Sacred Heart UNIVERSITY

AI-Powered Credit Risk Assessment and Approval System

Final Project Proposal

Presented by
Venkata Sivani Dasari
Dasariv9@mail.sacredheart.edu

Introduction

Credit risk assessment is a pivotal task in financial services, involving the prediction of whether a loan applicant will default on credit obligations. Traditional credit scoring relied on statistical models (e.g. logistic regression), but modern machine learning techniques have greatly enhanced predictive accuracy and decision speed. With machine learning (ML), credit risk models can analyze complex patterns in borrower data, leverage larger datasets, and enable more precise, real-time risk evaluation. This project proposes an end-to-end **AI-powered credit risk assessment system** that integrates robust ML models, interactive visual analytics, and generative AI. The system will predict credit approval and default risk, provide interpretability for decisions, and offer user-friendly tools for both analysts and decision-makers. By combining a predictive model with an interactive Streamlit web application, Tableau dashboards for exploratory insight, and an AI-generated executive summary, the project aims to demonstrate a comprehensive solution for intelligent credit screening. Ultimately, this project will illustrate how advanced analytics and AI can streamline credit approval decisions, reduce default rates, and enhance transparency in lending practices.

Objectives

- **Accurate Credit Approval Prediction:** Develop classification models (e.g. Random Forest, XGBoost) to predict whether a credit application should be approved, and the likelihood of default, with high accuracy. These ensemble methods are among the most popular and effective for credit scoring.
- **Risk Evaluation & Interpretability:** Evaluate the default risk of applicants and provide human-interpretable explanations for each prediction. We will use SHAP (Shapley Additive Explanations) to identify which features most influenced a given approval decision or default prediction. This addresses regulatory and trust requirements by making the "black-box" model more transparent.
- **Interactive Decision App:** Build a user-friendly **Streamlit** web application where users (e.g. loan officers) can input applicant details and immediately receive a model-driven approval decision. The app will display the predicted outcome (approve/deny) along with an explanation (key factors contributing to the decision). Streamlit, an open-source Python framework, allows rapid development of such interactive data apps with minimal coding.
- **Visualize Portfolio Insights:** Utilize **Tableau** to create interactive dashboards that show credit risk trends across the portfolio – for example, default rates segmented by income level, geographic region, loan amount, etc. This will enable stakeholders to spot patterns (like higher default risk in certain income brackets or locations) and make data-driven strategic decisions. Tableau's ability to drill down from aggregate portfolio views to specific segments will facilitate quick identification of risk concentrations.
- **AI-Generated Executive Summary:** Integrate a language model (GPT-based) to automatically generate an executive summary of the credit portfolio's health. After each model run or on-demand, the system will produce a concise narrative report highlighting key metrics (overall approval rate, default rate, high-risk segments, trends). This aligns with emerging uses of generative AI in credit risk management – for instance, automating routine risk reports and portfolio summaries for management.

By achieving these objectives, the project will deliver a functional prototype that mimics real-world credit underwriting tools, demonstrating competencies in data science, machine learning, business intelligence, and AI-driven automation.

Dataset

The primary dataset for this project is the **UCI Statlog Australian Credit Approval dataset**, a real-world credit card application dataset widely used as a benchmark in credit scoring research. This dataset contains 690 instances of credit applications, each with 14 attributes describing the applicant's personal and financial information, plus a class label indicating approval status. The features are a mix of numeric and categorical variables (6 numerical and 8 categorical attributes) and have been encoded with anonymous symbols to protect confidentiality. Notably, the dataset includes a variety of attribute types (continuous features like income or loan amount, and nominal features such as bank account status or employment status) and contains a few missing values. This diversity makes it well-suited for evaluating different preprocessing and modeling techniques.

We will perform an 80/20 train-test split (or use cross-validation) on this dataset to train and evaluate our models. If needed, additional public datasets or synthetic data may be incorporated to enrich the training data or test the model's robustness. For example, other credit scoring datasets (like the German Credit Data or Kaggle loan default datasets) could be used for comparative analysis, although the focus will remain on the Australian Credit dataset as the primary source. All data usage will comply with the dataset's license (CC BY 4.0), and the UCI repository link will be provided in the project documentation for reproducibility.

Methodology

Our methodology follows the standard CRISP-DM framework (Cross-Industry Standard Process for Data Mining) – encompassing data understanding, preparation, modeling, evaluation, and deployment – tailored to the project's components:

- **1. Data Preprocessing:** We will load the UCI credit approval data into pandas and first perform exploratory data analysis (EDA). During EDA, we'll summarize key statistics and visualize distributions of important features (e.g. histogram of income, counts of categorical variables) to understand data characteristics. Data cleaning steps will include handling missing values (e.g. using mean/mode imputation or simple strategies given the small proportion of missing entries) and encoding categorical variables. Categorical features will be encoded via one-hot encoding or ordinal encoding as appropriate (the dataset's documentation provides a mapping of original categorical symbols to numeric codes, which we will leverage). We will also normalize or scale continuous features if needed (for instance, if using models sensitive to feature scale). To address any class imbalance (if the dataset has disproportionate approved vs denied cases), we may apply techniques like SMOTE (Synthetic Minority Oversampling Technique) or class weight adjustments, as recommended by recent credit risk studies.
- **2. Feature Engineering (if applicable):** Given the anonymized nature of the features, domain-specific feature creation is limited. However, we will consider deriving any

meaningful composites (for example, if there are separate features for income and debt, one could create a debt-to-income ratio). We will also examine feature importance from simple models to decide if any feature can be binned or transformed to improve model performance (e.g., binning a continuous variable into categories if non-linear relationships are observed).

- **3. Modeling – Classification Algorithms:** We will train and compare multiple classification models to predict credit approval (or default risk). In particular, we plan to implement:
    - **Random Forest:** an ensemble tree-based model known for handling mixed data types and capturing non-linear relationships. Random Forests have been reported as one of the most effective classifiers in credit risk modeling, frequently used due to their high accuracy and robustness.
    - **XGBoost (Extreme Gradient Boosting):** a powerful gradient boosting algorithm that often achieves state-of-the-art results in structured data problems. XGBoost has been successfully applied in credit scoring competitions and research, and its ability to handle feature interactions and imbalanced data (with proper parameter tuning) makes it a strong candidate.
    - (We may also test a baseline **Logistic Regression** model for reference, as it is traditionally used in credit scoring, and possibly a **Neural Network** or other classifiers if time permits, to see if more complex models add value.)

  Model training will involve splitting the data into training and validation sets (or using k-fold cross-validation) to tune hyperparameters. We will perform hyperparameter optimization for each model (for example, using Grid Search or Randomized Search CV in scikit-learn) to find the best parameters (e.g., number of trees, max depth for Random Forest; learning rate, max depth, and tree count for XGBoost). For efficiency, we might use 5-fold cross-validation on the training set to evaluate parameter combinations. Our performance metric of choice will be **ROC-AUC** (Area Under the ROC Curve) and **F1-score**, since these consider both classes and are suitable for imbalanced classification evaluation. We will also monitor accuracy and confusion matrix, but ROC-AUC and F1 provide a better sense of model discrimination and balance between false positives/negatives in credit decisions. Recent research emphasizes evaluating multiple metrics (AUC, precision, recall, F1) for credit risk models, so we will report a suite of metrics to give a comprehensive view of model performance.

- **4. Model Evaluation:** After training, we will evaluate models on the hold-out test set (20% of data) to estimate how well they generalize to unseen applications. Key evaluation metrics will include Accuracy, Precision, Recall, F1-score, and ROC-AUC. Precision and recall are particularly important: a high recall (low false negatives) means fewer risky applicants are mistakenly approved, while high precision (low false positives) means we minimize denying credit to creditworthy applicants. We will select the "best" model by considering the business context – for instance, if false negatives (approving a bad risk) are more costly, we might favor the model with higher recall even at some cost to precision. The results will be presented in tables and possibly an ROC curve plot for visual comparison of model trade-offs.

- **5. Interpretability – SHAP Analysis:** To ensure the model's decisions are explainable, we will apply SHAP (Shapley Additive explanations) on the final model. SHAP assigns each feature a contribution value for a given prediction, indicating how that feature moved the prediction above or below the average prediction. We will compute SHAP values for sample inputs and the test set to understand global feature importance as well as local explanations for individual applicants. For example, SHAP might reveal that *Feature A* (perhaps representing income) has a strong positive impact on approving credit, while *Feature B* (perhaps representing number of past defaults) strongly contributes to rejections. SHAP is particularly well-suited here as it provides consistent, game-theoretic attributions and works effectively with tree-based models like Random Forests and XGBoost. We will include plots such as SHAP summary plots (showing feature impact distribution) and force plots for individual predictions in our analysis. These interpretable insights will be crucial for the Streamlit app and for the final report, allowing us to justify why the model made each decision – an essential aspect in credit lending for compliance and user trust.
- **6. Streamlit Web App Development:** With a trained model and the ability to interpret its outputs, we will develop a **Streamlit** application to make the solution interactive. Streamlit allows us to turn a Python script into a web interface easily. In the app, users will find input fields (sliders, dropdowns, text inputs as appropriate for each feature) to enter a new applicant's details. Upon submission, the app will run the input through the ML model (which will be loaded, likely via a saved model file using `joblib` or similar) and display the predicted approval decision (approve/deny) along with an explanation. The explanation could be a small list of the top 3 features that most influenced the decision, drawn from the SHAP values for that prediction – e.g., *"Income level is high (contributing +0.15 to approval score), but recent delinquencies contribute -0.20, resulting in an overall denial."* This gives the user immediate insight into the "why" behind the model's judgment. The app will also include confidence metrics (like predicted probability of default) and could allow the user to simulate changes (what-if analysis, e.g., see if increasing income or providing collateral might change the decision). We will ensure the UI is clean and intuitive: e.g., a sidebar for inputs and a main area showing the output decision, explanation, and perhaps a small gauge or graphic indicating risk level. This component demonstrates deployment of the model in a real-time decision tool, as might be used by loan officers.
- **7. Tableau Dashboard Development:** In parallel to the individual prediction app, we will use **Tableau** to create dashboards for aggregate analysis of the credit portfolio (using the dataset or simulation of a portfolio). We will load the data (with model predictions if needed) into Tableau and design several views:
    - **Portfolio Overview:** A summary dashboard showing overall approval rate, default rate (if historical outcomes are available or as predicted by model), and distribution of applicants by risk grade. This might include a pie or bar chart of approved vs denied applications, and a KPI displaying the average predicted default probability.
    - **Segmented Risk Trends:** Visualizations to explore how risk varies by different factors. For example, a bar chart or line chart of default rate by income bracket, a map or stacked bar by region, or a breakdown by employment type or other categorical features available. Users (e.g., risk managers) will be able to filter or

drill down – for instance, select a region to see the credit approval rate in that region, or filter to a certain income range to inspect its default trend. Tableau's interactive features (filters, tooltips, highlight actions) will make it easy to slice the data. This addresses questions like *"Which customer segments are the riskiest?"* or *"How does default risk compare between high-income and low-income applicants?"*.

- **Time Trends (if applicable):** If the dataset had a time component (e.g., application date or if we simulate portfolio performance over time), we would plot trends of default rates or number of applications over time. (If not available, this section may be omitted or a static trend assumed for demonstration.)

The Tableau dashboards will be designed for clarity and quick insight – for example, using color coding to highlight high-risk areas. A real-world inspiration is how financial institutions use dashboards to monitor risk exposure in real-time. In our case, while the data is static, the dashboards will demonstrate how one could monitor a live portfolio. We plan to include at least 2–3 distinct dashboard pages (overview and a couple of detailed views). These dashboards can be presented via Tableau Public (if permissible to upload the data) or as packaged workbook files for the professor to interact with.

- **8. AI-Generated Executive Summary:** As a final layer, we will implement a component to generate a written summary of the credit portfolio's status using a **GPT-based language model**. The process will involve programmatically assembling key findings (e.g., *"out of X applications, Y% were approved; the overall predicted default rate is Z%. High-risk segments identified include [segment A] with default rate M%, etc."*). This data will be fed into a prompt for a language model (such as OpenAI's GPT-4 via API, or a local large language model if API usage is not possible) to create a well-structured executive summary. Automating this with AI demonstrates how generative models can assist in translating data into natural language conclusions. This approach is in line with emerging industry practices, where generative AI is used to draft credit memos, risk reports, and portfolio analyses to save analysts' time. We will ensure that the generated summary is reviewed for correctness (the model will only be as good as the prompt and data provided). This summary could be displayed in the Streamlit app (e.g., a button to "Generate Portfolio Summary") or included in the final report deliverables as a sample output.

Tools & Technologies

This project will utilize a range of software tools and libraries, each chosen for its suitability to the task at hand:

- **Python (Programming Language):** All core modeling and data processing will be done in Python, given its rich ecosystem for data science. Key libraries include:
  - **pandas:** for data manipulation, cleaning, and exploratory analysis.
  - **scikit-learn:** for implementing machine learning models (logistic regression, RandomForestClassifier, etc.), preprocessing (imputation, encoding, scaling), and

evaluation metrics. Scikit-learn's consistent API will allow us to build pipelines that handle preprocessing and prediction seamlessly.
    - **XGBoost library:** for training the XGBoost model. We will use either the `xgboost` Python package or scikit-learn's wrapper (`XGBClassifier`), enabling advanced boosting functionality for better accuracy.
    - **SHAP:** an interpretability library to calculate SHAP values for model explanations. SHAP has become a standard for explainable AI in finance because it provides clear insights into feature effects. We will use it to generate plots and values that explain model predictions.
    - **joblib or pickle:** for saving trained model objects to disk so that they can be loaded in the Streamlit app without retraining. This ensures the web app can quickly load the pre-trained model and make predictions in real-time.
- **Streamlit:** a Python web application framework specifically designed for data science applications. Streamlit allows us to create an interactive web UI with just Python code (no need for separate HTML/JS/CSS development). We'll use Streamlit to design the credit approval prediction tool. It provides widgets for user input (sliders, dropdowns, text inputs) and simple commands to display outputs (tables, charts, text). This makes it ideal for rapidly prototyping our decision app and sharing it with users (in this case, our professor or classmates, who can run it locally or via Streamlit sharing). Streamlit's simplicity ("just run `streamlit run app.py` to launch") is a major advantage for demonstration purposes.
- **Tableau:** a leading business intelligence and data visualization platform. We choose Tableau for creating the risk dashboards because of its powerful drag-and-drop interface and its ability to handle quick segmentation and filtering, which is useful for non-technical stakeholders. Tableau will be used outside of Python – we will export the processed data (possibly including model outputs or risk scores) as CSV and load into Tableau to craft interactive visuals. The choice of Tableau (over, say, matplotlib or Plotly in Python) is to simulate a real corporate setting where data scientists present results in an easy-to-use dashboard for management. Tableau also provides options to publish dashboards (Tableau Public or Server) for sharing.
- **OpenAI GPT-4 (or similar LLM):** for the generative AI summary component. If internet access/API access is available, we will use OpenAI's GPT via their API to generate the executive summary. This involves using the `openai` Python package, sending a prompt with our data insights, and receiving the model's generated text. In case API access is not feasible (due to costs or restrictions), we may use an open-source large language model (like GPT-3.5 Turbo through Azure if provided by the university, or a local model like Llama 2) to demonstrate the concept. In the proposal context, we assume access to a GPT model. This tool represents the AI "brain" for turning numbers into narrative.
- **Miscellaneous:** Other tools include:
    - **Matplotlib/Seaborn:** for any needed static plots during EDA or to include in the report (e.g., distribution plots, correlation heatmaps).
    - **NumPy:** for numerical computations and array handling, often used under the hood by pandas and scikit-learn.
    - **Git and GitHub:** for version control and collaboration (ensuring we keep track of changes to code, and can revert or review history as needed).

- **IDE or Notebook:** Development will be done in Jupyter notebooks for initial analysis (because of the ease of iterative exploration) and in a code editor (VS Code/PyCharm) for building the final streamlined scripts (like the Streamlit app).

All these tools together form a modern data science tech stack that will enable us to implement the project end-to-end, from data processing and model building to deployment and reporting. We will also ensure compatibility between these tools (for instance, making sure the Python environment has all required library versions, and that the Tableau version can easily import our data). Any custom code (e.g., for data processing or the GPT prompt) will be documented and included in the project deliverables.

Deliverables

By the end of the project, we will produce a comprehensive set of deliverables demonstrating both the functioning system and the analysis behind it:

- **1. Streamlit Web Application:** A fully functional web app (in the form of a Python script, e.g. `app.py`) that can be run to interactively input applicant data and output credit approval decisions. This app will include the trained model and produce both the decision and an explanation for the decision. We will provide the source code and a short user guide (README) on how to run the app. Optionally, a live demo could be deployed on Streamlit's sharing platform for ease of access (if permitted).
- **2. Machine Learning Model & Pipeline:** The final ML model object (saved via joblib/pickle) along with the pipeline code used for preprocessing and prediction. We will also include documentation of model performance (evaluation metrics on test data) and any validation results. This may be accompanied by a Jupyter notebook or PDF report containing the analysis, charts (like ROC curve, feature importance), and commentary on model selection. Essentially, this is the data science report that validates the model's effectiveness.
- **3. Tableau Dashboards:** A packaged Tableau workbook (`.twbx`) containing the interactive dashboards developed. This will illustrate risk trends by various dimensions (income, region, etc.) as discussed. If Tableau Public is used, a link to the public dashboard will be provided. Screenshots of key dashboards will also be included in the report appendix for reference.
- **4. Executive Summary Report (AI-generated):** A sample executive summary of the credit portfolio, generated by the AI component. This can be delivered as a PDF or document that showcases the output from the GPT model. We will highlight that the text was AI-generated based on our data findings. Along with the summary itself, we will provide the methodology (prompt design) used to obtain it.
- **5. Documentation & Proposal Write-up:** A written report (approximately 10-15 pages) that includes the **Introduction, Objectives, Methodology, Results, and Conclusions**, akin to an academic report or thesis-style write-up. This proposal document (the one we are writing now) can serve as the basis for the final report, which will be updated with actual results and findings. We will ensure to include citations for any references used (research papers, documentation) in APA style and acknowledge the dataset source. The

documentation will also contain instructions for replicating the project (e.g., environment setup, how to run each component).

- **6. Code Repository:** All code (for data processing, modeling, Streamlit app, etc.) will be organized in a GitHub repository. The repository will be structured for clarity (with separate folders for data, notebooks, app, and docs). The professor and peers can access this to review the code implementation details.
- **7. Dataset Link and Description:** We will provide the link to the UCI Statlog Australian Credit Approval dataset (as per UCI Machine Learning Repository) and any other data sources used. If the dataset is small, we may include a CSV copy in the repository for completeness (subject to licensing).
- **8. Presentation Slides (if required):** Should the course require a presentation, we will prepare slides summarizing the project. These would cover the problem statement, approach, key results (with visuals from the dashboards and model performance), and a live demo of the Streamlit app. While not a primary deliverable in the proposal, it is worth noting for completeness.

Each deliverable is aimed at demonstrating a facet of the project: the **Streamlit app** and **dashboards** show practical application, the **model and analysis report** show the data science rigor, and the **executive summary** exemplifies innovative use of AI for communication. We believe this package of deliverables will comprehensively exhibit the knowledge and skills gained, and provide a valuable resource for future reference or portfolio.

Timeline

The project is expected to be completed over approximately **10 weeks**. Below is a tentative timeline with milestones, assuming a start at the beginning of the term:

- **Week 1-2: Project Planning & Data Familiarization** – Finalize project scope and design. Acquire the dataset from UCI and verify its contents. Perform initial data exploration (basic stats, any necessary data cleaning). Formulate evaluation plan (decide on metrics and what constitutes success).
- **Week 3: Data Preprocessing & EDA** – Complete data cleaning (handle missing values, encode categories). Conduct in-depth exploratory data analysis: visualize feature distributions and relationships, identify any anomalies or outliers to handle. Document findings (for example, note which features might be most predictive, check if any class imbalance exists and quantify it).
- **Week 4: Modeling – Baseline and Setup** – Split data into train/test sets. Train a simple baseline model (e.g., logistic regression) for reference performance. Set up the framework for more complex models (Random Forest and XGBoost), including defining a pipeline for preprocessing steps (so that any data transforms can be consistently applied to test data and in the app). Begin initial training of RF and XGBoost with default parameters to get a sense of performance.
- **Week 5: Hyperparameter Tuning & Model Selection** – Perform systematic hyperparameter tuning for the Random Forest and XGBoost models (using cross-validation on the training set). This may involve testing different tree counts, depths, learning rates, etc. Use grid search or random search to find optimal settings. Compare

the cross-validation results and select the best model (or decide if an ensemble of models might improve performance). By end of week, lock down the final model choice. Also, evaluate the final model on the test set and record the metrics.

- **Week 6: Model Interpretability Analysis** – Apply SHAP analysis on the chosen model. Generate global interpretability outputs (feature importance plot via SHAP summary) and local explanations for some test instances. Validate that the explanations make intuitive sense (for example, check that higher income actually increases approval odds, etc., to ensure no data leakage or anomaly). Possibly consult domain literature to ensure our model's behavior aligns with known credit risk factors. Begin integrating these insights (e.g., which top features drive decisions) into the narrative for the final report.
- **Week 7: Develop Streamlit App** – Start building the Streamlit application. Create the UI layout (Sidebar for inputs, main area for output). Write callbacks to take user input, feed through the model, and display results. Test the app locally with a few examples (including edge cases, like minimum or maximum values) to ensure it works robustly. Integrate SHAP or explanation output into the app (e.g., show the contribution of top features for the given input). Iterate on design for clarity (ensure that after a prediction, the app clearly shows "Approved" or "Denied" perhaps with color coding, and an explanation text). By end of week, have a working prototype of the app.
- **Week 8: Tableau Dashboards** – Prepare the dataset (and model outputs if needed) for Tableau. Design and create the visualizations as planned (overall summary, segment breakdowns, etc.). Refine the dashboards to be clear and insightful – e.g., add titles, labels, and brief annotations where helpful (like "High default rate segment" note). Test the interactivity (filters, etc.). If possible, get feedback from a peer or mentor on whether the dashboards effectively communicate insights without additional explanation. Aim to complete the core dashboards this week.
- **Week 9: AI Summary Integration & Refinement** – Implement the GPT-based summary generation. Develop the prompt by trial and error: decide which stats to feed in and how to ask the model to format the summary. Run a few test prompts and adjust for clarity/accuracy. Once a satisfactory summary generation process is set, integrate this into either the Streamlit app (a section that shows the summary) or prepare it as a separate script. In parallel, start compiling the final report document. Fill in sections like Introduction, Data, Methodology (much of which can be based on this proposal, updated with actual results). Include results from the model (metrics, perhaps a table comparing model performances) and snapshots of the dashboards. Also, capture the AI summary as an example in the report.
- **Week 10: Testing & Final Touches** – Rigorously test the entire pipeline: does the Streamlit app handle unusual inputs gracefully? Does the model file load correctly on another machine? Double-check Tableau calculations against original data (consistency verification). Fine-tune any visuals or text for professionalism. Complete the final report with conclusions and possible future work discussions (e.g., how it could be extended with more data or features like credit bureau data, or considerations of fairness). Ensure all citations are in place and create a bibliography. Prepare the deliverables for submission: ensure the code repo is clean and well-documented, the app is ready to run, and all deliverable files are organized. If a presentation is required, create, and practice it.

The above timeline is subject to adjustment based on project progress and any unforeseen challenges (for example, if tuning XGBoost is more time-consuming than expected, it might extend into week 6). However, we built buffer time by overlapping some tasks (e.g., starting the report in week 9). Regular meetings with the project supervisor (weekly check-ins) will be scheduled to report progress and get feedback, ensuring we stay on track. By allocating time to each major component (ML model, app, visualization, documentation), we aim to deliver a polished project by the end of the term.

References

(Selected key references used in the proposal, formatted in the required citation style):

- Quinlan, R. (1987). **Statlog (Australian Credit Approval) [Dataset]**. UCI Machine Learning Repository. *URL:* UCI Dataset Archive – Statlog Australian Credit (690 instances, 14 features).
- Paz, Á., et al. (2023). *Machine Learning and Metaheuristics Approach for Individual Credit Risk Assessment: A Systematic Literature Review*. **Biomimetics, 10**(5), 326. (Noting the prevalence of Random Forest and XGBoost in credit risk modeling).
- Jacquet, F., & Iehl, M.A. (2025). *Toward Explainable AI – SHAP: Bringing Clarity to Financial Decision-Making*. **DZone AI/ML Series**. (Explains SHAP for credit models and importance of interpretability in finance).
- McKinsey & Company. (2024). *Embracing Generative AI in Credit Risk*. (Discusses use cases of LLMs in credit risk reporting and portfolio monitoring).
- Minton, J. (2020). *Assessing Credit Risk with Tableau*. **Medium**. (Illustrates how real-time Tableau dashboards help identify risk trends quickly for financial services).
- Research Preprint (2025). *Explainable AI Credit Risk Assessment using Machine Learning*. (Demonstrated a system with Random Forest, XGBoost, SHAP, SMOTE, achieving high accuracy and interpretability in loan default prediction).