May 5, 2025

Jaxon Adams

C964: Computer Science Capstone

Task 2 parts A, B, C, and D

Part A: Letter of Transmittal

May 5, 2025

Johnathan Doe

Origination Solutions, Inc.

123 Rocky Road, Anytown USA, 12345

Subject: Proposal for Predictive Loan Default Classification Application

Dear Mr. Doe,

I am pleased to submit this proposal to develop a machine learning application that predicts the likelihood of loan default using historical Lending Club data. This application addresses the pressing challenge of minimizing financial risk by providing proactive insight into high-risk borrowers.

The proposed solution is a predictive web application powered by a Random Forest classifier trained on a large dataset of historical loan records. It will allow Origination Solutions, Inc. to identify loan applicants with a higher probability of default before approval. The application will expose a user-friendly interface and backend API, enabling seamless integration into internal workflows. In addition, a brief user guide and documentation will accompany the final deployment.

The system will enhance decision-making accuracy, reduce credit losses, and increase operational efficiency. Predictions are based on key borrower metrics such as credit score, loan purpose, and annual income, making the insights actionable and relevant. The machine learning model is expected to balance recall and precision and show high feature interpretability effectively.

The data will be sourced from a public Kaggle dataset, which reflects over a decade of anonymized loan outcomes. It will be cleaned, preprocessed, and ethically managed to meet all legal and organizational data standards. The solution will follow industry-standard CRISP-DM methodology and be deployed to AWS S3 for storage and Render for frontend and backend hosting. No personal or sensitive data will be handled, eliminating significant ethical risks.

The estimated cost is under $1,500, covering hosting, development time, and testing. The project is estimated to be completed within 4 weeks, with weekly milestones including model training, API development, interface design, and deployment testing.

As a software engineer with over five years of experience in machine learning systems and cloud-native application deployment, I am confident this solution will deliver real, measurable value to Origination Solutions, Inc. I look forward to your support in moving this project forward.

Sincerely,

Jaxon Adams

Jaxon Adams, Software Engineer

Part B: Project Proposal Plan

Project Summary

This project will address the problem of effectively assessing credit risk by predicting the likelihood of loan default before funds are issued. As our organization processes a high volume of personal loan applications, we need a data-driven tool to improve decision-making during the loan underwriting process. Our current risk assessment approach is rule-based and lacks modern machine learning systems' adaptability and predictive accuracy.

To meet our needs, this project will deliver the following:

- A trained machine learning model can predict loan default based on borrower characteristics and loan parameters.
- A Python-based application pipeline that automates data preprocessing, model training, and evaluation.
- Visualizations for performance metrics, feature importance, and threshold tuning.
- A serialized model file that can be reused in production systems or integrated into a larger loan decision engine.
- A simple API implemented with Flask to expose model prediction and provide relevant visualizations.
- A basic frontend web application using HTML, CSS, and vanilla JavaScript to consume the Flask API and provide an intuitive user interface for requesting model predictions.
- A user guide documenting how to operate the application, interpret the outputs, and retrain the model on updated data.

The application will help us improve the accuracy of our loan approval decisions, reduce risk exposure, and optimize return on investment by identifying potentially high-risk applicants before issuing funds. In doing so, this project will modernize and automate part of our credit risk evaluation process.

Data Summary

The data for this project will come from Lending Club's publicly available historical loan performance dataset, hosted on Kaggle under the name "Lending Club Loan Data 2007-2020Q3". This dataset contains over a decade of anonymized loan records, including borrower credit profiles, loan terms, and repayment outcomes.

The raw data will be cleaned and processed through the following stages:

- **Design Phase:** Features relevant to predicting loan default will be selected, and a binary classification target will be defined based on loan status.
- **Development Phase:** Categorical data will be encoded using ordinal encoding, missing values will be imputed using appropriate strategies, and SMOTE will be applied to address class imbalance.
- **Maintenance Phase:** The model will be serializable and reusable, allowing for retraining with updated data. All preprocessing steps will be embedded in a pipeline to ensure consistency.

The dataset meets this project’s needs by offering a wide variety of real-world borrowers and loan features along with labeled outcomes, making it ideal for supervised learning. Common anomalies such as missing data and skewed class distributions will be handled using industry-standard practices. Outliers will be retained unless they represent data corruption, in which case they will be filtered out during preprocessing.

There are no ethical or legal concerns associated with the use of this data, as it is anonymized and publicly available for educational and research purposes under Kaggle’s terms of use.

Implementation

The project will follow the Cross-Industry Standard Process for Data Mining methodology, or CRISP-DM. This methodology is widely used in the industry for data science and machine learning projects. This approach involves the following phases: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. To follow this practice, the implementation plan will include the following steps:

1. **Requirement Gathering and Business Understanding:** Time will be spent identifying ways to maximize the positive impact of a machine learning solution on the current loan origination process.
2. **Data Collection and Initial Exploration:** The Lending Club loans dataset will be downloaded from Kaggle and analyzed for trends, correlated features, and any fields that might require specific formatting.
3. **Data Cleaning and Preprocessing:** Initial scripts will be written to clean the data according to the exploratory analysis performed previously, including general preprocessing steps such as encoding categorical features.
4. **Model Selection and Training:** The Random Forest classifier will be implemented and trained on the preprocessed data.
5. **Model Evaluation and Threshold Tuning:** Visualizations and performance metrics will be created and evaluated using the preprocessed training and test data, and hyperparameters will be tuned to maximize predictive performance.
6. **Visualization and Reporting:** Data visualizations will be exported and prepared for use in a client-facing dashboard.
7. **Packaging and Documentation:** The project will be packaged and prepared for deployment, and all required forms of documentation will be written and compiled.
8. **Final Review and Handoff:** The project will be reviewed by key stakeholders and prepared for integration into our company’s existing origination software.

Each step will be tracked and validated before proceeding to the next, ensuring quality and traceability throughout the development lifecycle.

Timeline

Milestone or deliverable	**Project Dependencies**	**Resources**	**Start and End Date**	**Duration**

Requirements and Setup	N/A	Developer, Kaggle dataset	May 1 – May 3, 2025	3 days
Data Cleaning & Preprocessing	Dataset Access	Python, pandas, and the imblearn library	May 4 – May 6, 2025	3 days
Model Development and Training	Preprocessing complete	Python, the scikit-learn library	May 7 – May 10, 2025	4 days
Evaluation and Threshold Tuning	A trained model	Python, sklearn metrics, matplotlib	May 11 – May 13, 2025	3 days
Visualization and Analysis	Evaluation results	Python, seaborn, custom plots	May 14 – May 15, 2025	2 days
Final Model Packaging	Trained pipeline complete	Python, the joblib library	May 16, 2025	1 day
Documentation and User Guide	Final application and visuals	Markdown, Word, screenshots	May 17 – May 18, 2025	2 days
Final Review and Submission	All deliverables ready	Reviewer, the completed project	May 19, 2025	1 day

Evaluation Plan

At each development stage, specific verification steps will be taken. First, within the preprocessing script, data distributions and null value counts will be exported and visualized to confirm proper handling. Next, the model training process will be verified through printed classification metrics, and comparisons will be performed across the different probability thresholds tested. The final validation will be conducted using a hold-out test set, ensuring the model generalizes beyond the training data. Key metrics such as ROC AUC, average precision score, and confusion matrix outputs will be used to confirm performance.

Costs

The costs for this project will be minimal, consisting primarily of labor costs. A simple breakdown of the costs is as follows:

- Software: Python ($0) – Primary programming language for model development.
- Software: VSCode ($0) – IDE for application development.
- Data Source: Lending Club Dataset ($0) – Train and test data for the machine learning model.
- Hardware: Local Development Machine ($0 – already owned) – Primary hardware for application development.
- Hosting: Render free tier ($0) – Frontend and backend hosting for the model and web application.
- Labor: Roughly 30 hours at $40/hour ($1,200)

The total estimated cost for this project is $1,200.

Part C: Application

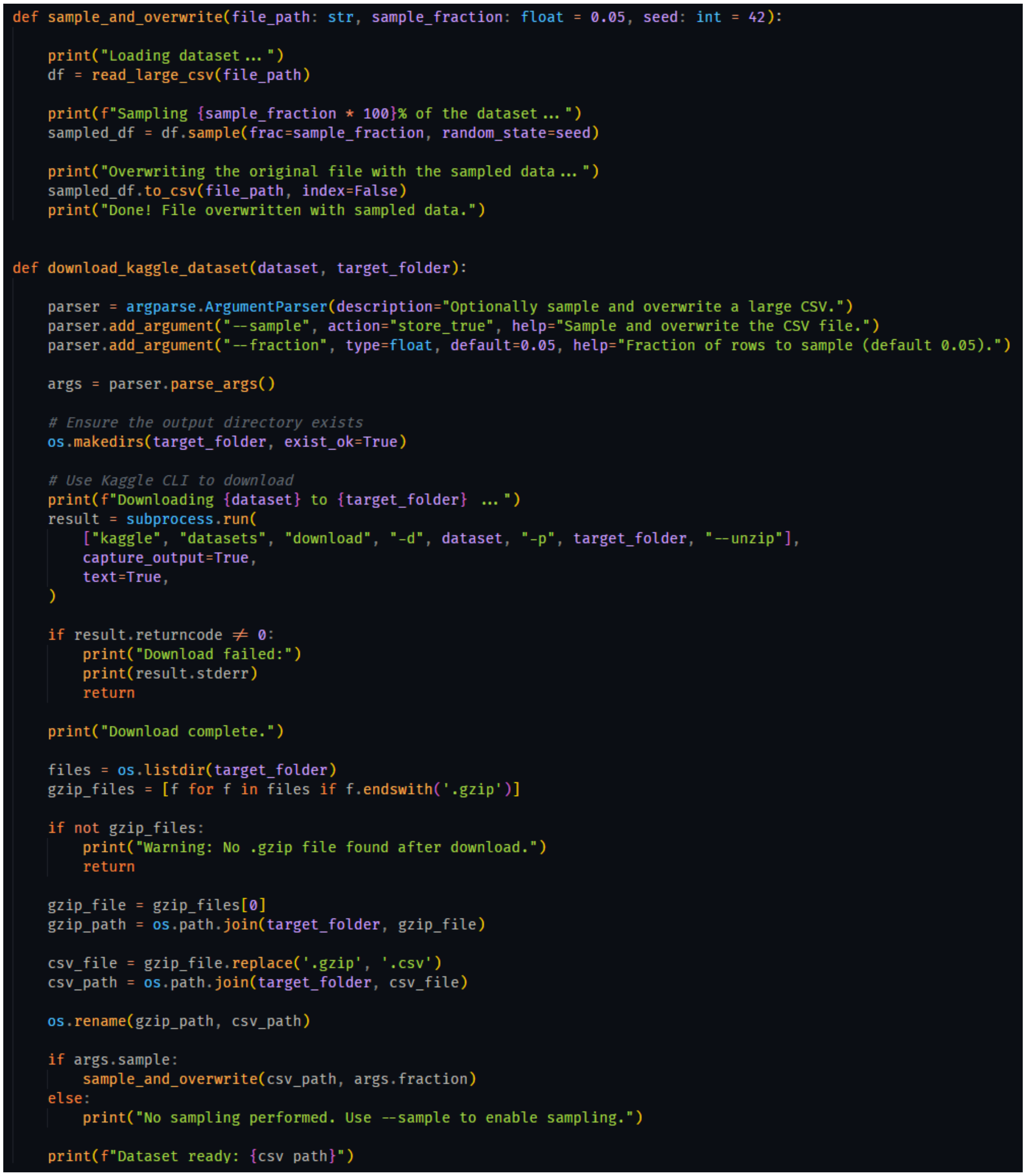
The project files for this application have been submitted along with this document. The following key folders contain source code with important functionality:

- “src/model” – This folder contains code used to preprocess data and train the machine learning model.
- “src/server” – This folder contains code used to create the Flask API and handle requests to endpoints for model prediction and data visualizations.
- “client” – This folder contains code used to create the frontend web application, including HTML, CSS, and JavaScript files.

Part D: Post-Implementation Report

This project addressed the problem of assessing credit risk by predicting the likelihood of loan default based on historical borrower data. Financial institutions need practical tools to evaluate loan applications and manage portfolio risk. Traditional credit scoring methods can be limited in flexibility and predictive power, especially when dealing with large, diverse datasets. This project implemented a supervised machine learning solution using a Random Forest classifier trained on real-world LendingClub loan data to improve accuracy and scalability.

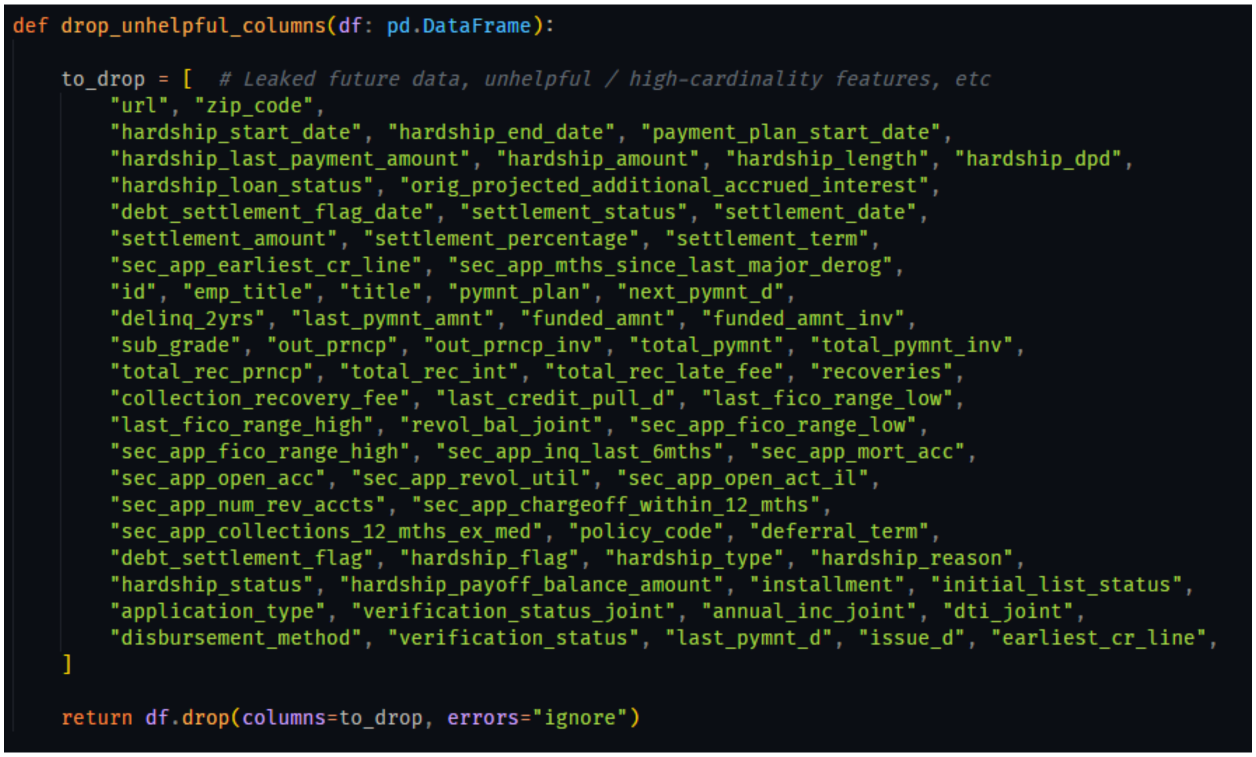
The raw historical loan data was downloaded from Kaggle using this utility script written in Python:



```python
def sample_and_overwrite(file_path: str, sample_fraction: float = 0.05, seed: int = 42):

    print("Loading dataset...")
    df = read_large_csv(file_path)

    print(f"Sampling {sample_fraction * 100}% of the dataset...")
    sampled_df = df.sample(frac=sample_fraction, random_state=seed)

    print("Overwriting the original file with the sampled data...")
    sampled_df.to_csv(file_path, index=False)
    print("Done! File overwritten with sampled data.")


def download_kaggle_dataset(dataset, target_folder):

    parser = argparse.ArgumentParser(description="Optionally sample and overwrite a large CSV.")
    parser.add_argument("--sample", action="store_true", help="Sample and overwrite the CSV file.")
    parser.add_argument("--fraction", type=float, default=0.05, help="Fraction of rows to sample (default 0.05).")

    args = parser.parse_args()

    # Ensure the output directory exists
    os.makedirs(target_folder, exist_ok=True)

    # Use Kaggle CLI to download
    print(f"Downloading {dataset} to {target_folder} ...")
    result = subprocess.run(
        ["kaggle", "datasets", "download", "-d", dataset, "-p", target_folder, "--unzip"],
        capture_output=True,
        text=True,
    )

    if result.returncode ≠ 0:
        print("Download failed:")
        print(result.stderr)
        return

    print("Download complete.")

    files = os.listdir(target_folder)
    gzip_files = [f for f in files if f.endswith('.gzip')]

    if not gzip_files:
        print("Warning: No .gzip file found after download.")
        return

    gzip_file = gzip_files[0]
    gzip_path = os.path.join(target_folder, gzip_file)

    csv_file = gzip_file.replace('.gzip', '.csv')
    csv_path = os.path.join(target_folder, csv_file)

    os.rename(gzip_path, csv_path)

    if args.sample:
        sample_and_overwrite(csv_path, args.fraction)
    else:
        print("No sampling performed. Use --sample to enable sampling.")

    print(f"Dataset ready: {csv_path}")
```

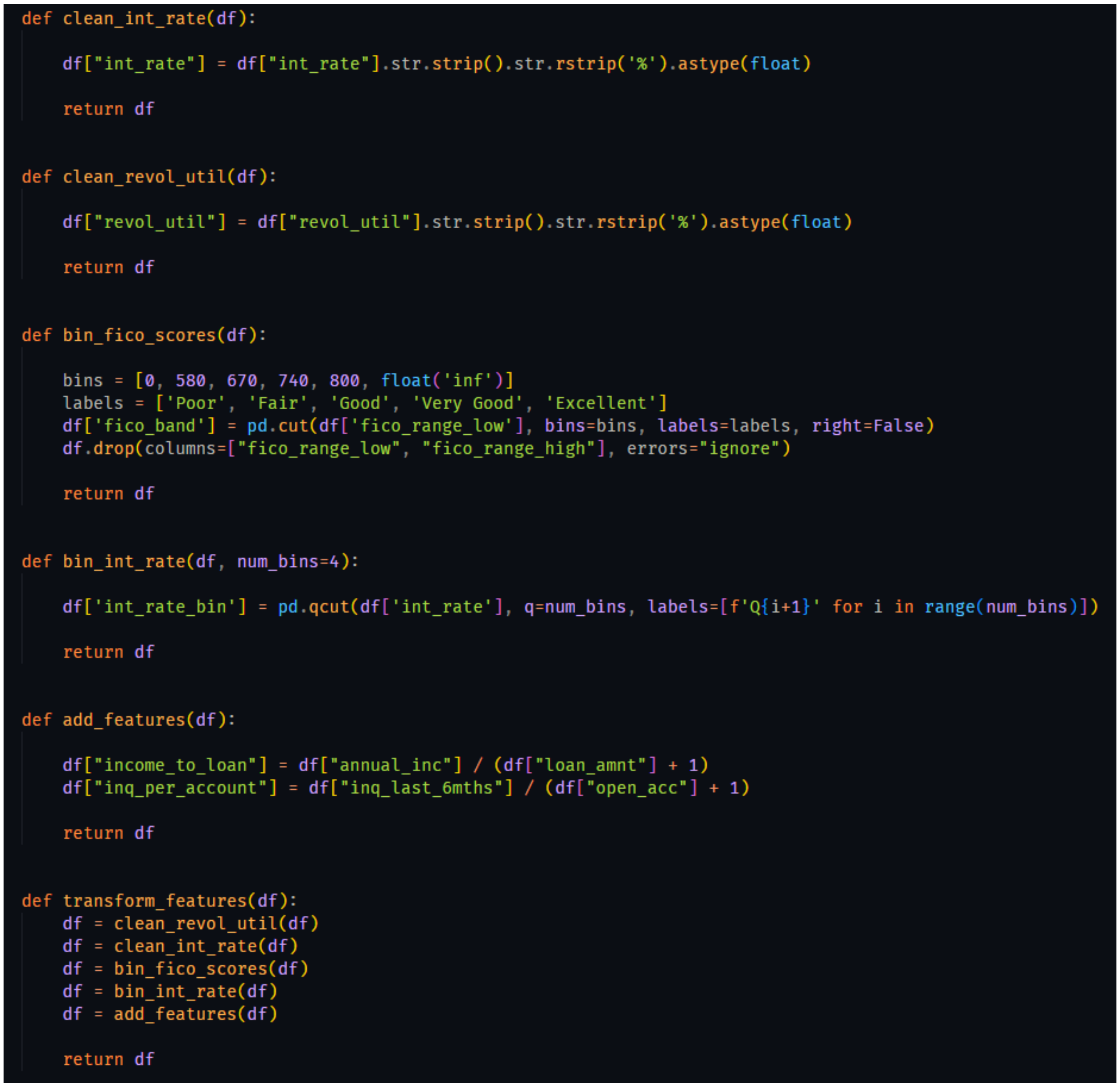
Initially, the whole dataset was used in model training. After finding complications in deploying the model due to its size, being trained on roughly 1.5 million loans, the entire dataset was instead sampled after being downloaded.

Several steps were identified and implemented to clean and preprocess the dataset throughout the application development lifecycle. First, several columns were dropped to avoid affecting the model through unnecessary noise and data leakage:



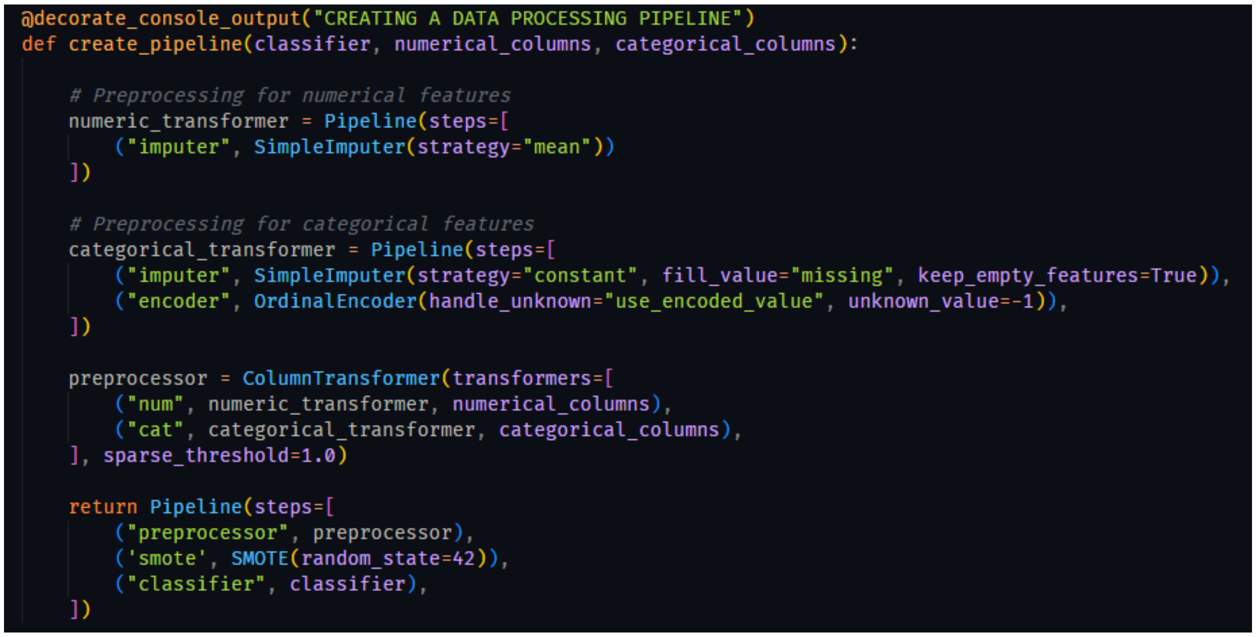
```python
def drop_unhelpful_columns(df: pd.DataFrame):

    to_drop = [  # Leaked future data, unhelpful / high-cardinality features, etc
        "url", "zip_code",
        "hardship_start_date", "hardship_end_date", "payment_plan_start_date",
        "hardship_last_payment_amount", "hardship_amount", "hardship_length", "hardship_dpd",
        "hardship_loan_status", "orig_projected_additional_accrued_interest",
        "debt_settlement_flag_date", "settlement_status", "settlement_date",
        "settlement_amount", "settlement_percentage", "settlement_term",
        "sec_app_earliest_cr_line", "sec_app_mths_since_last_major_derog",
        "id", "emp_title", "title", "pymnt_plan", "next_pymnt_d",
        "delinq_2yrs", "last_pymnt_amnt", "funded_amnt", "funded_amnt_inv",
        "sub_grade", "out_prncp", "out_prncp_inv", "total_pymnt", "total_pymnt_inv",
        "total_rec_prncp", "total_rec_int", "total_rec_late_fee", "recoveries",
        "collection_recovery_fee", "last_credit_pull_d", "last_fico_range_low",
        "last_fico_range_high", "revol_bal_joint", "sec_app_fico_range_low",
        "sec_app_fico_range_high", "sec_app_inq_last_6mths", "sec_app_mort_acc",
        "sec_app_open_acc", "sec_app_revol_util", "sec_app_open_act_il",
        "sec_app_num_rev_accts", "sec_app_chargeoff_within_12_mths",
        "sec_app_collections_12_mths_ex_med", "policy_code", "deferral_term",
        "debt_settlement_flag", "hardship_flag", "hardship_type", "hardship_reason",
        "hardship_status", "hardship_payoff_balance_amount", "installment", "initial_list_status",
        "application_type", "verification_status_joint", "annual_inc_joint", "dti_joint",
        "disbursement_method", "verification_status", "last_pymnt_d", "issue_d", "earliest_cr_line",
    ]

    return df.drop(columns=to_drop, errors="ignore")
```

Next, several data points were cleaned and formatted to make the model more easily understood. Two new features were also engineered and added to the dataset to boost its predictive performance:

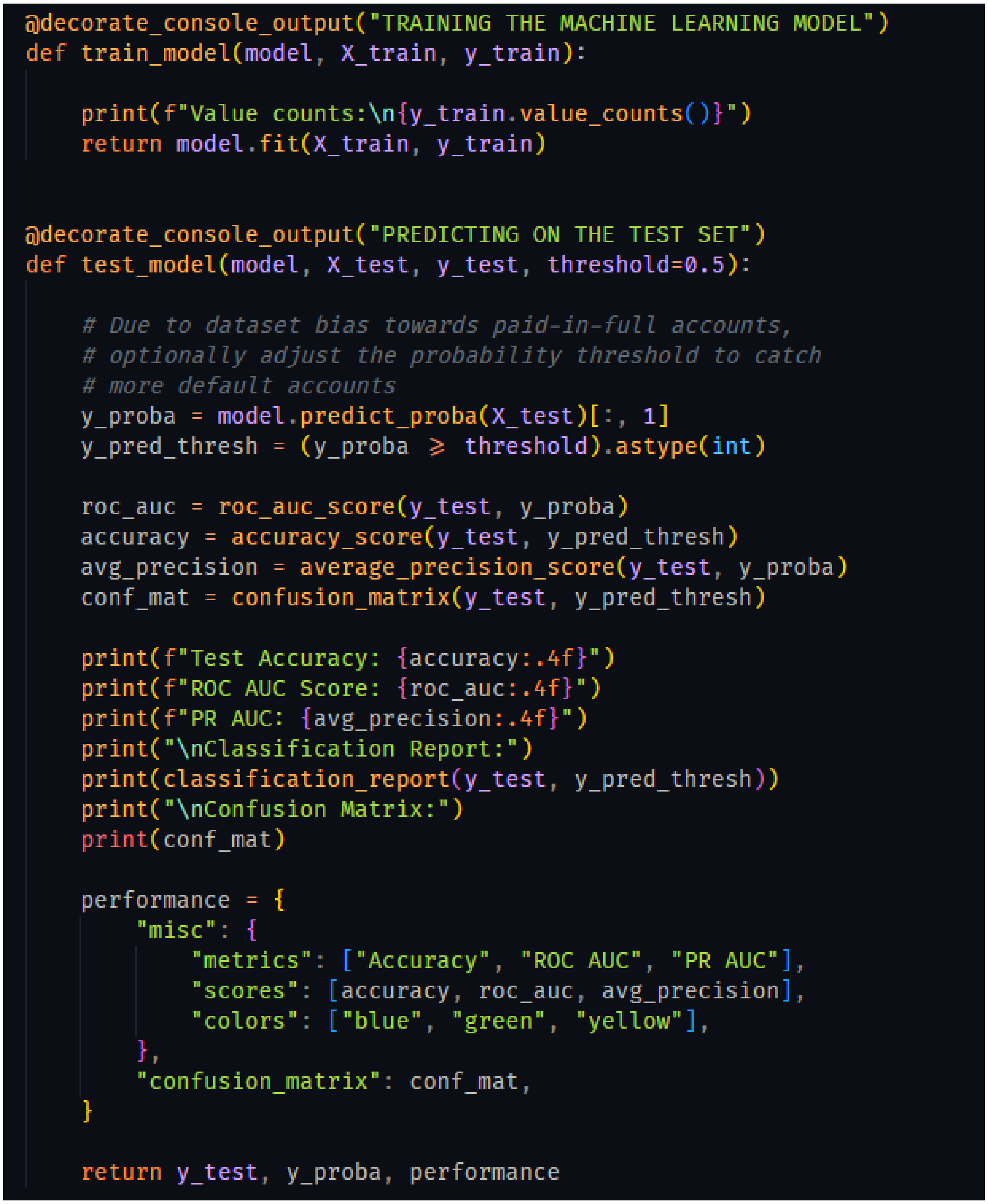


```python
def clean_int_rate(df):

    df["int_rate"] = df["int_rate"].str.strip().str.rstrip('%').astype(float)

    return df


def clean_revol_util(df):

    df["revol_util"] = df["revol_util"].str.strip().str.rstrip('%').astype(float)

    return df


def bin_fico_scores(df):

    bins = [0, 580, 670, 740, 800, float('inf')]
    labels = ['Poor', 'Fair', 'Good', 'Very Good', 'Excellent']
    df['fico_band'] = pd.cut(df['fico_range_low'], bins=bins, labels=labels, right=False)
    df.drop(columns=["fico_range_low", "fico_range_high"], errors="ignore")

    return df


def bin_int_rate(df, num_bins=4):

    df['int_rate_bin'] = pd.qcut(df['int_rate'], q=num_bins, labels=[f'Q{i+1}' for i in range(num_bins)])

    return df


def add_features(df):

    df["income_to_loan"] = df["annual_inc"] / (df["loan_amnt"] + 1)
    df["inq_per_account"] = df["inq_last_6mths"] / (df["open_acc"] + 1)

    return df


def transform_features(df):
    df = clean_revol_util(df)
    df = clean_int_rate(df)
    df = bin_fico_scores(df)
    df = bin_int_rate(df)
    df = add_features(df)

    return df
```

Next, the data was split into training and test data, using a standard 80/20 split. Categorical and numerical features were also identified, allowing missing fields to be correctly imputed based on the data type. Categorical data was also passed through an ordinal encoder to translate the data into a format the model can more easily reason about. SMOTE was also applied in the preprocessing pipeline to address the inherent class imbalance of the problem, where most loans will be paid in full and relatively few will default. This final stage of data preprocessing is defined in the following method:


```python
@decorate_console_output("CREATING A DATA PROCESSING PIPELINE")
def create_pipeline(classifier, numerical_columns, categorical_columns):

    # Preprocessing for numerical features
    numeric_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="mean"))
    ])

    # Preprocessing for categorical features
    categorical_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="constant", fill_value="missing", keep_empty_features=True)),
        ("encoder", OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value=-1)),
    ])

    preprocessor = ColumnTransformer(transformers=[
        ("num", numeric_transformer, numerical_columns),
        ("cat", categorical_transformer, categorical_columns),
    ], sparse_threshold=1.0)

    return Pipeline(steps=[
        ("preprocessor", preprocessor),
        ('smote', SMOTE(random_state=42)),
        ("classifier", classifier),
    ])
```

Upon completion of these preprocessing steps, the model was trained and tested on the data:

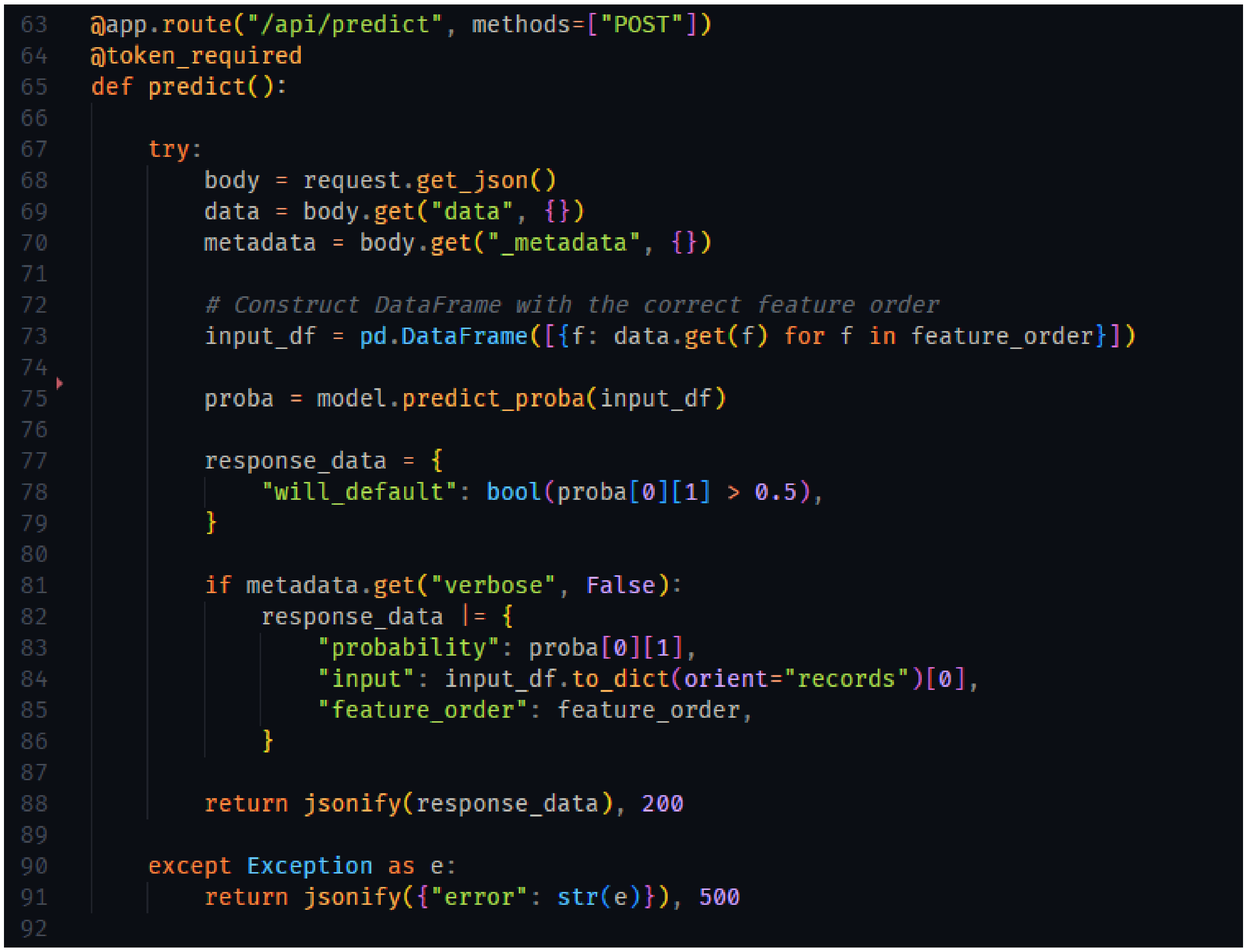


```python
@decorate_console_output("TRAINING THE MACHINE LEARNING MODEL")
def train_model(model, X_train, y_train):

    print(f"Value counts:\n{y_train.value_counts()}")
    return model.fit(X_train, y_train)


@decorate_console_output("PREDICTING ON THE TEST SET")
def test_model(model, X_test, y_test, threshold=0.5):

    # Due to dataset bias towards paid-in-full accounts,
    # optionally adjust the probability threshold to catch
    # more default accounts
    y_proba = model.predict_proba(X_test)[:, 1]
    y_pred_thresh = (y_proba >= threshold).astype(int)

    roc_auc = roc_auc_score(y_test, y_proba)
    accuracy = accuracy_score(y_test, y_pred_thresh)
    avg_precision = average_precision_score(y_test, y_proba)
    conf_mat = confusion_matrix(y_test, y_pred_thresh)

    print(f"Test Accuracy: {accuracy:.4f}")
    print(f"ROC AUC Score: {roc_auc:.4f}")
    print(f"PR AUC: {avg_precision:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred_thresh))
    print("\nConfusion Matrix:")
    print(conf_mat)

    performance = {
        "misc": {
            "metrics": ["Accuracy", "ROC AUC", "PR AUC"],
            "scores": [accuracy, roc_auc, avg_precision],
            "colors": ["blue", "green", "yellow"],
        },
        "confusion_matrix": conf_mat,
    }

    return y_test, y_proba, performance
```

The trained model outputs a probability of default for each applicant, with a decision threshold that can be configured to suit the client's risk tolerance best. This threshold-based prediction is beneficial for institutions wanting to customize approval strategies. Visualizations such as precision-recall curves, feature importance plots, and probability distributions are generated at the end of the model training process to support interpretability and operational decision-making. The application provides a modular, reproducible, and effective pipeline for improving loan default risk assessment using machine learning.

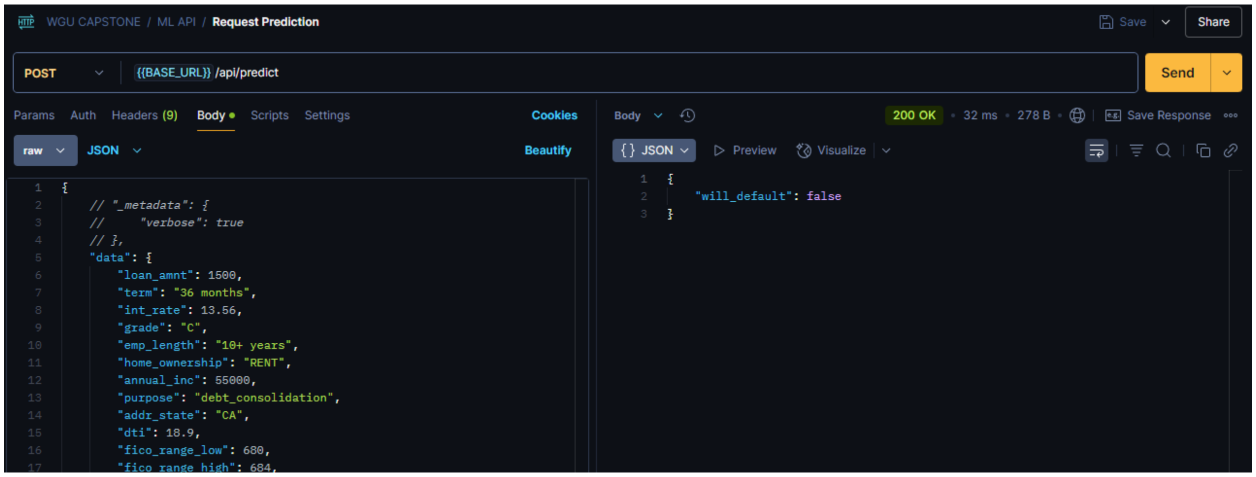
Machine Learning

The machine learning model's predictive functionality is accessed through the method "predict()" in the file "src/server/app.py":



```python
@app.route("/api/predict", methods=["POST"])
@token_required
def predict():

    try:
        body = request.get_json()
        data = body.get("data", {})
        metadata = body.get("_metadata", {})

        # Construct DataFrame with the correct feature order
        input_df = pd.DataFrame([{f: data.get(f) for f in feature_order}])

        proba = model.predict_proba(input_df)

        response_data = {
            "will_default": bool(proba[0][1] > 0.5),
        }

        if metadata.get("verbose", False):
            response_data |= {
                "probability": proba[0][1],
                "input": input_df.to_dict(orient="records")[0],
                "feature_order": feature_order,
            }

        return jsonify(response_data), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500

```

This method exposes an endpoint in the Flask API for model prediction. This allows loan application data to be sent in a JSON payload to the server, which queries the model and returns a prediction to the user. An example of the model's output is shown in the Postman request below:



```
POST {{BASE_URL}} /api/predict
```

```
{
    // "_metadata": {
    //     "verbose": true
    // },
    "data": {
        "loan_amnt": 1500,
        "term": "36 months",
        "int_rate": 13.56,
        "grade": "C",
        "emp_length": "10+ years",
        "home_ownership": "RENT",
        "annual_inc": 55000,
        "purpose": "debt_consolidation",
        "addr_state": "CA",
        "dti": 18.9,
        "fico_range_low": 680,
        "fico_range_high": 684,
```

```
{
    "will_default": false
}
```

This method achieves this functionality by using the list of features cached on the server to format the request data into a Pandas data frame. The data frame is submitted to the cached model through the model's built-in "predict_proba()" method, where the resulting probability is converted to a Boolean true/false prediction, where "true" means the borrower will likely default on their loan.

This method was selected and developed to make model predictions easily accessible to a frontend web application. Running computationally expensive methods, such as my machine learning model, on the server and making such functionality available to the frontend application through an API is a standard software engineering practice, so a prediction method that creates an API endpoint was a desirable way to expose machine learning functionality in my project.

Validation

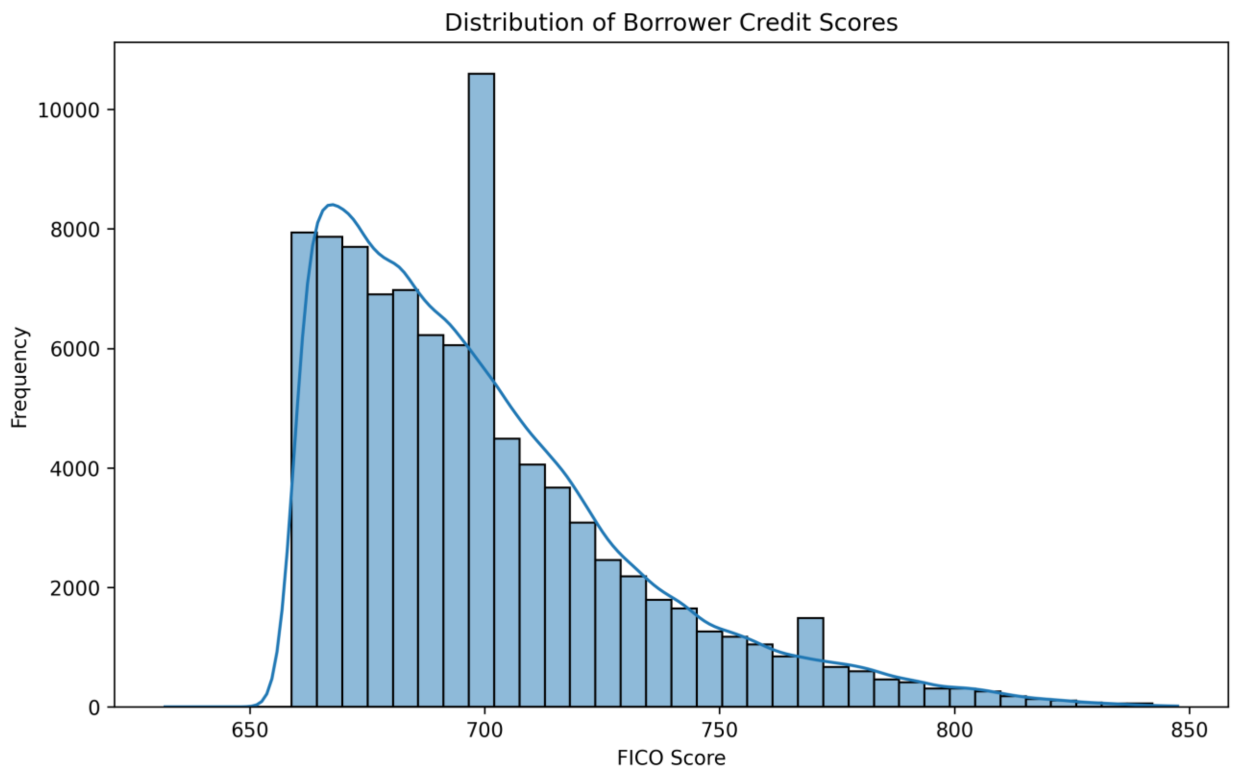
The machine learning model used in this project is a Random Forest classifier, which belongs to the supervised learning category. This model type is well-suited for binary classification tasks like predicting loan defaults because it can capture nonlinear patterns and handle imbalanced data effectively, especially when paired with techniques such as SMOTE (Synthetic Minority Oversampling Technique), which was incorporated into the pipeline to address class imbalance. The model was trained and evaluated using a train-test split, with separate training and test datasets derived from the original LendingClub loan data. This approach provided a precise, unbiased evaluation of the model's generalization performance on unseen data.

Several evaluation metrics were used to measure model performance. The primary metric was the Area Under the Receiver Operating Characteristic Curve, or AUC-ROC. This metric is appropriate for imbalanced classification problems because it measures the model's ability to distinguish between classes across different thresholds. Additional metrics, including accuracy, precision, recall, and F1-score, were also calculated to provide a well-rounded performance assessment. The final model achieved an AUC-ROC score of 0.7038, indicating strong discriminative ability. The accuracy was 69%, with a precision of 0.77, a recall of 0.69, and an F1-score of 0.72. These results demonstrate the model's effectiveness in predicting loan defaults while managing the trade-off between false positives and false negatives, a critical consideration in financial risk modeling.

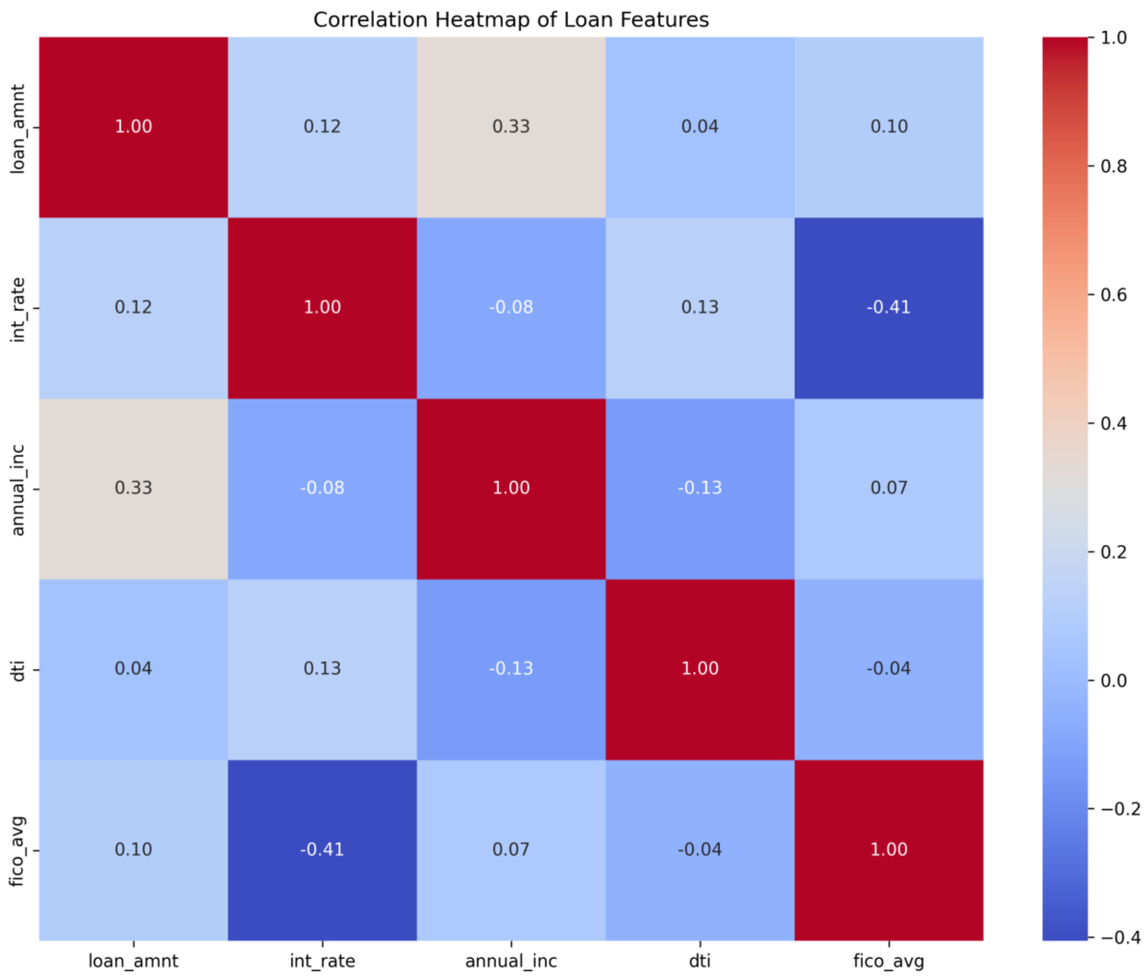
Visualizations

Several key visualizations were created to understand better the model's performance and the data it was trained on. These visualizations can be accessed in a few ways. First, they are available in this project's "data/visualizations" subdirectory. Second, they can be viewed on the web application dashboard; see the user guide below for information on accessing this project live or in a local environment. Finally, links to the visualizations in a public S3 bucket can be retrieved by sending a request to the "GET /api/visualizations" endpoint in the application's Flask API. These visualizations are also provided below:

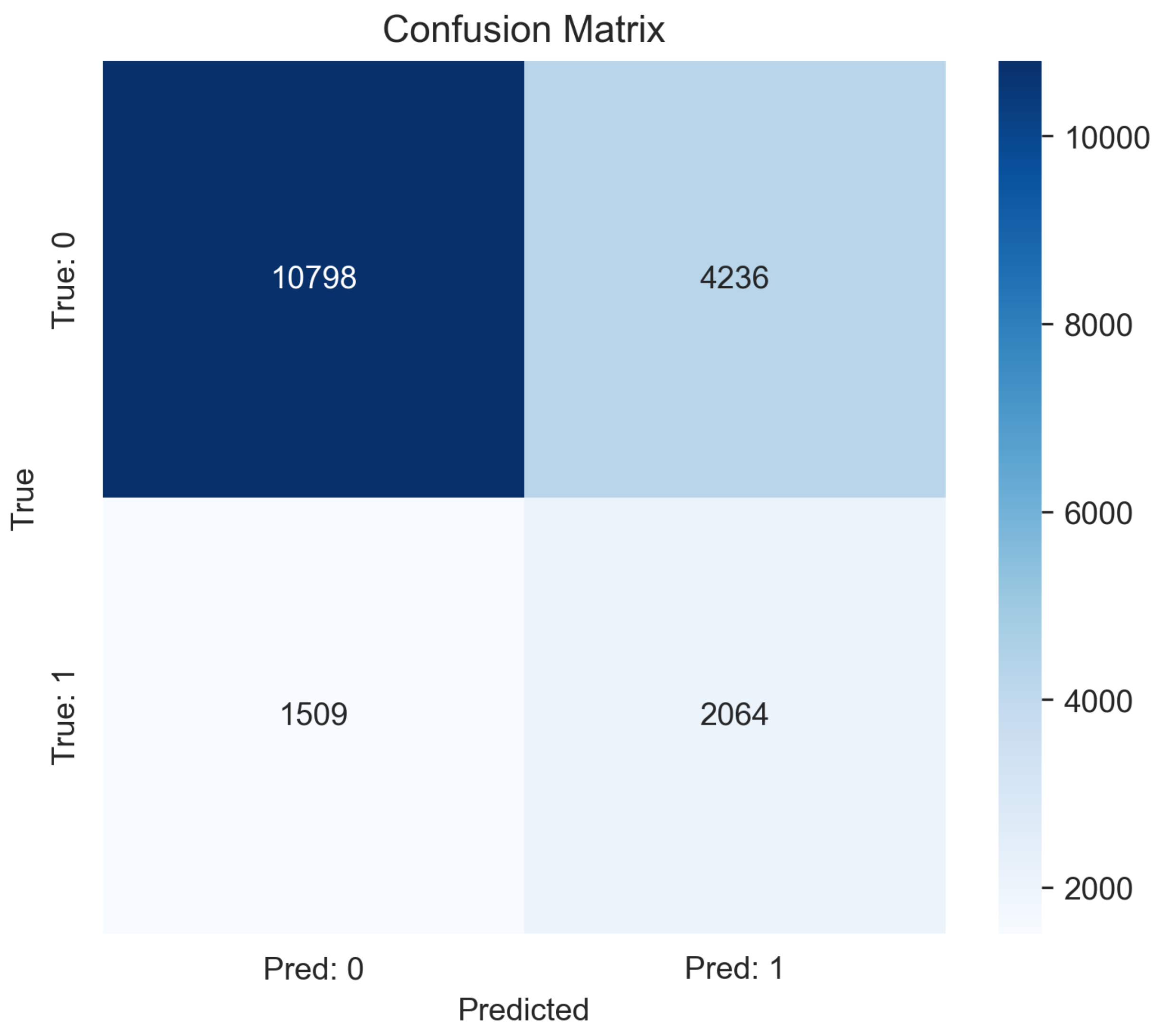
FICO Distribution in Training Dataset:



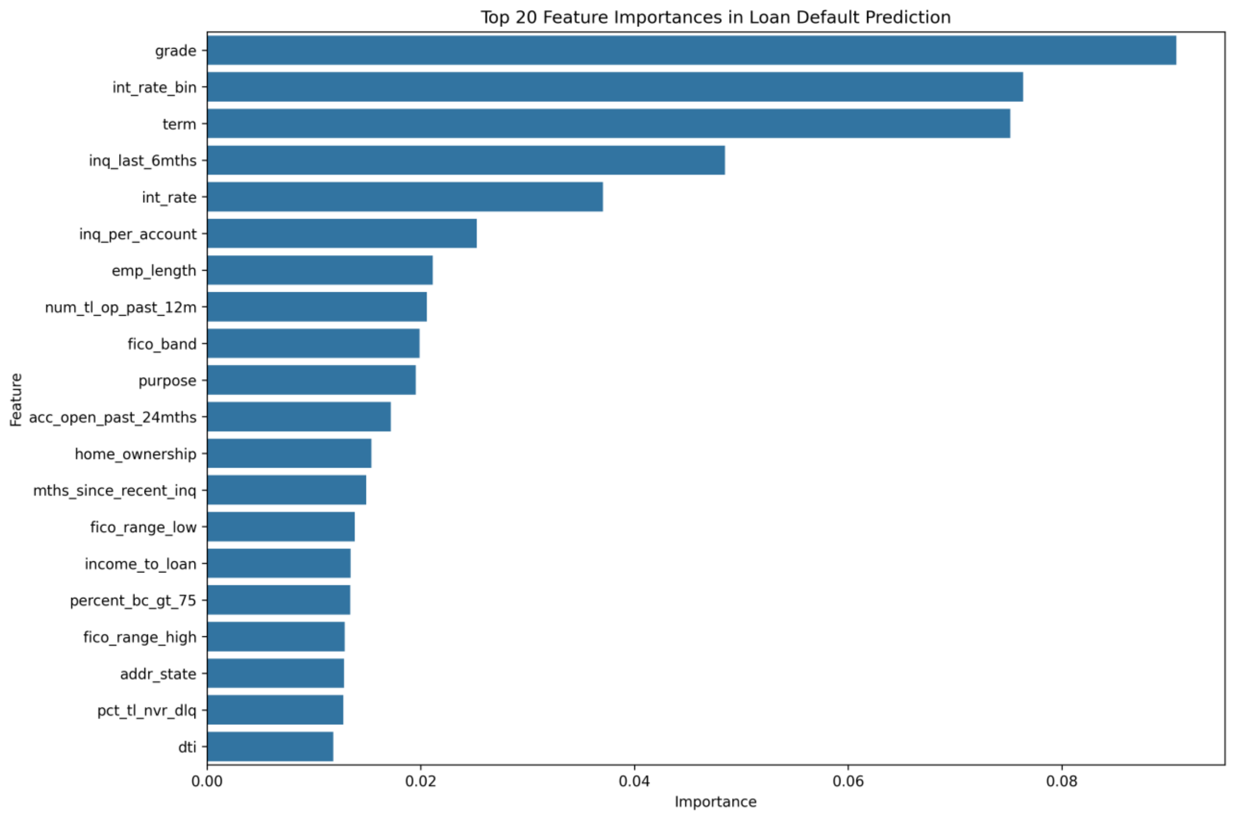
Common Feature Correlation Heatmap:



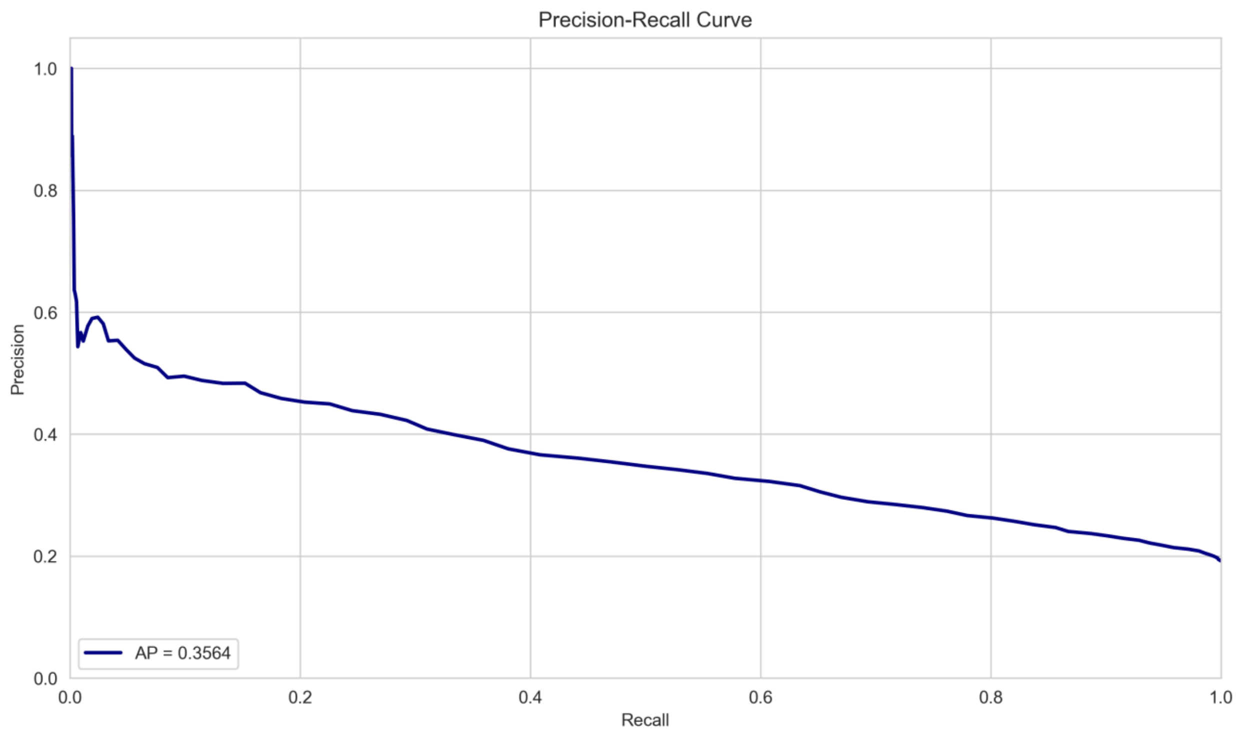
Confusion Matrix:



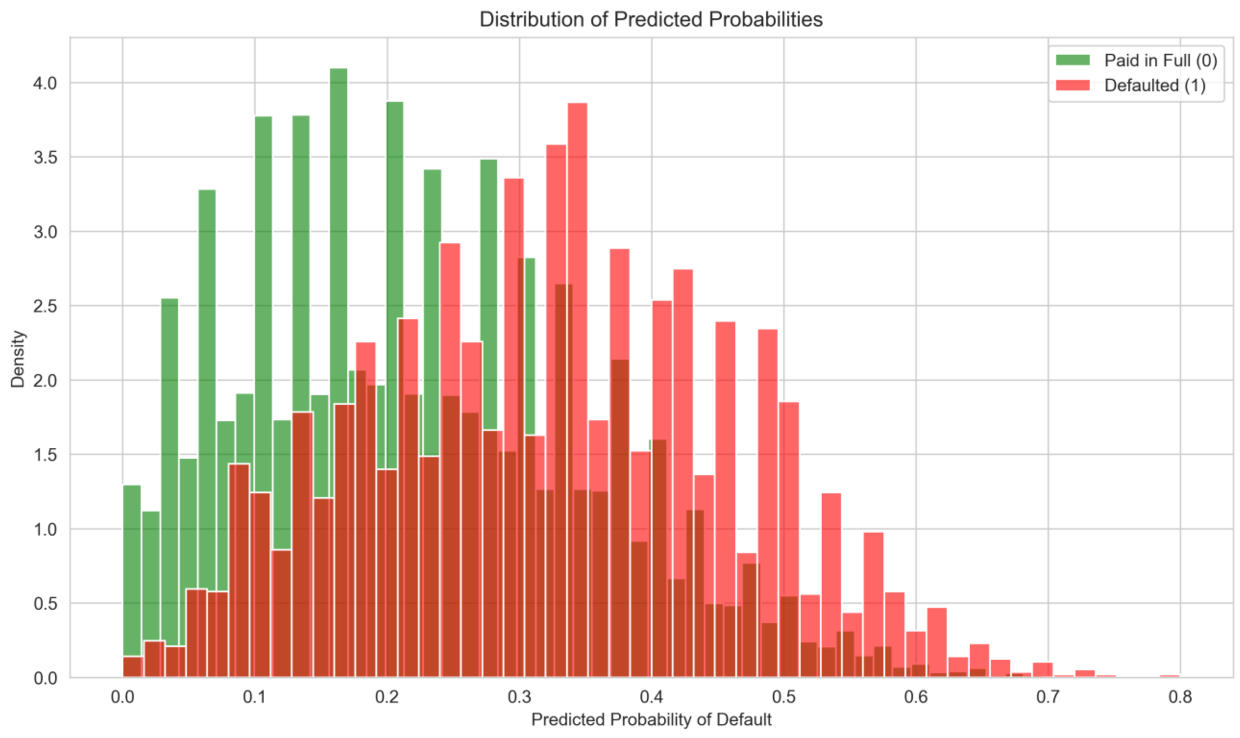
Feature Importance:



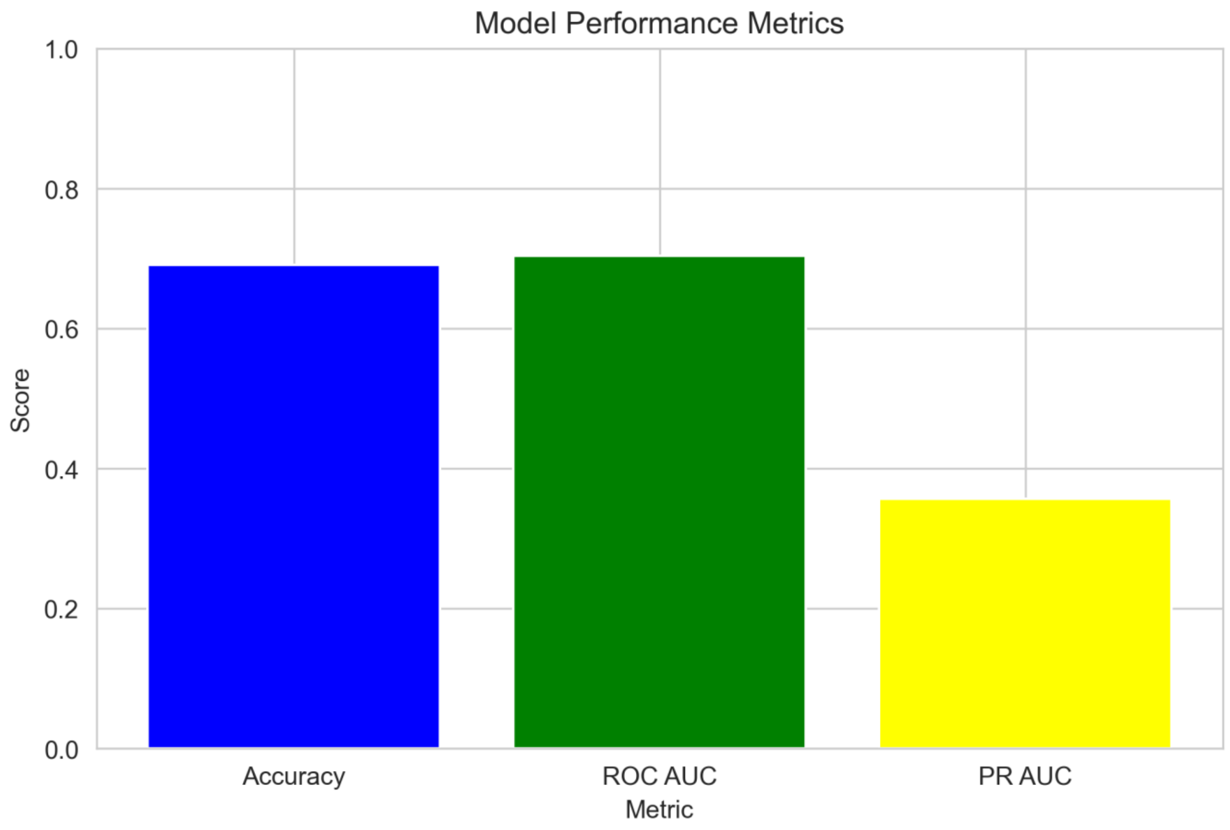
Precision-Recall Curve:



Probability Distribution:



General Performance Metrics:



User Guide

The application can be used either in a live environment or hosted locally. To use the live application, navigate to the following page in your web browser of choice:

https://wgu-capstone-client.onrender.com/

If you wish to run the application locally, first ensure you have a recent version of Python (3.11+) and Node.js (19+) installed on your computer. Python can be downloaded from https://www.python.org/downloads/, and Node.js from https://nodejs.org/en/download/. Next, perform the following steps:

1. Open a command prompt (PowerShell, Git Bash, etc.) and navigate to the project directory. For example, run the command "cd path/to/project".
2. Initialize a Python virtual environment with the command "python -m venv .venv".
3. Activate your new virtual environment; for example, run "./.venv/Scripts/activate". The actual command may vary depending on your environment. For more details, visit https://docs.python.org/3/library/venv.html.
4. Run the following command to install dependencies, download a sample of the full LendingClub dataset, train a Random Forest classifier, and run the full-stack application locally: "npm run for:evaluator". This process may take several minutes. When the development environment is ready, it should automatically open the web page in your preferred browser.

Once you have the web application (hosted locally or live) open in your web browser, perform the following steps to simulate a client using the application:

1. At the top right of the web page, click the link "Loan Default Prediction" to navigate to the next page.
2. Review the possible inputs for submitting a loan application for prediction. For convenience, defaults are populated for most fields.
3. Click the "Submit Application" button at the top of the form to submit the provided data to the ML model for prediction. After a moment, you should see the model's prediction listed above the application form.
4. Change a few input fields to simulate a risky borrower; for example, set "Loan Amount" to "40000", "Grade" to "D", and "Employment Length" to "< 1 year" and submit the application. Note how the model predicts the loan will default. Now, lower the interest rate to "7.99" and submit the application again; the model should now expect the loan will not default.

Reference Page

No outside sources were used in this project.