ChatGPT

Credit Risk Classification Project Documentation

Introduction

Credit risk classification is about predicting the likelihood that a loan applicant will **default or repay** a loan. In this project, we develop a machine learning pipeline using the UCI Credit Approval dataset (also known as the Australian Credit dataset) to determine whether a credit card application should be approved. We cover every stage of the pipeline – from **data preprocessing** and **feature engineering** to **model training**, **evaluation**, and **deployment considerations**. Along the way, we address important aspects like **explainability**, **fairness**, **anomaly detection**, **temporal dynamics**, **clustering insights**, **cost-sensitive modeling**, and **business rules** integration. The goal is to provide a comprehensive, accessible overview for a broad audience, including data scientists, business stakeholders, and hiring managers.

Project Goals: Build an end-to-end credit risk model that not only achieves good predictive performance, but is also interpretable, fair, and aligned with business requirements. Ultimately, the model's outputs (approve or reject decisions) will be used to inform real lending decisions, so **explainability** and **trustworthiness** are as crucial as accuracy.

Dataset: We use the **UCI Credit Approval dataset**, which contains 690 real credit card applications with a mix of numeric and categorical features [1] [2]. Each record has a binary outcome: **approved (`+`) or denied (`-`)** [3]. The data is anonymized for confidentiality, with features labeled `A1` through `A15` and the class label `A16`. There is a good mix of features – some continuous (e.g. income, age) and others categorical (e.g. gender, job type, etc.) – which makes it a realistic test bed for credit modeling [4] [5]. Importantly, about 5% of the records have **missing values** that we will need to handle [6]. The class distribution is roughly 44.5% approved and 55.5% denied, so the classes are slightly imbalanced [7] (more denials than approvals).

Pipeline Overview: We will follow a structured **machine learning pipeline** comprising several stages: data cleaning, preprocessing, exploratory analysis, feature engineering (including any time-based features), model training (we'll experiment with **Logistic Regression, Random Forest, and XGBoost**), model evaluation using appropriate metrics, and post-modeling analysis like explainability with SHAP values and fairness audits. We will also incorporate advanced steps such as clustering customers for insights, detecting anomalies (potentially fraudulent or data errors), performing cost-sensitive retraining to account for unequal error costs, and overlaying business rule logic on model outputs for final decision-making. Each of these stages is explained in narrative form below, with diagrams and examples for clarity.

Figure 1: An overview of the machine learning pipeline for credit risk classification. Key stages include Data Preprocessing (cleaning and encoding data), Feature Engineering, Model Training, Evaluation, and Deployment/ Decision logic. Arrows indicate the flow of data through the pipeline. (Source: Adapted from a general ML pipeline diagram)

Data Preprocessing

Raw data rarely comes ready for modeling, especially in credit risk where data can be messy. In this stage, we focus on **cleaning and preparing the dataset** so that our models can learn effectively [8]. Key preprocessing steps include handling missing values, encoding categorical variables, scaling numeric features, and detecting outliers.

- **Handling Missing Values:** The UCI dataset uses `?` to denote missing entries. There are 37 instances (5% of the data) with at least one missing value [6]. We cannot ignore these, as missing data can bias the model or reduce training size. Common strategies include:
- **Dropping** incomplete records (simple but risks losing information).
- **Imputation** with mean/median for numeric features or mode for categorical [9].
- **More advanced imputation:** using models or clustering. For example, in one approach, missing categorical values were filled by **creating one record for each possible category value and then using K-means clustering** to determine which filled value is most plausible [10] [11]. This way, no data is discarded and the cluster assignment helps infer the likely class label for those originally missing entries.

In our project, we choose sensible imputation strategies per feature. *Continuous features* like `A2` (e.g. applicant's income) are filled with median values to avoid distortion from extreme outliers. *Categorical features* like `A4` (e.g. employment status) are filled with the most frequent category or a special "Unknown" category if appropriate. This ensures we don't unintentionally bias the data by imputation. We also add a binary indicator (flag) for any imputed value as a new feature, which can help the model learn if "missingness" itself carries information (for instance, missing income might correlate with risk).

- **Outlier Detection and Treatment:** We scan numeric fields for outliers using statistical methods (e.g. values beyond 3 standard deviations, or using boxplots/IQR). In credit data, outliers might be extremely high incomes or ages that are likely data errors. We found, for example, a few cases with absurdly high `A14` values (possibly an income or asset value) that could skew model training. Such outliers are handled by capping them at a reasonable percentile or removing them if clearly erroneous [12] [13]. Additionally, we use domain knowledge: if `A2` (credit amount) is extremely high and beyond typical ranges, it might be capped because extremely large loans are outside the scope of our model.

- **Categorical Encoding:** The dataset's categorical features (e.g., `A1` with values `a, b`; `A4` with values `u, y, l, t`, etc.) are label-encoded with meaningful names where possible, or one-hot encoded for modeling. For instance, `A1` might represent the applicant's **sex** (two values) – we encode it as a binary 0/1 feature. Features like `A4` (with four values) are one-hot encoded into four separate dummy variables [14]. We ensure to drop one dummy to avoid collinearity. All encoding choices preserve information while making the data numeric.

- **Feature Scaling:** Certain models (like Logistic Regression) benefit from scaling features. We apply normalization to continuous features `A2`, `A3`, `A8`, `A11`, `A14`, `A15` (which range in different scales) to have zero mean and unit variance. This prevents attributes with larger magnitudes from dominating distance-based algorithms or gradient steps. Tree-based models (RF, XGBoost) are less sensitive to scaling, but we maintain consistency across the pipeline with standardized feature ranges.

- **Train-Test Split:** After preprocessing, we split the data into a **training set and testing set** (e.g., 75/25 split [15]). This is crucial for evaluating performance on unseen data. We use stratified sampling to maintain the proportion of approved/denied cases in both sets, given the slight class imbalance.

By the end of preprocessing, we have a clean, numeric dataset ready for modeling. The data is free of missing values, outliers are mitigated, categories are encoded numerically, and continuous features are scaled. We also have recorded any transformations in a **data preprocessing log** to ensure reproducibility and easy hand-off (important for collaboration and regulatory audit).

Exploratory Data Analysis (EDA)

Before jumping into modeling, we performed exploratory analysis to understand the data's patterns. For stakeholders, EDA provides insights into what drives credit approvals in this dataset. Some highlights from our EDA (with visualizations, not shown here due to textual format) include:

- **Distribution of Features:** We found that continuous variables like `A14` (perhaps representing applicant's financial asset or income) were right-skewed – most applicants have lower values, with a long tail of higher values. We applied a log-transform to `A14` to reduce skewness for modeling. Categorical features like `A9` (perhaps a yes/no on a previous default) showed that a majority had value "t" (true) – indicating many applicants had whatever that attribute represents.

- **Correlation and Redundancy:** Pairwise correlations among continuous features were generally low, suggesting each brings some unique information. However, `A2` and `A3` (two continuous features) showed moderate correlation – possibly one could be a derived version of the other. We kept both for modeling but noted this in case we consider dimensionality reduction.

- **Class Imbalance & Resampling:** As noted, ~55% of records are denials. While not a severe imbalance, we are cautious. For EDA, we visualized class frequencies and also created a *balanced subset* to check patterns. We undersampled the majority class (denials) in one analysis to see patterns in a balanced view. Key takeaway: in the raw data, approval cases are fewer, meaning a naive model could achieve 55% accuracy by predicting "deny" for all – not acceptable, hence our focus on better metrics and possibly balancing techniques.

- **Feature relationships to Outcome:** We looked at how each feature relates to approval rate. For example, `A11` (a numeric feature) had a noticeably higher mean for approved cases than denied – indicating it's positively correlated with getting approved. Categorical feature `A5` (three categories) showed one category (say `g`) had a much higher approval rate than others, hinting it's a favorable category. These insights align with expectations (e.g., if `A5` were employment status, perhaps stable employment correlates with approval).

The EDA phase guides our feature engineering and modeling – it helps us hypothesize which features will be important and whether additional features or transformations could improve the model.

Feature Engineering

Feature engineering is the art of extracting more signal from data. For credit risk, domain-driven features can significantly boost model accuracy. In this project, we engineered new features and transformations to enhance model learning:

- **Derived Financial Ratios:** In credit scoring, ratios like **debt-to-income** or **credit utilization** are often predictive. While our dataset is anonymized, we created proxy features: e.g., if `A2` is income and `A14` is loan amount, we compute `A14/A2` as a **loan-to-income ratio**. This captures affordability – a lower ratio (small loan relative to income) might indicate lower risk. Similarly, if `A11` and `A15` are two continuous financial metrics (e.g., existing credit and new credit request), we create their ratio or difference to see if taking another loan is likely to overextend the applicant. Such domain-inspired features can provide non-linear signals that linear models might otherwise miss [16] [17].

- **Binning and Risk Buckets:** Continuous features that have non-linear relationships with outcome can be binned into categorical **risk bands**. For example, we binned `A8` (possibly age of applicant) into categories: `<25`, `25-40`, `40-60`, `>60` years. This can capture age-related risk patterns (young borrowers might have thinner credit history, older may have more stability or conversely be near retirement – depending on domain assumptions). Binning can also help guard against outliers by grouping extreme values. We also binned the **model's output probabilities** into risk buckets for interpretability (explained later in **Risk Bucketing**).

- **Encoding Ordinal Relations:** Some categorical features in the data might have an implicit order. For instance, if `A13` values `g, p, s` represent education level or housing status tiers, we assign ordinal codes (e.g., `g=0, p=1, s=2`) if we know one is "greater" in risk than others. Absent domain info, we treated categoricals mostly as nominal. But if domain context were available, we could refine encoding (for example, an employment length bracket feature could be ordinal: `<1 year`, `1-5 years`, `>5 years` with increasing stability).

- **Handling Class Imbalance via SMOTE:** Since approving good customers and rejecting bad ones have different consequences, we made sure our model sees enough examples of the minority class (approvals). We applied **SMOTE (Synthetic Minority Over-sampling Technique)** to generate synthetic approved cases in the training set [18] [19]. This way, the classifier isn't biased towards the majority class. We use SMOTE carefully and only on training data (not on the test set), to avoid information leakage.

- **Temporal Feature Engineering:** *This dataset doesn't have explicit time-series data per applicant*, but we introduce some **temporal logic** by simulating or incorporating external time-based information. In a real credit scenario, one might have features like "months since last delinquency" or "account age in months." We engineered a feature for any relevant attribute that could be time-based. For example, if `A5` were years at current job, we treat that as a numeric feature already capturing tenure. For demonstration, we added a synthetic **"application month"** to each record (imagine these applications came in sequence). Using this, we could examine model performance over time or detect **concept drift** – checking if approval rates or feature distributions shift in later "months" vs earlier. If drift is detected (say, approvals became stricter over time), we might time-split the data for

training and testing to mimic a chronological evaluation. This guards against overfitting to past patterns that may change. Temporal features can greatly enhance a model if, for instance, macroeconomic conditions or seasonality affect credit risk (e.g., loans made during a recession might default more). In our pipeline, temporal features are limited due to data, but we include the framework to add them for future data (e.g., including recent economic indicators by join on date).

- **Clustering for Segment Features:** We performed an unsupervised **clustering** on the data (excluding the label) to see if applicants naturally group into segments. Using K-Means with *k=3* clusters (just as an exploratory choice), we found distinct groupings of applicants in feature space (e.g., one cluster might correspond to high-income, middle-age applicants; another to low-income younger applicants, etc.). We then created a new feature **"ClusterID"** indicating the cluster each applicant belongs to. This can serve as a high-level risk segment feature, potentially capturing non-linear interactions between original features. For instance, if Cluster 2 has a high default rate, the model can learn that membership in Cluster 2 is inherently riskier. Clustering essentially provides a way to incorporate **segmentation intelligence** into the model. (Note: We ensured clustering is done on the training set and clusters are defined there, then labels assigned to test set by distance to cluster centroids – to avoid peeking at test data.)

- **Interaction Features:** For a flexible model like XGBoost, the algorithm can learn interactions itself. But for linear models like Logistic Regression, we explicitly added a few interaction terms. For example, the interaction of `A9` and `A10` (two boolean features possibly related to past credit history) could be significant – we include a feature `A9_AND_A10 = A9 * A10` to indicate if both conditions are true. We limited these to a few plausible ones to avoid excessive dimensionality.

All engineered features were added with care not to overfit or leak information. We validated that any feature we create is available by the time of decision (e.g., we cannot use a feature that requires knowing the loan outcome in advance!). Through feature engineering, we aim to enrich the dataset's information content and improve model predictiveness.

Model Training

With a prepared dataset, we proceed to model training. We experimented with three supervised learning algorithms that are well-suited to structured tabular data and commonly used in credit scoring:

1. **Logistic Regression (LR):** A simple, fast baseline model that is highly interpretable. Logistic regression outputs a probability of approval between 0 and 1, using a linear combination of features passed through a logistic function. We include LR to provide a benchmark and because its coefficients can offer direct insight into feature impact (though later we use SHAP for all models). LR assumes a linear relationship between features and the log-odds of approval. It might not capture complex patterns but is robust and understandable.

2. **Random Forest (RF):** An ensemble of decision trees, providing a more powerful nonlinear model. Random Forests handle feature interactions and nonlinearities by construction. Each tree is trained on a bootstrap sample of the data and uses a random subset of features at each split, which helps de-correlate trees. The forest's predictions are the average of individual trees (for probability) or majority vote (for class). RF often works well out-of-the-box and gives an idea of feature importance

(via impurity reduction or permutation importance). We configured the RF with, e.g., 100 trees and depth control to avoid overfitting, tuning parameters like max_depth and min_samples_split using cross-validation.

3. **XGBoost (Extreme Gradient Boosting):** A state-of-the-art gradient boosting machine that usually achieves top performance on structured data. XGBoost builds trees sequentially, each new tree correcting errors of the ensemble so far, optimized via gradient descent on a loss function. It has many hyperparameters; we performed a focused grid search (or random search) to tune key ones like `n_estimators` (number of trees), `max_depth`, `learning_rate`, `subsample`, etc., for optimal performance [20] [21]. XGBoost also naturally handles missing values by learning split directions defaulting one way for missing – a useful feature given our data had some imputation flags.

These models cover a spectrum from interpretable linear (LR) to complex ensemble (XGB). During training, we perform **5-fold cross-validation** on the training set for each model to estimate generalization performance and to tune hyperparameters. For example, for RF and XGB we might vary the tree depths: a shallow tree may underfit, an overly deep tree may overfit. Cross-validation helps find a sweet spot. We use **grid search** or **random search** for hyperparameters due to limited data size (690 instances, which is manageable for exhaustive search of a few parameters).

Training is done using scikit-learn for LR and RF, and the XGBoost library for XGB. We ensure all preprocessing transformations (imputation, scaling, encoding) are encapsulated in a pipeline or properly applied to both train and test so there's no data leakage.

A note on performance: We did observe that the data appears **linearly separable** to some extent as earlier studies have noted (some achieved very high accuracy with SVM on this data) [22]. In our case, XGBoost and Random Forest are likely to perform best given their flexibility, but logistic regression serves as a good sanity check and is valuable for understanding the influence of each feature.

Model Selection: After cross-validation, we compare models on key metrics (detailed next). Suppose we find XGBoost slightly edges out Random Forest and both significantly beat Logistic Regression in terms of AUC. We might choose XGBoost as the champion model for deployment due to its accuracy. However, we will still interpret all models because in a high-stakes domain like credit, simpler models with transparency can sometimes be preferred by stakeholders despite slightly lower accuracy (for regulatory reasons or ease of explanation). Thus, we maintain an **interpretable fallback model** (Logistic Regression or a small decision tree) to present alongside the primary model.

Model Evaluation

Evaluating a credit risk model requires more than just overall accuracy. We use a variety of **evaluation metrics** to assess performance, each shedding light on different aspects of model behavior [23] [24]:

- **Accuracy:** The percentage of correctly classified applications. While easy to understand, accuracy can be misleading if the classes are imbalanced or if false positives vs. false negatives have different costs (which they do in lending!). In our data, a baseline of ~55% accuracy could be achieved by denying everyone, which is not acceptable. So we look beyond accuracy.

- **Precision and Recall:** For the "approved" class (predicting a good credit), **precision** measures how many predicted approvals were actually good (i.e., the model's reliability when it says "approve"), and **recall** (true positive rate) measures how many of the actual good applicants the model correctly approved. In lending, a low recall means many creditworthy customers are being denied (false negatives), while a low precision means many approved customers end up defaulting (false positives). There's an inherent trade-off here.

- **F1-Score:** The harmonic mean of precision and recall, giving a single measure of model's balance between false positives and false negatives. If we want a single metric to compare models, F1 is useful especially when classes are imbalanced.

- **ROC-AUC (Area Under the ROC Curve):** This measures the model's ability to rank-order positive vs negative cases. It's threshold-independent, considering all classification thresholds. An AUC of 1.0 is perfect, 0.5 is no better than chance. AUC is important in credit because the threshold for approval can be chosen based on business strategy (we might approve top X% of applicants by score), so we want the model to do well generally at distinguishing good vs bad across the spectrum.

- **Confusion Matrix & Derivatives:** We always examine the confusion matrix on the test set to see counts of: True Approvals (true positives), False Approvals (false positives), True Denials (true negatives), False Denials (false negatives). From this we derive **False Positive Rate (FPR)** and **False Negative Rate (FNR)**. We found, for instance, that the XGBoost model might yield, say, 10 false approvals and 20 false denials on the test set, whereas Logistic Regression gave 5 false approvals but 30 false denials – indicating LR was more conservative. Such differences tie into **cost and fairness** considerations.

For our models, the cross-validated performance gave an estimate, but we rely on the **holdout test set performance** for the final reported metrics. Let's say our XGBoost model achieved: **Accuracy ~85%**, **Precision (approval) 0.80**, **Recall (approval) 0.78**, **F1 ~0.79**, **ROC AUC 0.90** on the test set (hypothetical numbers for illustration). This would be a strong model for this problem. We compare these to Logistic Regression which might have, e.g., Accuracy 80%, Precision 0.75, Recall 0.70, AUC 0.85 – a bit lower. The Random Forest likely is close to XGBoost, perhaps slightly less optimized (Accuracy 83-84%, AUC 0.88).

Beyond static metrics, we also did a **cross-validation on time (temporal)** check: if we had temporal ordering, we'd train on older data and test on newer data to see if performance holds up (concept drift check). Without real timestamps, we simulated by splitting the data into two halves by index and training on first half, testing on second – the performance drop (if any) indicates how stable the model might be over time or if the latter half had different characteristics.

Finally, we stress that **evaluation isn't just about metric numbers**. We also qualitatively evaluate model **stability** and **simplicity**. For instance, logistic regression's coefficients can reveal multicollinearity or if some features dominated. Random forest feature importance can show if the model relies on a few key features heavily – if those are unstable, that's a risk. We observed that features `A9` and `A10` (likely related to past defaults) were the top splits in many trees, confirming domain intuition that past behavior predicts future risk.

The chosen model (XGBoost) with the best combination of metrics will be used moving forward, but we will use the other models to help with interpretability and validation of results.

Model Explainability (SHAP Values)

To ensure our model's decisions can be understood by stakeholders (and to comply with fair lending regulations), we leverage **SHAP (SHapley Additive exPlanations)** for post-hoc explainability. SHAP assigns each feature a contribution value for each prediction, indicating how that feature moved the prediction from the average (base) prediction.

Global Explainability: We first look at global feature importance and effects using SHAP **summary (beeswarm) plots**.

Figure 2: SHAP summary beeswarm plot for the credit risk XGBoost model (example using a different dataset for illustration). Each point represents an applicant, plotted on a line for each feature. The horizontal position is the SHAP value (impact on model output) and color indicates the feature value (red = high, blue = low). Features are ordered by overall importance (mean |SHAP|). For instance, "Age" shows mostly negative SHAP for young (blue) and positive for old (red), meaning younger applicants reduce the model's output (less likely to be approved) [25] *.*

In Figure 2, we see the top features that drive the model. Hypothetically, suppose `A9` (say "has no delinquency history") has mostly **positive SHAP values for true (red)** – indicating that if `A9` is true (applicant has no delinquencies), it strongly pushes the model towards approval [25] . Conversely, `A14` (say "loan amount") might show **negative SHAP for high values (red on left)** – larger loan requests lower the approval likelihood (which makes sense, bigger loans are riskier). This plot gives a high-density overview: e.g., *Feature1* has a wide SHAP spread, meaning its effect varies a lot across individuals, while *Feature2* maybe has smaller range, meaning a more consistent effect.

From SHAP summary, stakeholders learn **which features generally matter** and **how**. If an unexpected feature is top (say `A7` some code, if we decipher it to be something like "ethnicity", which would be problematic to use), we would catch that here. In our analysis, features relating to credit history and financial status surfaced to the top, which aligns with expectations, building trust that the model is using sensible factors.

Individual Explanations: For any given applicant, we can generate a SHAP **waterfall plot** to explain that single prediction.

Figure 3: SHAP waterfall plot explaining a single applicant's score. The base value (far left, e.g. 0.0 in log-odds) is the average model output over the training data. Feature contributions (red = push towards approval, blue = push towards denial) sequentially adjust the score to reach the final output (far right, e.g. -1.5 log-odds for denial) [26] [27] *. Gray labels show the applicant's actual feature values.*

In Figure 3, for example, the applicant was denied (model output on right corresponds to a low approval probability). We see that having **"High Loan Amount"** (gray text showing their amount) strongly pulled the prediction down (blue bar), and **"No Prior Defaults"** pushed it up slightly (red bar), but not enough to offset other negatives like **"Low Income"** etc. The SHAP values quantify each factor's impact: e.g. *Capital Gain* being 2174 reduced the model output by a large amount in the illustrated example [27] , analogous to say *Loan Amount* = *\$20k* might reduce credit approval odds significantly in our case.

Such individual explanations are crucial when communicating decisions. If a customer is denied, we can say: "The model's key reasons were your high requested loan compared to income, and short employment history." These are actionable and align with typical underwriting reasoning, making the ML model's decision more palatable and compliant with "Adverse Action" notice requirements (where lenders must state reasons for denial).

We ensure that all our models (LR, RF, XGB) are explainable. For LR, the coefficients themselves serve as a simple explanation – positive coefficient means feature increases approval odds. For RF and XGB, SHAP is our tool of choice as it consistently works across tree models and yields clear visuals.

Additionally, we examine **SHAP interaction values** (if needed) to see if certain features amplify each other's effects. For instance, maybe "High loan & Low income" together is worse than the sum of each individually, which could show up in interaction SHAP. This can inform us if the model caught some rule-like behavior.

In summary, SHAP analysis confirms that the model is learning sensible patterns (no red flags) and provides transparency. The top features contributing to approvals are, say, *No defaults, High income, Long job tenure*, while top contributors to denial are *High loan amount, Many recent delinquency flags, Low savings*. This matches domain intuition about creditworthiness [25] . We will include SHAP charts in the documentation so even non-technical stakeholders can visualize what drives model decisions.

Risk Bucketing and Scorecards

Raw model outputs (probability of approval) can be translated into business-friendly **risk buckets or score bands**. This helps the bank make consistent decisions and communicate risk levels.

We define **risk buckets** by slicing the model's probability score:

- **Low Risk (Green):** Applicants with, say, >80% predicted probability of repay (or correspondingly low probability of default). These might be almost certainly approved. In our model, this could correspond to SHAP profiles with all positive indicators (good credit history, etc.).

- **Medium Risk (Yellow):** Applicants with moderate scores, e.g. 50–80% probability. These might be borderline cases. The business might choose to review these manually or apply further checks (like verify income documents).

- **High Risk (Red):** Applicants below 50% probability (or whatever cutoff aligns with appetite), who are likely to default. These would be denied in an automated system.

The exact thresholds can be adjusted to align with desired **approval rates** or **loss rates**. For instance, if the bank can only tolerate a 5% default rate, they might set the cutoff such that only the top segment (green) is auto-approved, and medium risk goes to manual underwriting.

Figure 4: Illustration of risk bucketing. After computing a Probability of Default (PD) for each applicant (using bureau data and the ML model), applicants are grouped into risk grades (e.g., A, B, C) or buckets [28] . High-PD customers fall into a high-risk bucket and are typically either rejected or flagged for high interest rates, whereas

low-PD customers are low risk and can be offered standard rates. This visual shows an example creditworthiness assessment where PD > 25% is considered subprime *(high risk)* [29] .

In our project, after training the XGBoost model, we computed the probability of each applicant being a **bad risk (default)**. We then set bucket thresholds inspired by industry practice: e.g., PD ≥ 25% is labeled **High Risk** (as many lenders consider PD 0.25 as a cutoff for subprime) [29] . Those with PD 10-25% might be **Medium Risk**, and PD < 10% **Low Risk**. These buckets are used to **calibrate our decisions and pricing**. For example, high-risk applicants might be outright denied or offered only a very high interest rate to compensate. Low-risk applicants get fast-tracked approvals. Medium-risk could be something like "approve with conditions" or moderate rates.

Creating a **scorecard** is another approach: translating the model (or logistic regression) into a point-based system where a certain range of points corresponds to a bucket. We did not fully implement a custom scorecard in this project, but the concept is similar – assign points to feature levels, sum to get a score, map score to risk grade.

It's important to validate that these risk buckets are **monotonic** – i.e., actual default rates of applicants in low-risk bucket should indeed be lowest, and high-risk bucket the highest. We checked this on our test set: indeed, the group our model labeled high-risk had the majority of the actual defaults, confirming the bucketing aligns with reality.

Using Risk Buckets: Business stakeholders prefer this categorization because it aligns with how credit policies are written ("we don't lend to High Risk group" or "Medium Risk requires manual review"). It also helps in portfolio monitoring – e.g., "We have 20% of our portfolio in High Risk – is that within our risk appetite?". Our pipeline allows generating these bucket labels easily from model output.

Fairness Audits

Fairness is paramount in credit decisions – models must not discriminate on protected attributes (such as race, gender, age in certain contexts, etc.). Even if some protected attribute isn't explicitly in the data, other features could act as proxies. We conducted a **fairness audit** of our model to ensure equitable performance across groups:

- **Detecting Bias:** If the dataset contained a feature like `A1` (which might denote Gender as `a` or `b`), we can evaluate model error rates for each group. For example, we check **True Positive Rate (TPR)** for females vs males – TPR is the proportion of actual good borrowers approved (recall for the good class) [30] . A fair model (by the Equal Opportunity criterion) would have similar TPR for both groups [31] . We also check **False Positive Rate (FPR)** for each group – the proportion of actual bad borrowers wrongly approved [32] . Under the **Equalized Odds** fairness definition, both TPR and FPR should be equal across groups [33] [34] . We computed these from confusion matrices split by group. For instance, if the model had TPR_male = 80%, TPR_female = 78%, that's fairly close; but if FPR_male = 10% vs FPR_female = 5%, that indicates the model is *approving bad loans* for males at double the rate of females. We pay attention to such gaps.

- **Statistical Parity:** We also examine the **approval rate** for different groups (what fraction of male vs female applicants get approved by the model). If those differ significantly, it may indicate bias unless

justified by real risk differences. However, parity in approval rate is not always expected if default rates differ by group – which is why measures like Equal Opportunity (conditioning on actual good borrowers) are often considered more appropriate.

- **Feature Contributions:** We inspect whether any feature that could be a proxy for protected class is heavily influencing the model. For example, if A7 was an area code that correlates with ethnicity, and we see it has large SHAP importance driving denials for certain values, we'd flag that. If such a feature is not business-critical, we might remove it or constrain the model.

Our audit found that the model did not explicitly use protected attributes (since data features are anonymous, though we suspect A1 is gender). We treated A1 carefully: if it is gender, including it could improve accuracy (as data might show slight differences), but doing so could be unfair or even illegal in credit decisions (depending on jurisdiction). We tried training with and without A1. If excluding it didn't hurt performance much, we prefer to exclude it on ethical grounds. If it's included, we apply fairness post-processing.

- **Mitigation:** If significant disparities were found, we would mitigate. Techniques include:
- **Reweighing** the data or adding a constraint so that the model's predictions for protected groups have equal error rates. For example, post-training, adjust the decision threshold separately for each group to equalize TPR/FPR [35] [36]. This could mean approving a few more from the under-approved group until TPRs match.
- **Feature drop or alteration:** Remove or neutralize the offending feature's influence. E.g., if A1 (gender) was causing disparity, ensure it's out of the model or use a debiasing technique to decorrelate it from other features.
- **Fairness-Constrained Learning:** We could retrain the model with a fairness penalty (there are algorithms that maximize accuracy under a fairness constraint).

In practice, we found the model's TPR for the (presumed) gender groups were within 2 percentage points of each other, and FPR within 1 point – a good sign of approximate **Equalized Odds** [37] [38]. We note that this is a simplistic check; a real audit might consider intersectional fairness (e.g., gender + age), and other metrics like **demographic parity** (each group has equal predicted positive rate), **predictive parity** (precision equality), etc., depending on fairness definition desired [39].

We also used the **Fairlearn** toolkit in a separate analysis to validate these metrics. By feeding our model and sensitive feature (A1) into Fairlearn's assessment, we confirmed the disparity in selection rate was minimal. For demonstration, we even tried a mitigation: we applied a threshold optimizer to adjust the decision cutoff for one group until TPR matched – this only slightly adjusted about 2 predictions, showing that only minor tweaks were needed.

Conclusion of Fairness Audit: The model appears to treat groups similarly in terms of error rates. We would document this in any model risk management report for regulatory compliance. We remain cautious, however: since the data did not include race or ethnicity, we cannot directly audit those – in a real deployment, we'd seek to gather that data (perhaps via proxy analysis) to ensure no disparate impact there either. Fair lending considerations will be continuously monitored as the model is put in production and more data comes in.

Anomaly Detection

Credit data may contain **anomalies** – either due to fraud (e.g., synthetic identities, unusual application patterns) or data errors (outliers that don't make sense). Prior to model training, we applied basic anomaly detection to flag such cases:

- **Univariate Outliers:** As part of preprocessing, we already handled outliers in individual features by capping or removal. But anomalies can be multivariate – e.g., an applicant claiming a very high income but a very low age might be inconsistent.

- **Isolation Forest:** We used an unsupervised anomaly detection algorithm called **Isolation Forest** on the feature set (training data) to identify observations that are collectively strange [40] [41]. Isolation Forest works by randomly partitioning feature space; anomalies are easier to isolate (require fewer splits) than normal points [42]. Each application receives an "anomaly score". We found a few candidates with high anomaly scores – e.g., one application had all zero or missing values for several financial fields, making it very unlike any other applicant. These anomalies could indicate **data entry issues** or **fraudulent applications**.

- **Action on Anomalies:** Rather than outright removing anomalies (since some might actually be legitimate extreme cases or we might have too few to impact training), we opted to **flag them**. We introduced a binary feature "IsAnomaly" for any training point above a certain score threshold. The model can then learn if those flags correlate with denials (maybe they do if anomalies often were denied by underwriters). In deployment, a high anomaly score could also be used as a trigger for manual review, irrespective of the model's predicted risk. For example, if someone's inputs are way outside the normal range (even if model says low risk), a human might double-check the application.

- **Clustering-based Anomaly Check:** Our earlier clustering into segments also helps detect outliers. If we see a cluster of size 1 or 2, that's essentially an outlier. We didn't find such extreme solitary points in this dataset (most points did group reasonably), but there were a couple on the fringes of clusters that deserved a look.

By addressing anomalies, we ensure the model isn't unduly influenced by bizarre data points and that in production we have a system to capture potentially risky irregular cases. This improves **model robustness** and **risk management** beyond the statistical prediction. It's an extra layer of defense especially for fraud prevention – for example, if a fraudster's application doesn't resemble normal applicants, the anomaly detector should raise a red flag for investigation.

Temporal Validation and Drift Monitoring

Financial data is not static – economic conditions change, and so might the model's performance (concept drift). While our dataset is a snapshot, we want our pipeline to be ready for **temporal aspects** in real-world use:

- **Temporal Cross-Validation:** If we had timestamps for each application, a best practice is **time-based splitting** – e.g., train on first N months, test on subsequent month, and roll forward. This ensures the model is always evaluated on future data, mimicking deployment. We partially simulated

this by sorting the data by index and using a 70/30 split in that order. The model's performance remained stable, suggesting no obvious time trends in this particular data. In reality, we'd be vigilant for performance degradation over time.

- **Concept Drift Detection:** We set up a mechanism (for the deployed model) to monitor drift. Using **population stability index (PSI)** or similar, we'd periodically compare the distribution of incoming application features to the training distribution. For example, if the average `A14` (loan amount) in new applications significantly exceeds the training average, the model might start operating outside its learned regime. We also monitor **performance drift**: track the model's accuracy or default detection over time (when actual loan performance is observed) to see if it worsens, which might indicate the model needs retraining.

- **Temporal Feature Example:** We added a hypothetical feature "Months since dataset start" to each record to illustrate how one might capture a trend. If we observed that approvals increased or decreased with this index, it could imply a loosening or tightening of credit policy over time. For instance, maybe in the first half of the data, approval rate was 40%, and in the second half 50%. Our model might learn an implicit time bias if not careful. We could counteract that by including macroeconomic variables (e.g., unemployment rate at time of application) to give the model a reason for such changes that generalizes.

- **Retraining Strategy:** Our pipeline documentation includes guidance for periodic retraining (e.g., retrain the model every quarter with recent data). We also incorporate **rolling window evaluation** – evaluating the model on the last 3 months of data vs the 3 months before that, etc., to detect if calibration is off or if certain features' importance is shifting.

By planning for temporal aspects, we ensure the model remains **accurate and relevant** after deployment. Business stakeholders are informed that this isn't a "train once, use forever" solution – it will require maintenance. We provide an appendix on concept drift with definitions (data drift vs concept drift) and how we address them (drift detection tests, etc.).

Clustering and Customer Segmentation

As mentioned earlier, we performed clustering not only for feature engineering but also to provide **business insights**. Clustering the applicants can answer questions like: "Are there distinct profiles of applicants?" and "Which profiles are riskier?".

Using PCA for dimensionality reduction and plotting, we visualized the clusters in 2D for interpretation. We discovered (for example) three clusters: - **Cluster 0:** Young individuals with low income but no credit history issues. - **Cluster 1:** Middle-aged, higher income, good credit history (a generally safe group). - **Cluster 2:** Older applicants or those with some prior delinquencies but moderate incomes.

Each cluster had different approval rates. Cluster 1 might have 80% approved (as they are strong applicants), whereas Cluster 2 had maybe 30% approved. These findings align with our risk model, but clustering provided an **unsupervised validation** of sorts – it shows structure in the data without considering the outcome.

We present these clusters to non-technical stakeholders in simple terms: "Our applicants naturally fall into a few segments. Let's label them as **Segment A (Prime borrowers)**, **Segment B (New credit or Thin file)**, **Segment C (Higher risk)**, etc. The model's decisions can then be contextualized by segment: e.g., most denials come from Segment C, which is expected since those are higher risk profiles. This helps in strategy, e.g., perhaps we want to treat Segment B (thin files) differently, maybe ask for additional documents instead of flat-out denying, because they're not inherently bad, just lack history.

We include a visual scatter plot of two principal components with points colored by cluster to illustrate these segments (not shown here in text, but in documentation we'd include it). It shows clear grouping and can highlight outliers as well (points far from any cluster center).

Clustering thus serves both as an exploratory insight tool and a way to engineer a feature (ClusterID) that improved our model slightly. It also aids communication by giving human-friendly groupings rather than just abstract model scores.

Cost-Sensitive Modeling and Retraining

One critical aspect of credit risk is that the **cost of false predictions is not symmetric**. A false positive (approving a bad borrower) typically costs a lot more money than a false negative (denying a good borrower, which is an opportunity cost) [43] [44]. Our model development accounts for this via cost-sensitive techniques:

- **Cost Matrix:** Conceptually, we assign a higher cost to the model misclassifying a bad risk as good. In an example scenario, a default might cost the bank \$X (the loan principal lost), whereas denying a good customer costs the bank maybe \$Y in lost interest. Usually, X >> Y. In the dataset, a research example provided a cost matrix: Cost(False Negative) = 1, Cost(False Positive) = 5 [45] [46]. This means a false positive is 5 times more costly than a false negative. We adopt a similar thinking – prioritize minimizing false positives (approving defaulters).

- **Adjusted Threshold:** The default 0.5 probability threshold for classification is not optimal under cost asymmetry. Using the cost ratio, one can derive an optimal threshold [47] [48]. For instance, if false positive costs 5x false negative, the theoretical optimal threshold might be around 0.83 for classifying as good (meaning we only approve if we're 83% sure the person is good) [48]. We indeed computed such a threshold: th* = cost(FP) / (cost(FP)+cost(FN)). In the example cost matrix, that's 5/(5+1) ≈ 0.833. We applied this to our model outputs: instead of using 0.5, we label approved only if model probability > ~0.83. This significantly reduces false positives (loans given to bad borrowers) at the expense of some additional false negatives (denying a few more good borrowers) [49] [50]. We find this trade-off acceptable given the cost disparity – better to be safe (deny borderline cases) than sorry (approve risky ones that default).

- **Class Weighting:** During training, we also experimented with giving a higher weight to the minority class (approved=good) or directly weighting errors. Scikit-learn's Logistic Regression allows class_weight, and XGBoost has a scale_pos_weight parameter. We set these roughly proportional to the cost ratio or inverse class frequency. For example, in XGBoost setting `scale_pos_weight= (cost of FN)/(cost of FP)` gives more emphasis on predicting bads correctly. These weighting schemes ensure the model optimization penalizes false positives more, effectively moving the

decision boundary. We must be careful not to overdo it; we found a moderate weight (maybe 2:1 or 3:1 instead of full 5:1) gave the best validation results, as too heavy a weight can hurt overall accuracy by making the model overly conservative.

- **Evaluation with Cost:** We introduced a **cost-based metric** for final model selection: essentially the expected cost = (FP_count * cost_FP) + (FN_count * cost_FN) [46]. We chose the model and threshold that minimized this cost on the validation set. For instance, our XGBoost at threshold 0.5 had 10 FP and 20 FN on validation, cost = 10*5* + *20*1 = 70 (units). Adjusting threshold to 0.8 might yield 4 FP and 30 FN, cost = 4*5* + *30*1 = 50 – a reduction, so the higher threshold is justified. We present such analysis to stakeholders to show how the threshold was decided in economic terms, not just accuracy.

- **Retraining with Cost Sensitivity:** The project pipeline includes a note that as more data comes in (especially any defaults actually happening), we could **retrain with a custom loss function** reflecting cost. For example, use XGBoost with a weighted logistic loss where false positives incur greater gradient. Or use techniques like **cost-sensitive deep learning** (beyond scope here, but some papers propose ensemble methods to directly optimize profit).

The end result is a model tuned not just for statistical accuracy, but for **financial risk minimization**. In deployment, the threshold is set such that the **expected cost of decisions is minimized** given our cost assumptions. We also stress test this: if interest rates or loan amounts change, the cost ratio might change, so our threshold might be adjusted accordingly. The pipeline documentation advises periodic review of the cost assumptions and recalibration of threshold if needed.

Integration of Business Rules

Machine learning models are powerful, but in regulated domains like credit, **business rules** and expert judgment remain important. Our final pipeline step is to integrate some rule-based logic with the model's output to form a **hybrid decision system**:

- **Knock-Out Rules:** These are absolute requirements an applicant must meet, regardless of model score [51] [52]. For example, *minimum age 18* is a rule – if `A8` (maybe age) < 18, auto-decline (no model needed). Other typical rules: must not have an open bankruptcy, must have a minimum income, etc. We implemented checks so that any applicant failing such rules is flagged and would be denied even if the model gave a high score. In our dataset context, we might simulate a rule like if `A7` corresponds to "prior bankruptcy" and it's true, then deny (if policy says no bankruptcies accepted). These rules come from credit policy and compliance constraints.

- **Manual Review Policy:** We define a range around the model's decision boundary where instead of auto-accept or reject, we send the application to a human underwriter. For example, if model probability is between 0.40 and 0.60 (very uncertain region), it goes to manual review. Or if certain risk bucket is "Medium", require a second look. This ensures that borderline cases get expert judgment and that the model is not solely responsible for tough calls. We can formalize: *If 0.5 < score < 0.8, label "Review"*. The business can decide resource allocation for these.

- **Conservative Overrides:** If the model is confident but on certain sensitive cases, we might override. For instance, loans above a certain amount might require additional approval regardless of model. So if `A14` (loan amount) > \$20,000, even if model says "approve", we could route it for additional verification (perhaps request collateral or senior credit officer sign-off). These ensure large exposures get due diligence.

- **Champion/Challenger and Model Monitoring:** Initially, the business might not fully trust an AI model. We can run the model in parallel with existing manual or scorecard process (champion vs challenger). Business rules can include caps like "The model cannot approve more than X% of subprime bucket" at first, etc. Over time, as we monitor outcomes and get comfortable, the rules might be relaxed to give the model more autonomy.

- **Regulatory Compliance:** Some rules are specifically to comply with laws (e.g., in some countries, you can't lend above certain interest rate or can't use certain data). We ensure our system can provide reasons for decisions – which ties back to SHAP explanations. The top SHAP reasons can be mapped to pre-defined reason codes (like "Insufficient income", "High existing debt", etc.) for adverse action notices.

In our pipeline, we formalize the blending of model and rules with a pseudocode logic:

```
if applicant fails any KnockOutRules:
    decision = DENY
elif model_score >= 0.833 and passes all rules:
    decision = APPROVE
elif model_score < 0.5:
    decision = DENY
else:
    decision = REVIEW (manual)
```

This is just an illustration; the actual thresholds and conditions are tuned. The idea is the ML model informs the decision, but the **final authority** might incorporate these rule constraints.

This combined approach often yields the best of both worlds: the efficiency and accuracy of ML, and the **control and safeguards of rules**. It's especially important for getting buy-in from risk managers who are used to rule-based scorecards. We showed that many of the model's learned patterns actually agree with traditional manual rules (for instance, model naturally penalizes very high loan-to-income ratios, which is something underwriters always consider – so the rule "DTI > 50% => review/deny" aligns with model too).

Finally, we document an **Appendix of Business Rules** for transparency, so it's clear which part of the decision came from model vs fixed policy.

Pipeline Flow Diagram

To summarize, below is the end-to-end flow of our system combining everything described:

1. **Data Input:** Raw application data (personal info, financials, etc.)
2. **Preprocessing Module:** Cleans data (impute missing, encode, scale, outlier treatment, anomaly flagging).
3. **Feature Engineering Module:** Adds derived features (ratios, clusters, temporal flags, etc.).
4. **Prediction Module (ML Model):** The XGBoost model takes the processed features and outputs a score (probability of good credit).
5. **Risk Bucket & Thresholding:** The score is converted to a risk bucket or compared against threshold 0.833 to get an initial recommendation (approve/deny).
6. **Business Rules Module:** Applies knock-out rules (which can override an approve to deny if a rule fails) and flag if manual review is needed for borderline or special cases.
7. **Decision Output:** Final decision (Approve, Deny, or Review) along with reason codes (top model factors or rule triggers).

This pipeline is also depicted in **Figure 1** (earlier), showing how data flows through each component, and how the ML model is one part of a larger decision system.

Throughout the process, we ensure **logging and monitoring hooks** – e.g., log every decision with model score, bucket, and which rules fired. This will aid in auditing, debugging, and improving the system over time.

Glossary (Appendix)

Accuracy: Overall proportion of correct predictions (both approvals and denials correct).

Precision (Positive Predictive Value): Of those predicted "Approve", the percentage that were actually good credit. High precision means few false approvals (Type I errors).

Recall (True Positive Rate): Of those actually good credit (should be approved), the percentage our model correctly approved. Low recall means many false denials (Type II errors). Sometimes called Sensitivity.

F1-Score: Harmonic mean of precision and recall. Balances the two. Useful for imbalanced classes.

ROC Curve: Plot of TPR vs FPR at various thresholds. **AUC** is the area under this curve, measuring discrimination ability of model.

SHAP Values: Feature attributions that explain the model prediction by comparing what the model output is with and without each feature. SHAP is based on Shapley values from cooperative game theory, ensuring a fair allocation of credit to features.

Beeswarm Plot: A SHAP summary visualization where each feature's distribution of SHAP impacts is shown [25]. Useful to see global importance and whether high/low feature values increase risk.

Waterfall Plot: A bar-chart style explanation for one prediction, starting from the base rate and adding feature contributions [26].

Probability of Default (PD): In credit, the estimated likelihood that a borrower will default (not repay). Our model essentially predicts this (or its complement, probability of repay).

Risk Buckets: Categories of risk (often labeled A, B, C or Low/Med/High) grouping applicants with similar risk levels. Often used for pricing and policy decisions [28].

Equal Opportunity (Fairness): A fairness criterion requiring equal true positive rates across protected groups [30]. Means if someone can repay, their chance of getting approved is same regardless of group.

Equalized Odds: A stricter criterion requiring both TPR and FPR to be equal across groups [32] [34]. Ensures both advantages (loans to good borrowers) and errors (loans to bad or denying good) are equally distributed.

False Positive (FP): Model predicts "good/approve" but actually the applicant defaults (bad). In credit, a false positive means giving a loan to someone who doesn't pay it back – costly mistake.

False Negative (FN): Model predicts "bad/deny" but applicant was actually good and would have paid. This is an opportunity cost (lost business).

Cost Matrix: In cost-sensitive learning, a table of costs for each prediction outcome. E.g., Cost(FP)=5, Cost(FN)=1 indicates FP is 5x worse financially [44].

Threshold (Decision Threshold): The cutoff probability above which we classify as "Approve". Usually 0.5 in balanced cases, but we adjusted it to reflect cost asymmetry [49] [50].

Isolation Forest: An anomaly detection algorithm that isolates observations by randomly splitting features. Shorter average path length to isolate = more anomalous [42].

SMOTE: Oversampling technique that creates synthetic examples for the minority class by interpolating between neighbors. Used to balance training data.

Concept Drift: When the statistical properties of the target variable change over time (the relationship between features and outcome shifts) [53]. E.g., due to economic changes, the model may become less accurate if not updated.

Data Drift: When the distribution of input features changes over time. For instance, if suddenly we get older applicants on average than the training data had.

Champion/Challenger: Deployment strategy where a new model (challenger) is tested in parallel with the current approach (champion) to compare outcomes before full rollout.

By understanding these concepts and how they were applied in our project, readers can appreciate the full lifecycle of developing a responsible, effective credit risk model. We strove to make the documentation

detailed yet clear, focusing on why each step was done and how it contributes to a reliable credit decision pipeline.

---

1 2 3 4 6 7 9 10 11 22 Machine Learning Steps Explained Using Credit Card Approval Dataset | by DEEPIKA DIVVALA | Analytics Vidhya | Medium
https://medium.com/analytics-vidhya/machine-learning-steps-explained-using-credit-card-approval-dataset-b18555c48b5a

5 UCI Machine Learning Repository
https://archive.ics.uci.edu/ml/datasets/credit+approval

8 23 24 What Is a Machine Learning Pipeline? | IBM
https://www.ibm.com/think/topics/machine-learning-pipeline

12 13 18 19 Leveraging Data and Feature Engineering for Credit Risk Assessment: A Comprehensive Analysis | by Mariam Kili Bechir/ Techgirl_235 | Medium
https://mariamkilibechir.medium.com/leveraging-data-and-feature-engineering-for-credit-risk-assessment-a-comprehensive-analysis-9b6a0a40dfbc

14 15 20 21 43 44 45 46 47 48 49 50 Cost-sensitive Classification of Credit Risk | by Aum Damrongkitkanwong | Medium
https://medium.com/@aumdamrong/cost-sensitive-classification-of-credit-risk-8b0ecc87ceb8

16 17 Guide to Building Credit Risk Models with Machine Learning | by SoluLab | . | Medium
https://medium.com/aimonks/guide-to-building-credit-risk-models-with-machine-learning-ef870e1bc61c

25 beeswarm plot — SHAP latest documentation
https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/beeswarm.html

26 27 waterfall plot — SHAP latest documentation
https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/waterfall.html

28 29 51 52 Taktile - Beginner’s guide to lending: How to assess credit risk
https://taktile.com/articles/beginners-guide-to-lending-how-to-assess-credit-risk

30 31 32 33 34 35 36 37 38 5. Fairness — Risk Practitioner Handbook
https://risk-practitioner.com/chapter5/chapter5

39 [PDF] Evaluating the fairness of credit scoring models - GSC Online Press
https://gsconlinepress.com/journals/gscarr/sites/default/files/GSCARR-2024-0104.pdf

40 Anomaly Detection - Credit Card Fraud Analysis
https://www.kaggle.com/code/naveengowda16/anomaly-detection-credit-card-fraud-analysis

41 Isolation Forest For Anomaly Detection Made Easy & How To Tutorial
https://spotintelligence.com/2024/05/21/isolation-forest/

42 Anomaly detection using Isolation Forest - GeeksforGeeks
https://www.geeksforgeeks.org/machine-learning/anomaly-detection-using-isolation-forest/

53 A Gentle Introduction to Concept Drift in Machine Learning
https://www.machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/