Sacred Heart
UNIVERSITY

AI-Powered Credit Risk Assessment and Approval System

Final Proposal

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

1. Introduction

Credit risk assessment is one of the core functions of banks and financial institutions. Every time a customer applies for a credit card or a loan, the lender must decide whether to approve the application and how risky the borrower is. Traditionally, this has been done using statistical models such as logistic regression and manually defined scorecards. While effective, these approaches can struggle with complex, non-linear relationships and often provide limited interpretability beyond a few hand-crafted rules.

Recent advances in machine learning allow us to build more powerful models that can automatically learn patterns from historical data, handle mixed types of features, and deliver better predictive performance. In parallel, modern data apps and dashboards make it easier to put these models directly in the hands of business users like loan officers and risk managers.

This project implements an **end-to-end AI-powered credit risk assessment system** using the **UCI Statlog Australian Credit Approval dataset**. The system combines:

- A **Random Forest classifier** trained on real credit application data
- A **preprocessing pipeline** that handles missing values and mixed numeric/categorical features
- A **Streamlit web application** that walks the user through a 14-question credit application form and returns a model-driven "approve/deny" decision with probability
- A **feature importance explanation** that highlights which features are most important globally in the model's decisions
- A conceptual integration with **Tableau dashboards** for portfolio-level analysis

The final result mimics a simplified version of a real-world credit underwriting tool and demonstrates practical skills in data science, machine learning, model deployment, and explainability.

2. Business Problem & Objectives

2.1 Business Problem

Financial institutions need to balance **two competing goals**:

1. **Approve as many good customers as possible** (maximize revenue and growth).
2. **Avoid approving high-risk customers who are likely to default** (minimize losses).

A poor credit risk model can either approve too many risky applications or reject too many creditworthy customers. Both are costly.

2.2 Objectives

This project aims to:

1. **Predict credit approval**
   Build a supervised learning model that predicts whether a credit application should be **approved (1)** or **denied (0)** based on applicant attributes.
2. **Achieve strong model performance**
   Use robust metrics (Accuracy, ROC-AUC, F1-score) to measure model quality and ensure the model handles the class imbalance reasonably.
3. **Provide an interactive decision tool**
   Implement a **Streamlit app** where a loan officer can answer a sequence of questions about an applicant and instantly get a decision and probability.
4. **Increase transparency and explainability**
   Provide **feature importance explanations** so the user can understand which variables are most influential overall (e.g., existing debt, years employed, housing status).
5. **Support portfolio-level analytics** (conceptually)
   Use exported data to build **Tableau dashboards** that show portfolio approval rates and risk patterns across different customer segments.

3. Dataset Description

3.1 UCI Statlog Australian Credit Approval Dataset

The primary dataset is the **Australian Credit Approval** dataset from the **UCI Machine Learning Repository**. It consists of:

- **690** credit applications
- **14 input attributes** (a mix of numeric and categorical)
- **1 binary target** variable indicating application **approval status**:
  - `1` = approved
  - `0` = not approved

The attributes are anonymized in the original dataset and labeled as `A1` to `A14`. In the Streamlit app, you've mapped these to more **human-friendly question labels**, such as:

- **A1 → Marital status (Married / Not married)**
- **A2 → Age bucket (18-25, 26-45, 46-80)**
- **A3 → Existing debt amount**
- **A4 → Employment category**
- **A5 → Loan purpose**
- **A6 → Account type**
- **A7 → Years of employment**
- **A8 → Occupation / Job class**
- **A9 → Housing status (Rent / Own house)**
- **A10 → Savings / Investment amount**

- **A11 → Number of existing credit cards / loans**
- **A12 → Bank relationship length**
- **A13 → Existing credit balance**
- **A14 → Credit score range**

This mapping makes the app intuitive for non-technical users while keeping the underlying numeric codes compatible with the original dataset.

3.2 Data Source in the Implementation

In the Streamlit app (`credit_app.py`):

- The model retrieves the dataset from the official UCI URL: `https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/australian/australian.dat`
- The data is read into a pandas DataFrame using `pd.read_csv(..., sep=" ", header=None)`
- The **first 14 columns (0–13)** are used as features `X`
- The **15th column (14)** is used as the target `y`

For evaluation in this write-up, I replicated the pipeline on the locally uploaded `australian.dat` file in the same format.

4. Methodology

The methodology follows a simplified **CRISP-DM** structure: data understanding, preprocessing, modeling, evaluation, and deployment.

4.1 Data Preprocessing

The core preprocessing is implemented using **scikit-learn's `ColumnTransformer` and `Pipeline`**.

1. **Train–test split**
   - The dataset is split into **70% training** and **30% test** using `train_test_split(X, y, test_size=0.3, random_state=42)`.
2. **Separate categorical and numerical attributes**

   In code, you explicitly define:

   ```
   cat_idx = [0, 3, 4, 5, 7, 8, 10, 11]
   num_idx = list(set(range(14)) - set(cat_idx))
   ```

   - **Categorical columns (`cat_idx`)**: 8 columns
   - **Numerical columns (`num_idx`)**: the remaining 6 columns

3. Imputation & encoding

A `ColumnTransformer` handles both data types:

```
preprocessor = ColumnTransformer([
    ("num", SimpleImputer(strategy="mean"), num_idx),
    ("cat", Pipeline([
        ("imp", SimpleImputer(strategy="most_frequent")),
        ("enc", OneHotEncoder(handle_unknown="ignore"))
    ]), cat_idx)
])
```

- Numerical columns: missing values → **mean imputation**
- Categorical columns: missing values → **most frequent category**, followed by **one-hot encoding**
- `handle_unknown="ignore"` ensures the model doesn't crash if it sees an unseen categorical level at prediction time.

4. **Integrated pipeline**

   The preprocessing is combined with a **Random Forest classifier** using `Pipeline`:

```
model = Pipeline([
    ("prep", preprocessor),
    ("clf", RandomForestClassifier(n_estimators=100, random_state=42))
])
```

   This ensures that **the same preprocessing steps** are applied consistently during training and when the Streamlit app makes runtime predictions.

4.2 Modeling

The main classifier used is a **Random Forest**:

- `RandomForestClassifier(n_estimators=100, random_state=42)`
- Handles **non-linear relationships** and mixed feature types well
- Provides **feature importances** for global explainability

In your earlier proposal you discussed trying XGBoost and logistic regression as baselines. In the current implementation, the deployed app focuses on **one carefully tuned and working Random Forest model**, which is often a strong and stable predictor for this kind of dataset.

4.2.1 Training

Inside `load_or_train_model()`:

1. If a saved model file (`model_rf.joblib`) does not exist:
   - Load the data
   - Build the preprocessing + Random Forest pipeline
   - Split the data into train/test sets

    - Fit the model on the training data
2. If the model file **does** exist:
    - Load the pre-trained model from disk using `joblib.load`
    - This makes the Streamlit app start quickly and avoids retraining every time.

This function is wrapped with `@st.cache_resource`, so Streamlit caches the model in memory and avoids reloading/retraining unnecessarily.

4.3 Model Evaluation

During the **first run**, the app trains the model and computes three key metrics:

- **Accuracy**
- **ROC-AUC**
- **F1-score**

The metrics dictionary in your code is:

```
metrics = {
    "accuracy": accuracy_score(y_test, y_pred),
    "auc": roc_auc_score(y_test, y_proba),
    "f1": f1_score(y_test, y_pred),
}
```

Using the same pipeline on the uploaded `australian.dat` file, the obtained performance (70/30 split with `random_state=42`) is approximately:

- **Accuracy:** 0.88
- **ROC-AUC:** 0.93
- **F1-score:** 0.84

These numbers indicate:

- The model is **accurate** on unseen data (~88% of applications correctly classified).
- The **ROC-AUC of ~0.93** shows excellent ability to discriminate between approved and denied cases.
- The **F1-score of ~0.84** balances precision and recall for the positive class (approved), which is critical when both false approvals and false rejections matter.

The Streamlit app displays these metrics in a success message like:

“Model trained: Accuracy = 0.xx, AUC = 0.xx, F1 = 0.xx”

On subsequent runs, the app simply loads the existing model and prints:

“Loaded existing trained model from disk.”

5. Streamlit Application Design

The **`credit_app.py`** file implements a user-friendly Streamlit web interface on top of the trained model.

5.1 Page Configuration

At the top of the script:

```
st.set_page_config(page_title="Credit Approval Predictor", layout="wide")
st.title("✅ Credit Approval Predictor (UCI Australian Dataset)")
```

- Sets a **wide layout** suitable for multi-column forms
- Shows a **clear title** and branding for the app

A left **sidebar** titled "Application Controls" allows users to:

- Start a **new credit application**
- See which question (step) they are on
- See the current progress

5.2 Session State & Multi-Step Workflow

The app uses `st.session_state` to store:

- `step` – current question step
  - `-1` → initial applicant info page (name, sex, email)
  - `0-13` → the 14 credit application questions
- `answers` – a list of 14 numeric values corresponding to features A1–A14
- `finished` – whether the questionnaire is completed
- `decision` – final approve/deny result
- `probability` – predicted probability of approval
- `user_name`, `user_sex`, `user_email` – basic applicant information

There is a **reset function**:

```
def reset_application():
    st.session_state.step = -1
    st.session_state.answers = [0.0] * 14
    st.session_state.finished = False
    st.session_state.decision = None
    st.session_state.probability = None
```

This allows users to **start a fresh application** at any time.

5.3 Question Logic: Human-Friendly Mapping

The core UI for user inputs is implemented in a helper function:

```
def ask_feature(step: int, current_value: float):
    ...
```

For each `step` from 0 to 13, the function:

1. Presents a **human-friendly label** and options (`st.selectbox` for categorical, `st.number_input` for numeric).
2. Maps the user's choice back to a **numeric value** compatible with the trained model.

Example:

- **Step 0: Marital status (A1)**
- `options = ["Married", "Not married"]`
- `mapping = {"Married": 1.0, "Not married": 0.0}`
- `choice = st.selectbox("Marital status", options, ...)`
- `return mapping[choice]`
- **Step 1: Age bucket (A2)** – bucketed into age ranges, mapped to midpoints:
- `options = ["18-25", "26-45", "46-80"]`
- `mapping = {`
- `    "18-25": 21.0,`
- `    "26-45": 35.0,`
- `    "46-80": 60.0,`
- `}`
- **Step 2: Existing debt (A3)** – numeric:
- `return st.number_input("Existing debt amount", ...)`
- **Step 8: Housing status (A9)**:
- `options = ["Rent", "Own house property"]`
- `mapping = {"Rent": 0.0, "Own house property": 1.0}`
- **Step 13: Credit score (A14)** – ranges mapped to representative numeric values.

This pattern is repeated for all 14 features, transforming the anonymized UCI fields into realistic credit application questions.

5.4 Navigation & User Experience

For steps `0-13`, the app displays:

- A subheader: "📝 Application Questionnaire"
- The current question via `ask_feature(step, current_val)`
- Two buttons in a 2-column layout:
    - **Next** ➡️ – moves to the next question and saves the answer
    - ⬅️ **Previous** – goes back one question if the user wants to edit

On the **first page (step = -1)**, the app shows "👤 Applicant Information" with:

- `st.text_input` for **Applicant Name**
- `st.selectbox` for **Sex** (Male, Female, Other)
- `st.text_input` for **Email Address**
- A **"Start Application"** button to begin the questionnaire

This multi-step wizard design makes the application process feel natural and prevents the interface from becoming cluttered.

6. Prediction & Explanation

6.1 Generating the Prediction

Once all 14 questions are answered, `st.session_state.finished` is set to `True`. The app:

1. Builds a **single-row pandas DataFrame** from the 14 numeric answers.
2. Passes this DataFrame to the trained **Random Forest pipeline**:
    - `model.predict(...)` → approval decision (0 or 1)
    - `model.predict_proba(...)` → probability of approval
3. Displays the result:
    - If approved (1):
        - A green success message such as:

          "Application Approved ✅ with probability XX%"

    - If denied (0):
        - A red warning message such as:

          "Application Denied ❌ with probability XX%"

4. Uses the applicant's name (if provided) to personalize the message.

6.2 Global Feature Importance Explanation

To give the user insight into **what the model cares about overall**, the app computes **feature importances** from the Random Forest classifier:

```
prep = model.named_steps["prep"]
clf = model.named_steps["clf"]

try:
    feature_names = prep.get_feature_names_out()
except Exception:
    feature_names = [f"feature_{i}" for i in range(clf.n_features_in_)]

importances = clf.feature_importances_
```

Then it builds a DataFrame:

```
fi = pd.DataFrame({
    "feature": feature_names,
    "importance": importances,
}).sort_values("importance", ascending=False).head(5)
```

Finally, a horizontal bar chart is plotted:

- y-axis: **Top 5 most important transformed features**
- x-axis: **Importance scores**

With title: *"Top 5 features affecting approval decision"*

This helps the user understand which variables the model considers most influential, such as:

- Credit history or previous defaults
- Level of existing credit balance
- Length of bank relationship
- Age bucket or years of employment

In your original proposal you planned to use **SHAP** values to generate **local explanations per individual prediction**. In the current implementation, the app imports `shap` but primarily uses **Random Forest feature importances** as a practical and stable explanation method. SHAP remains a logical extension for future work if computational resources and deployment constraints allow it.

7. Tools & Technologies

The `requirements.txt` for the app includes:

```
streamlit
pandas
numpy
scikit-learn
joblib
matplotlib
typing_extensions
protobuf
altair
```

Key tools:

- **Python** for all data science and app logic
- **pandas** & **NumPy** for data manipulation
- **scikit-learn** for preprocessing, model training, and evaluation

- **RandomForestClassifier** as the main classifier
- **joblib** for saving/loading the trained model
- **Streamlit** for building the interactive UI
- **matplotlib** for plotting feature importance charts
- **Tableau** (outside this code base) for building portfolio-level dashboards
- **SHAP** (imported, conceptual use) for advanced explainability

8. Tableau Dashboards (Conceptual Component)

Although the Streamlit app focuses on **individual application decisions**, the same dataset and model outputs can be exported to **Tableau** to support **portfolio-level analysis**. The planned Tableau dashboards include:

1. **Portfolio Overview Dashboard**
    - Overall approval vs. rejection rates
    - Distribution of predicted probabilities of approval
    - Key KPIs (e.g., average approval probability, share of high-risk cases)
2. **Segmented Risk Dashboard**
    - Approval rates and average predicted risk segmented by:
        - Age group
        - Employment category
        - Housing status (rent vs. own)
        - Loan purpose
    - Filters to focus on specific sub-segments (e.g., only self-employed customers)
3. **Credit Exposure Dashboard**
    - Distribution of existing credit balances
    - Relationship between existing debt and model-predicted approval probability
    - Identification of segments with high exposure but lower approval probability

These dashboards are meant to mirror how risk managers and executives monitor the health of the portfolio and identify **high-risk customer segments** or **emerging trends**.

9. Results & Discussion

9.1 Predictive Performance

The Random Forest model achieves:

- **Accuracy ~ 0.88**
- **ROC-AUC ~ 0.93**
- **F1-score ~ 0.84**

on a 30% hold-out test set.

This indicates:

- The model is **well-calibrated** and distinguishes approved from denied applications effectively.
- A high **ROC-AUC** suggests the model can rank applicants by risk reliably.
- The F1-score shows a good balance between approving good customers and avoiding bad risks.

9.2 User Experience

The Streamlit app turns a technical ML model into a **simple decision tool**:

- Loan officers do not need to know anything about Random Forests, pipelines, or the UCI dataset.
- They simply:
    1. Enter basic applicant info
    2. Answer 14 guided questions
    3. See the decision and probability instantly

The step-by-step structure reduces cognitive load and makes the application process more realistic and interactive.

9.3 Explainability

The **feature importance bar chart** provides an accessible explanation of what drives decisions **at the model level**. While it does not yet provide **per-customer SHAP explanations**, it still:

- Builds trust by showing that the model is not purely a black box
- Allows stakeholders to sanity-check that important features (e.g., existing debt, number of prior credits, credit score) are ranked high

10. Limitations & Future Work

10.1 Limitations

1. **Single dataset & limited size**
    - Only 690 applications, all from one historical dataset.
    - Real-world models would be trained on much larger and more recent data.
2. **Anonymized / synthetic feature semantics**
    - The original attributes are anonymized; the mapping to real-world attributes in the UI is approximate and illustrative.

3. **No fairness or bias analysis**
    - The project does not evaluate potential bias across groups (e.g., by gender, age).
4. **Explainability constrained to global feature importances**
    - Local explanations per applicant (e.g., SHAP force plots) are not currently shown in the Streamlit UI.
5. **No generative AI narrative integrated yet**
    - While the proposal discussed using GPT to auto-generate portfolio summaries, the current code focuses on the core prediction app.

10.2 Future Enhancements

1. **Integrate SHAP-based local explanations**
    - Show per-applicant SHAP values to answer: "Why was this particular application denied/approved?"
2. **Add GPT-based executive summaries**
    - Generate natural-language summaries of portfolio risk using model outputs and key statistics.
3. **Multi-model comparison**
    - Implement and compare Logistic Regression, XGBoost, and perhaps a shallow neural network, and allow the user to switch between models in the app.
4. **Fairness and robustness analysis**
    - Evaluate whether model performance is consistent across demographic segments and test for potential biases.
5. **Richer dashboards**
    - Connect the trained model outputs directly to Tableau or another BI tool to support near real-time risk monitoring.

11. Conclusion

This project demonstrates a complete **AI-powered credit risk assessment pipeline**, starting from raw data and ending with a working **Streamlit web application** that a loan officer can actually use.

Key achievements include:

- Training a robust **Random Forest classifier** on the **UCI Australian Credit dataset**, achieving **high predictive performance** (AUC ≈ 0.93).
- Implementing a **scikit-learn pipeline** that handles both numerical and categorical features with appropriate imputation and encoding.
- Designing a **wizard-style Streamlit interface** that collects realistic credit application information through 14 intuitive questions.
- Providing **global feature importance explanations** to improve transparency and user trust.

- Establishing a clear architecture that can be extended with **Tableau dashboards, SHAP explanations, and GPT-based narrative summaries**.

Overall, the project shows how machine learning, interactive web apps, and data visualization can work together to **streamline credit approval decisions**, **reduce manual effort**, and **enhance risk understanding** in a financial institution.