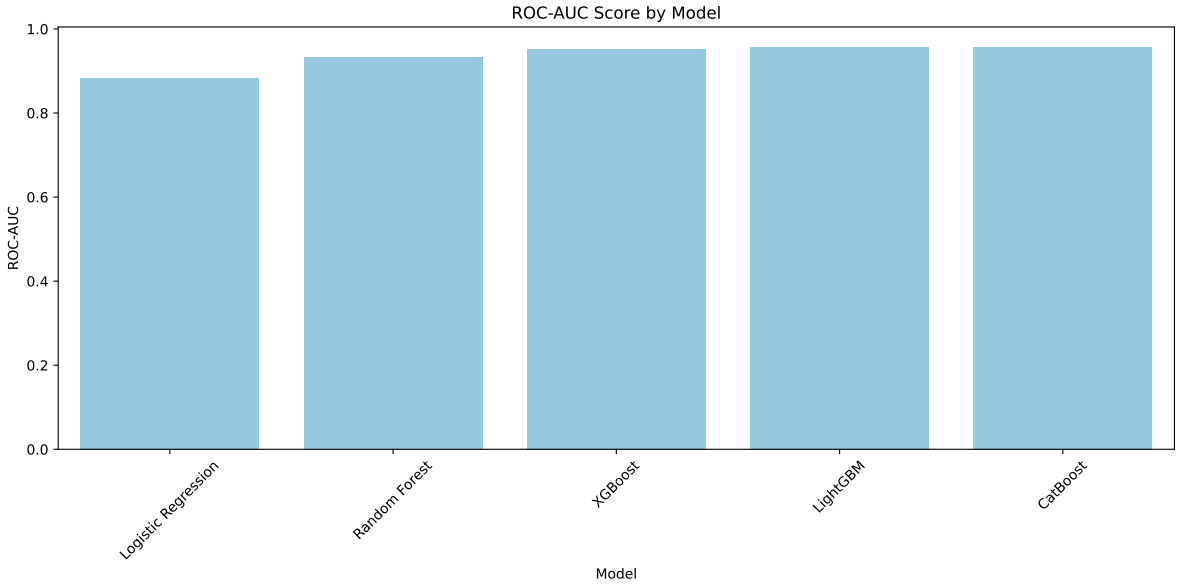

Top 20 Feature Importances - CatBoost
loan_percent_income
person_home_ownership
loan_grade
person_income
loan_intent
loan_int_rate
person_emp_length
loan_amnt
person_age
cb_person_cred_hist_length
cb_person_default_on_file
feature
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
importance

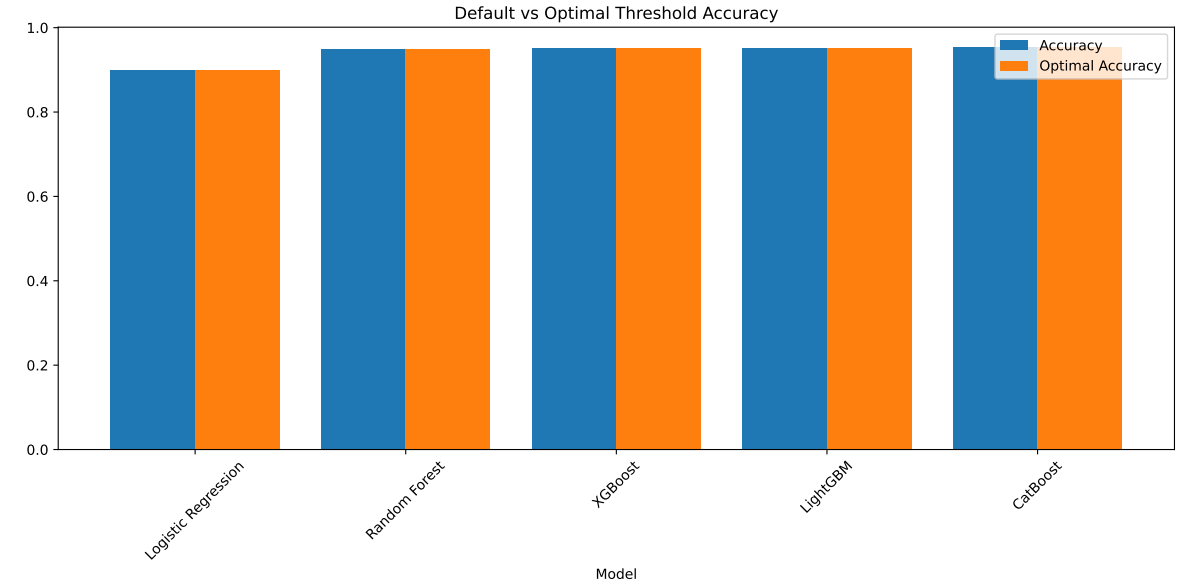

Top 20 Feature Coefficients - Logistic Regression
loan_grade
loan_percent_income
person_home_ownership
person_income
loan_intent
loan_amnt
person_age
person_emp_length
cb_person_default_on_file
cb_person_cred_hist_length
loan_int_rate
feature
0.0
0.2
0.4
0.6
0.8
1.0
coefficient

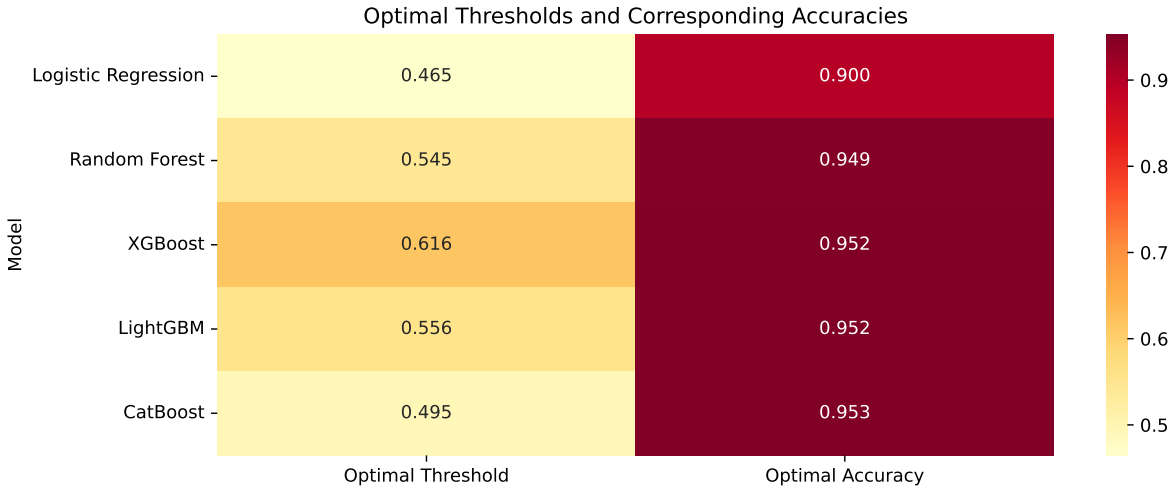

Precision by Model
Precision
Recall by Model
Recall
F1-Score by Model
F1-Score
Accuracy by Model
Accuracy
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression
Random Forest
XGBoost
LightGBM
CatBoost
Model

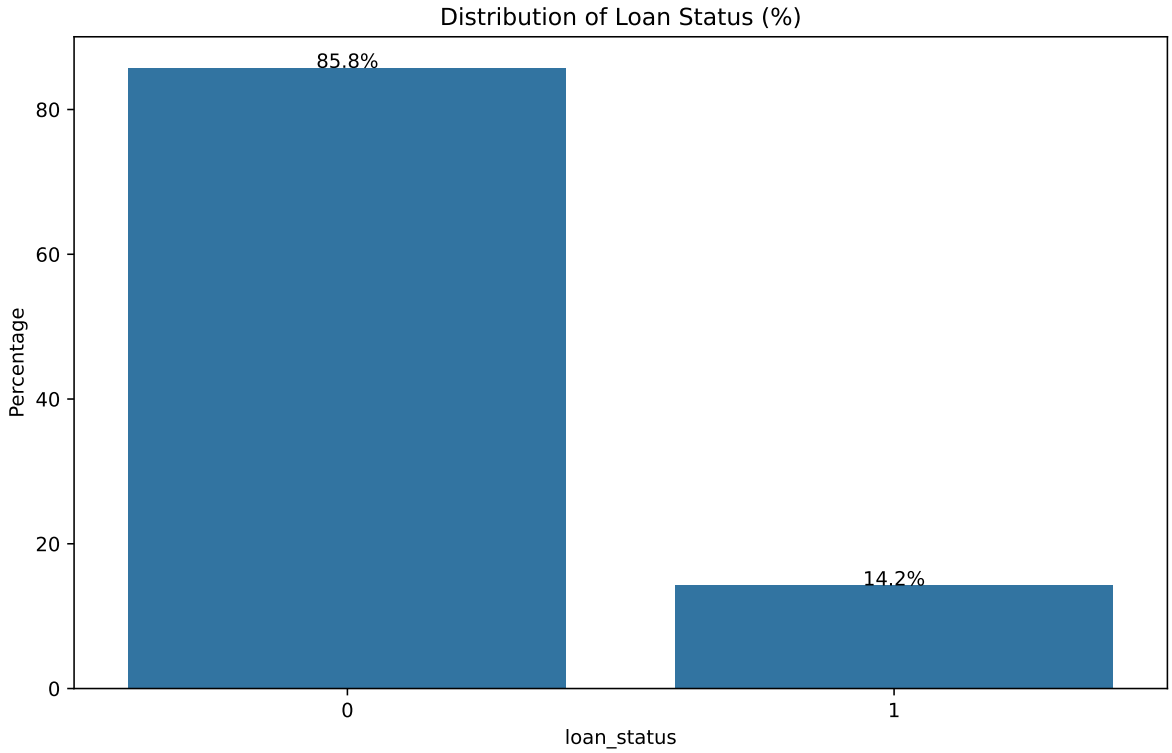

Model Performance Metrics Comparison
Logistic Regression
Random Forest
XGBoost
LightGBM
CatBoost
Model
0.890
0.899
0.887
0.899
0.948
0.949
0.945
0.949
0.950
0.951
0.949
0.951
0.951
0.952
0.950
0.952
0.952
0.953
0.951
0.953
Precision
Recall
F1-Score
Accuracy
0.95
0.94
0.93
0.92
0.91
0.90
0.89

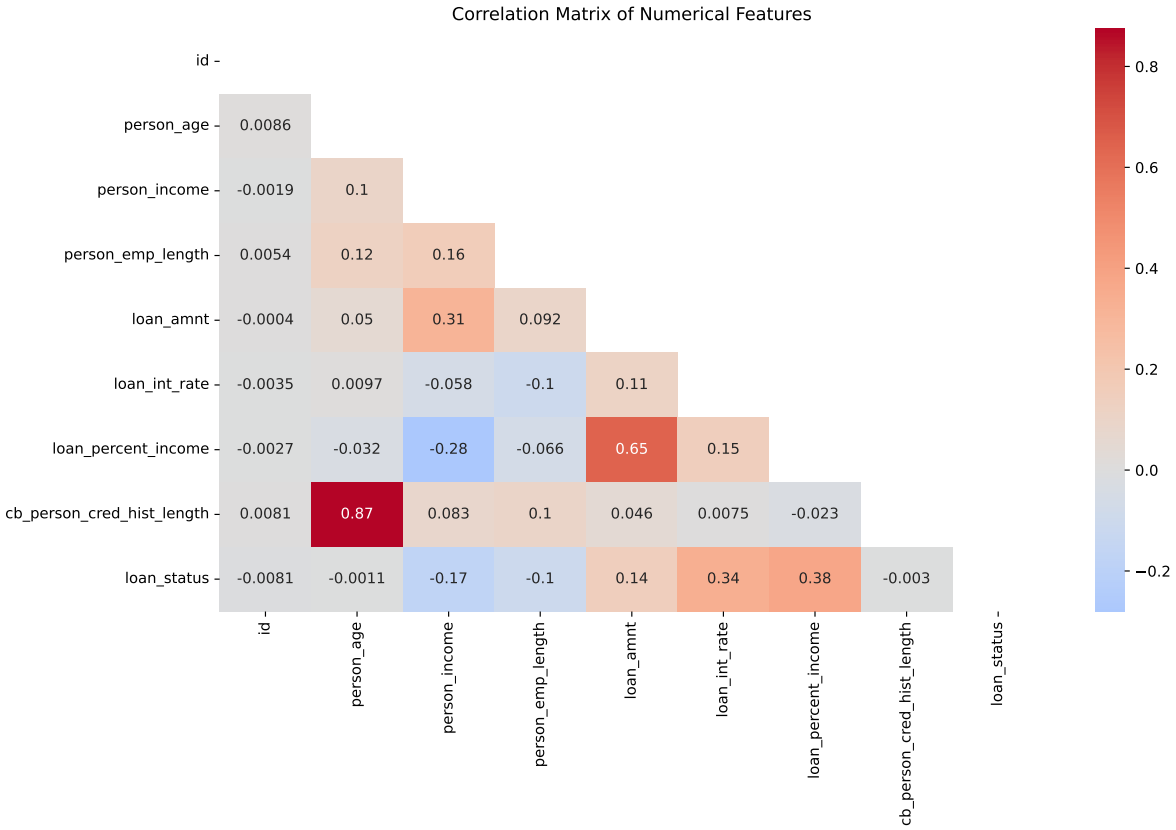

Precision by Model and Class
Recall by Model and Class
F1-Score by Model and Class
Class
Class 0
Class 1
Precision
Recall
F1-Score
Model
Logistic Regression
Random Forest
XGBoost
LightGBM
CatBoost
0.0
0.2
0.4
0.6
0.8
1.0

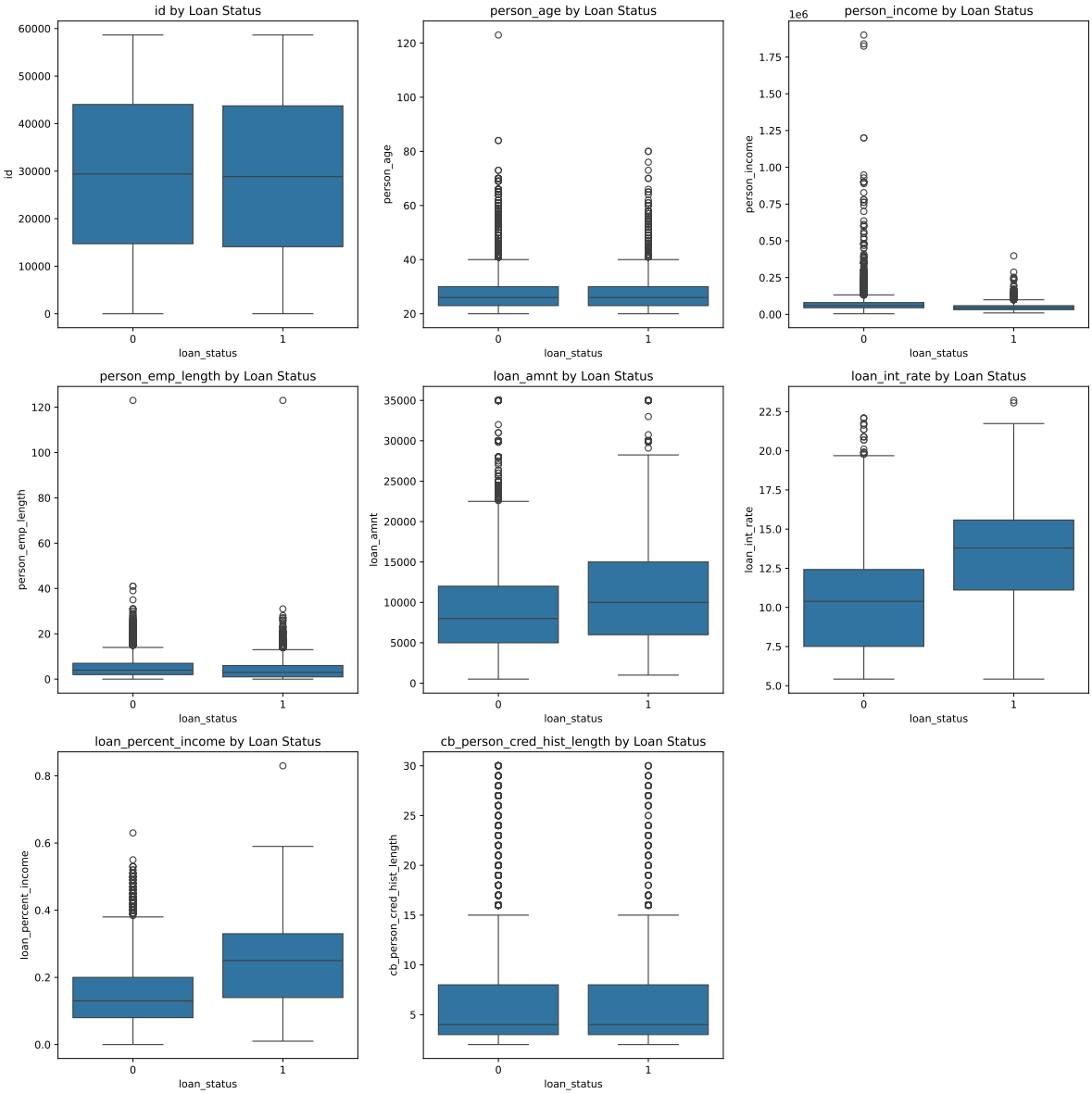

ROC-AUC Score by Model
ROC-AUC
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
Random Forest
XGBoost
LightGBM
CatBoost
Model

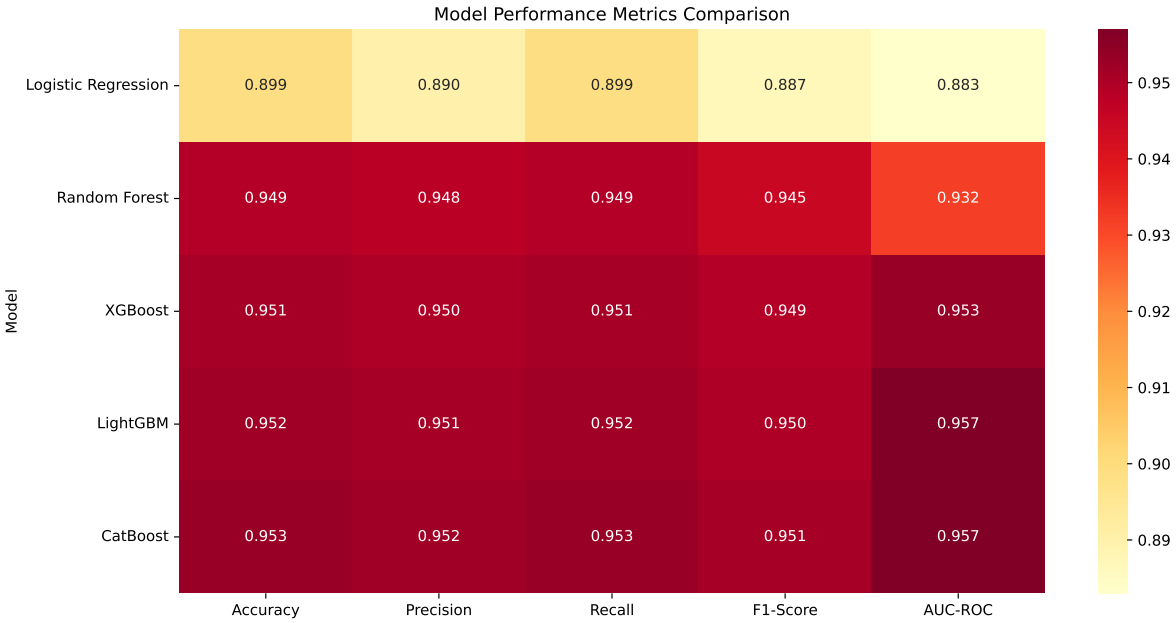

Default vs Optimal Threshold Accuracy
Accuracy
Optimal Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
Random Forest
XGBoost
LightGBM
CatBoost
Model

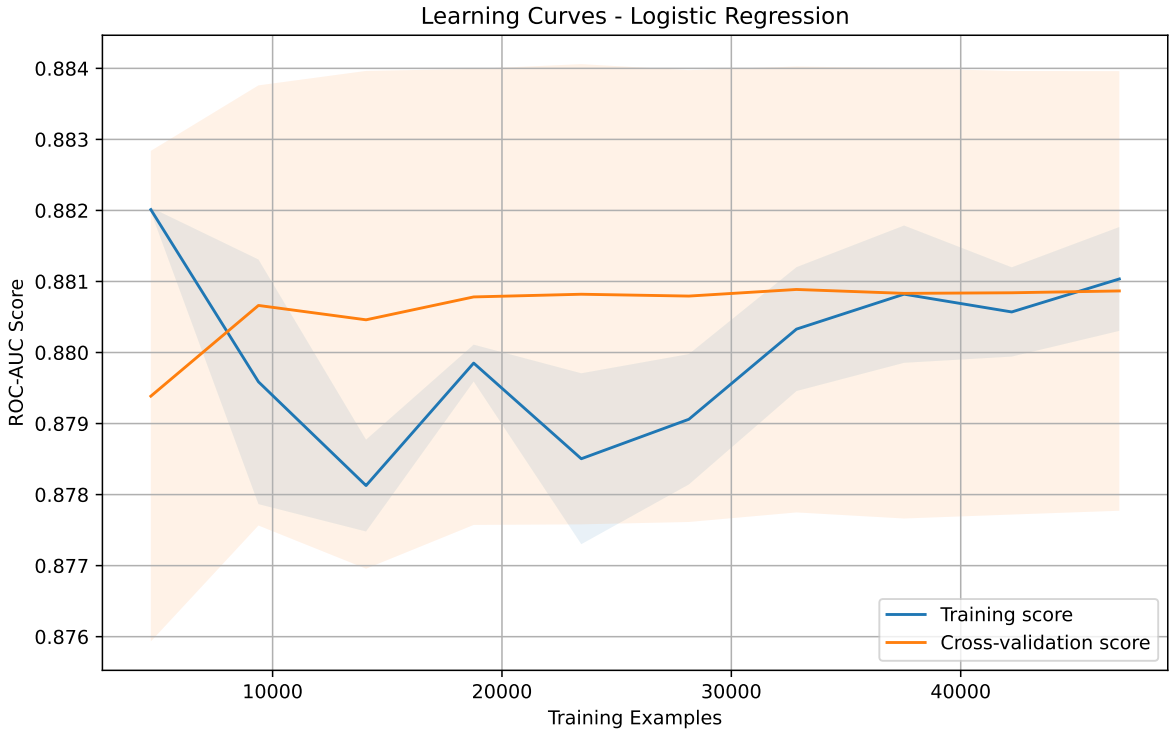

Optimal Thresholds and Corresponding Accuracies
Logistic Regression
Random Forest
XGBoost
LightGBM
CatBoost
Model
0.465
0.900
0.545
0.949
0.616
0.952
0.556
0.952
0.495
0.953
Optimal Threshold
Optimal Accuracy
0.9
0.8
0.7
0.6
0.5

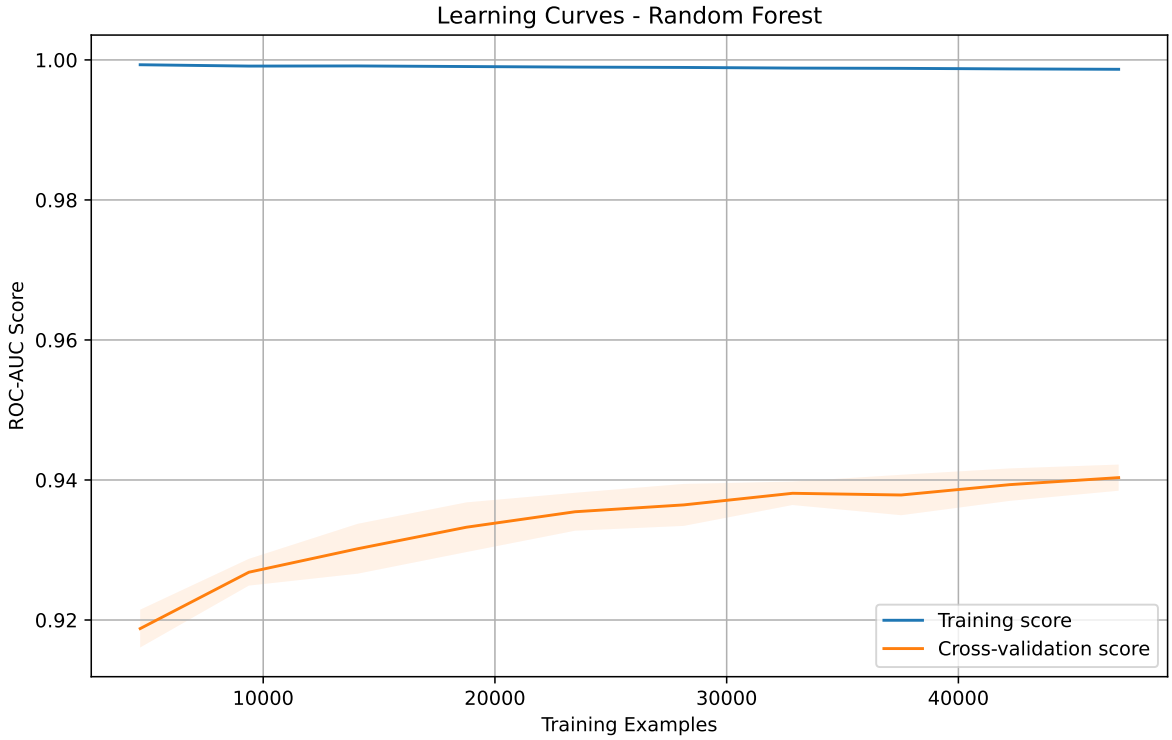

Distribution of Loan Status (%)
85.8%
14.2%
Percentage
0
20
40
60
80
0
1
loan_status

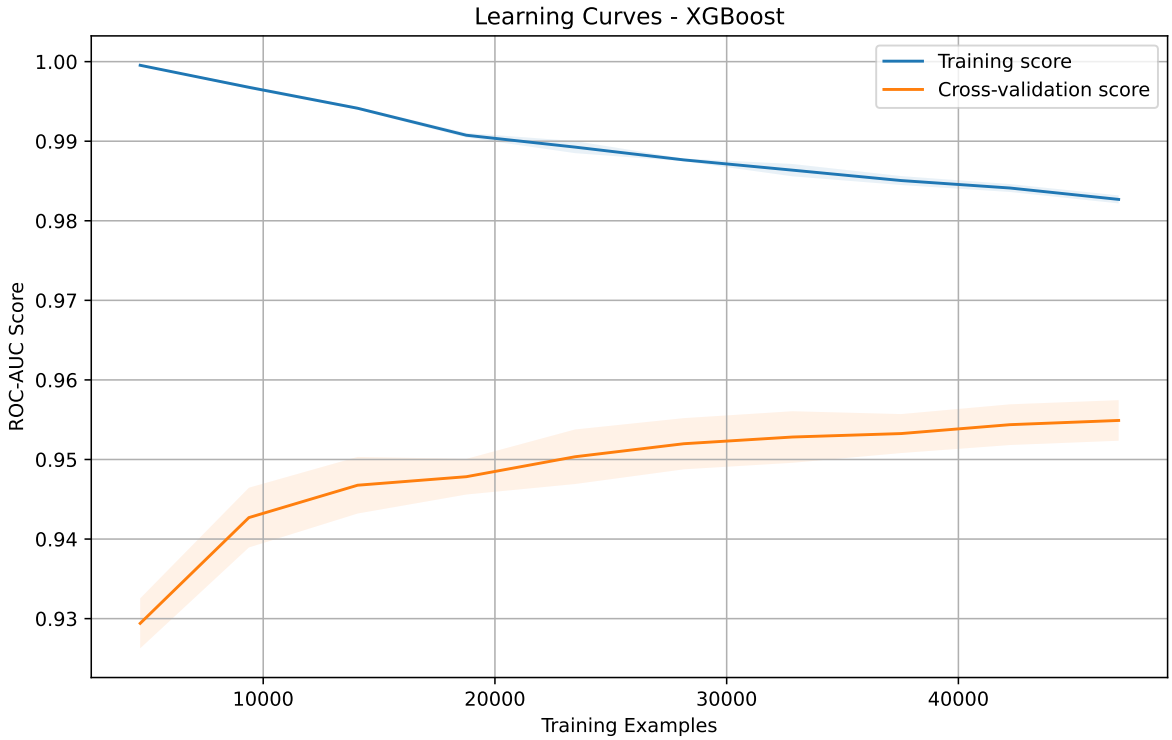

Correlation Matrix of Numerical Features
id
person_age
person_income
person_emp_length
loan_amnt
loan_int_rate
loan_percent_income
cb_person_cred_hist_length
loan_status
0.0086
-0.0019 0.1
0.0054 0.12 0.16
-0.0004 0.05 0.31 0.092
-0.0035 0.0097 -0.058 -0.1 0.11
-0.0027 -0.032 -0.28 -0.066 0.65 0.15
0.0081 0.87 0.083 0.1 0.046 0.0075 -0.023
-0.0081 -0.0011 -0.17 -0.1 0.14 0.34 0.38 -0.003
0.8
0.6
0.4
0.2
0.0
−0.2

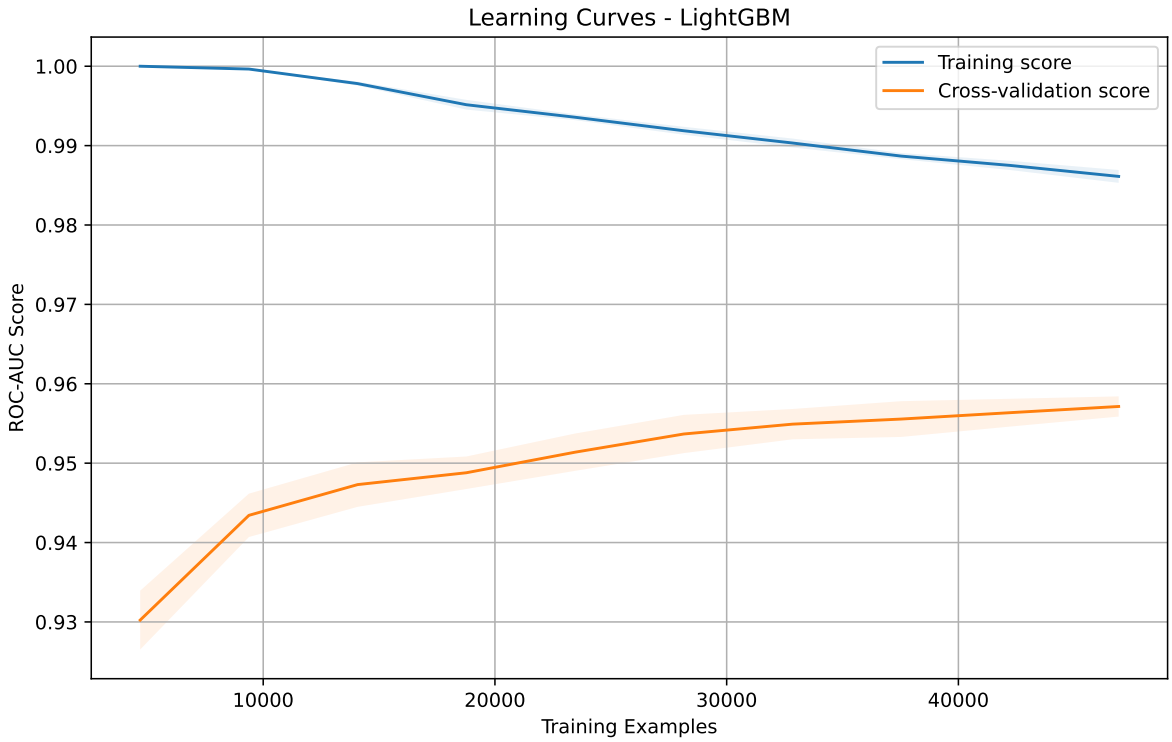

id by Loan Status
id
loan_status
person_age by Loan Status
person_age
loan_status
person_income by Loan Status
1e6
person_income
loan_status
person_emp_length by Loan Status
person_emp_length
loan_status
loan_amnt by Loan Status
loan_amnt
loan_status
loan_int_rate by Loan Status
loan_int_rate
loan_status
loan_percent_income by Loan Status
loan_percent_income
loan_status
cb_person_cred_hist_length by Loan Status
cb_person_cred_hist_length
loan_status

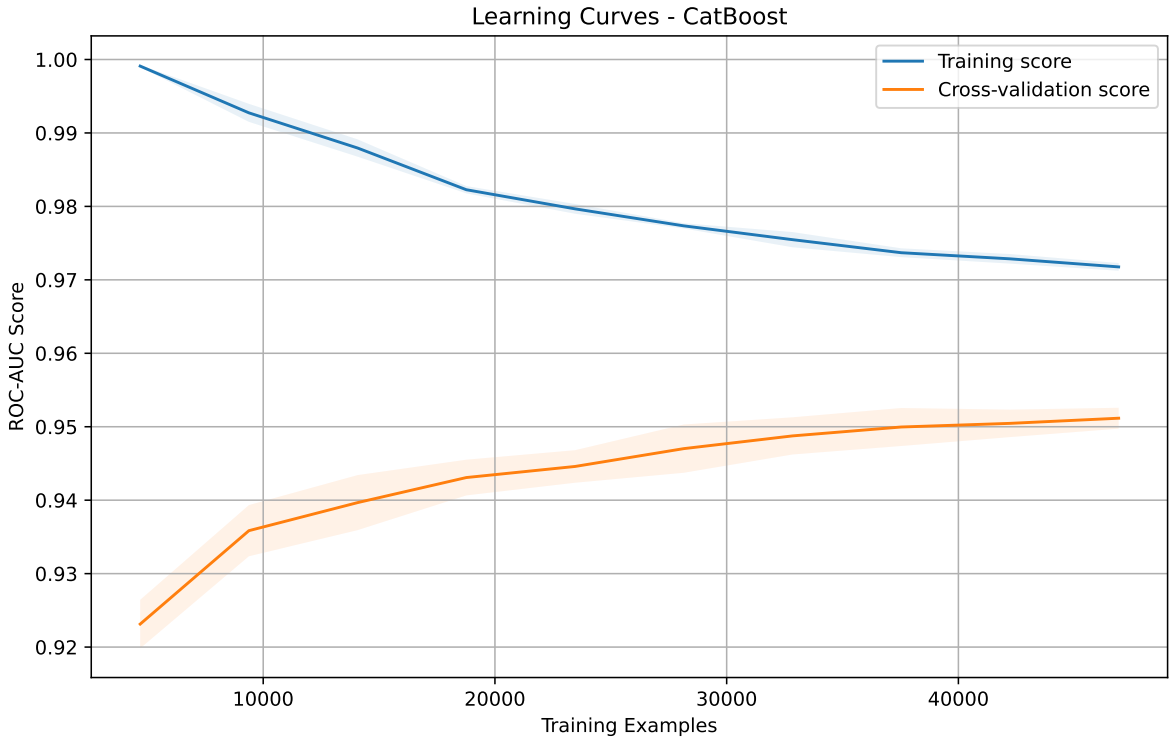

Model Performance Metrics Comparison
Logistic Regression
Random Forest
XGBoost
LightGBM
CatBoost
Model
0.899
0.890
0.899
0.887
0.883
0.949
0.948
0.949
0.945
0.932
0.951
0.950
0.951
0.949
0.953
0.952
0.951
0.952
0.950
0.957
0.953
0.952
0.953
0.951
0.957
Accuracy
Precision
Recall
F1-Score
AUC-ROC
0.95
0.94
0.93
0.92
0.91
0.90
0.89


Learning Curves - Logistic Regression
ROC-AUC Score
0.884
0.883
0.882
0.881
0.880
0.879
0.878
0.877
0.876
10000
20000
30000
40000
Training Examples
Training score
Cross-validation score


Learning Curves - Random Forest
1.00
0.98
0.96
0.94
0.92
ROC-AUC Score
Training score
Cross-validation score
10000
20000
30000
40000
Training Examples

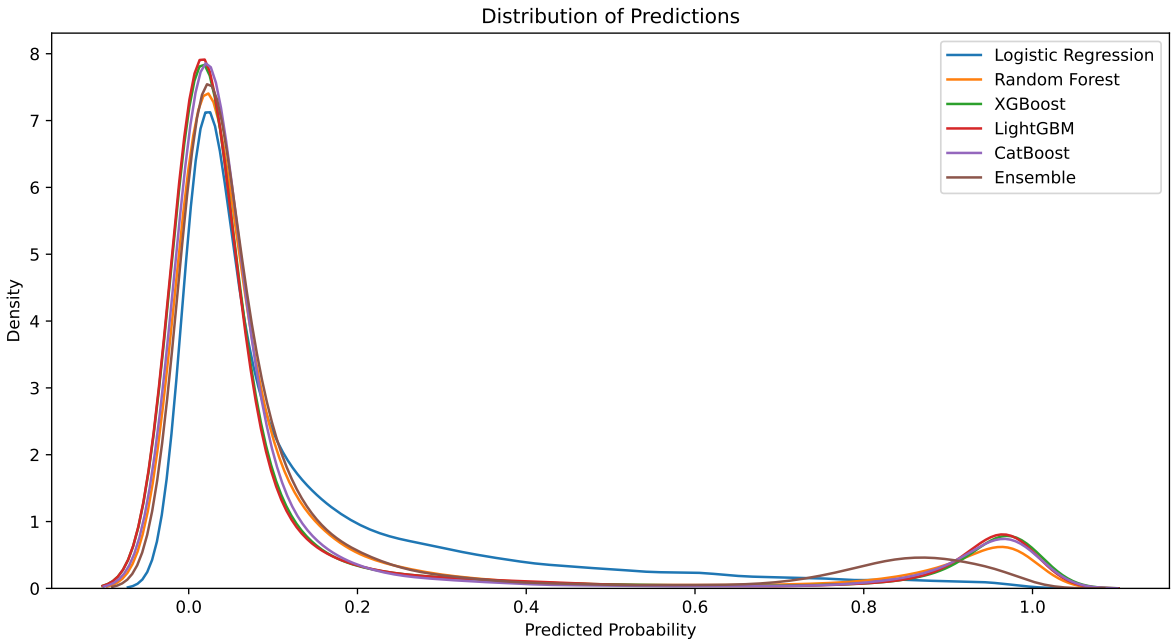

Learning Curves - XGBoost
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
ROC-AUC Score
Training score
Cross-validation score
10000
20000
30000
40000
Training Examples


Learning Curves - LightGBM
Training score
Cross-validation score
ROC-AUC Score
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
10000
20000
30000
40000
Training Examples


Learning Curves - CatBoost
Training score
Cross-validation score
ROC-AUC Score
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
10000
20000
30000
40000
Training Examples

Final Predictions and Ensemble

```
# Generate predictions for each model
predictions = {}
for name, model in tuned_models.items():
    predictions[name] = model.predict_proba(X_test_scaled)[:, 1]

# Create ensemble predictions (simple average)
ensemble_predictions = np.mean([pred for pred in predictions.values()], axis=0)

# Create submission files
for name, pred in predictions.items():
    submission = pd.DataFrame({
        'id': test_df['id'],
        'loan_status': pred
    })
    submission.to_csv(
        f'submission_{name.lower().replace(" ", "_")}.csv', index=False)

# Create ensemble submission
ensemble_submission = pd.DataFrame({
    'id': test_df['id'],
    'loan_status': ensemble_predictions
})
ensemble_submission.to_csv('submission_ensemble.csv', index=False)

# Compare prediction distributions
plt.figure(figsize=(12, 6))
for name, pred in predictions.items():
    sns.kdeplot(pred, label=name)
sns.kdeplot(ensemble_predictions, label='Ensemble')
plt.xlabel('Predicted Probability')
plt.ylabel('Density')
plt.title('Distribution of Predictions')
plt.legend()
plt.show()

# Print final model comparison
print("\nFinal Model Comparison:")
for name, (model, results) in model_results.items():
    val_auc = roc_auc_score(results[2], results[3])
    print(f"{name} Validation AUC: {val_auc:.4f}")
```




```
Final Model Comparison:
Logistic Regression Validation AUC: 0.8833
Random Forest Validation AUC: 0.9317
XGBoost Validation AUC: 0.9529
LightGBM Validation AUC: 0.9569
CatBoost Validation AUC: 0.9565
```

Conclusion

This analysis implements a comprehensive approach to the Loan Approval Prediction task, including:

1. Data preprocessing with proper handling of missing values and categorical variables
2. Implementation of multiple classification models:
    - Logistic Regression (baseline)
    - Random Forest
    - XGBoost
    - LightGBM
    - CatBoost

3. Hyperparameter tuning using GridSearchCV
4. Model performance visualization using ROC curves
5. Feature importance analysis for each model
6. Ensemble prediction using model averaging

Key findings: 1. [Add your specific findings about model performance] 2. [Add insights about most important features] 3. [Add insights about model strengths/weaknesses]

Areas for improvement: 1. Feature engineering (creating interaction terms, polynomial features) 2. More advanced ensemble methods (stacking, weighted averaging) 3. More extensive hyperparameter tuning 4. Cross-validation with different random seeds 5. Handling class imbalance if present 6. Model calibration analysis