# Task 2 — Data Analytics Report

## Research Question

I aim to determine whether recent repayment status, bill amounts, and payment amounts can estimate the likelihood that a cardholder will default next month. This addresses a practical need: when customers are at risk of falling behind, losses can escalate quickly and operations teams have limited capacity. A clear, short term, explainable risk score helps focus outreach on the right customers, offer payment plans earlier, and make timely credit line decisions. Without such a signal, teams either miss likely defaulters and incur higher losses, or spend time on low risk accounts and reduce efficiency and customer satisfaction.

I selected a one month horizon because it is directly actionable. A near term score can drive next day call lists and outreach while remaining easier to explain and govern since the inputs are familiar, including recent delinquencies, bills, and payments.

My null hypothesis is that these behaviors do not outperform chance and produce an AUC equal to 0.50. My alternative hypothesis is that at least one of these behaviors is predictive and the model achieves an AUC greater than 0.50. Given the dataset includes clear delinquency codes such as PAY\_0 through PAY\_6 and detailed payment patterns, the alternative hypothesis is reasonable.

## Data Collection

I downloaded the data as a public CSV from the UCI Machine Learning Repository (Yeh, 2016). The dataset is de‑identified and is provided for educational use. The main advantage of this approach is transparency. Anyone can download the same file, which supports reproducibility and version control in GitHub. The main disadvantage is that the dataset comes from one geography and one time period, so results may not generalize to all portfolios or economic conditions.

I encountered two minor issues. First, the label column name varies across copies of the dataset. I resolved this by standardizing names in a dbt staging model and by detecting the label case insensitivity in the analysis script. Second, the CSV includes a header row that must be skipped. I handled this by defining a Snowflake file format that skips the header during the COPY step.

In Snowflake I use the feature view CREDIT\_DEFAULT.MODEL.FEATURE\_VIEW\_CREDIT\_DEFAULT. The table contains thirty thousand rows and twenty‑four columns. The label column is TARGET. The class distribution is imbalanced with 23,364 non defaults and 6,636 defaults.

## Data Extraction and Preparation

I built a small pipeline to keep the data engineering simple and auditable. I created an extra‑small warehouse and a CREDIT\_DEFAULT database in Snowflake. I set up a CSV file format and a named stage, then I used COPY INTO to load the raw table. In dbt I created a staging model that typed and renamed columns and dropped the ID field, and I created a feature view that selects the fields used for modeling. I added dbt tests for null checks and accepted values. I configured the project to use GitHub Actions so that each push runs dbt build and then runs the Python analysis.

I selected Snowflake and dbt because they allow me to define transformations declaratively and to test them automatically at low cost. The advantage of this approach is that the pipeline is reproducible, versioned, and easy to inspect. The main limitation is that I intentionally scoped the pipeline to be lightweight, so I did not add complex joins or entity resolution logic. I include screenshots that show warehouse creation, staging and COPY steps, raw row counts, the dbt build in CI, and a preview of the feature view.

## C.1 Selected Data Engineering Screenshots

Screenshots that show warehouse and database setup, uploading the CSV to a stage, the COPY step, and raw row counts.

A screenshot of a computer

AI-generated content may be incorrect.

Warehouse + database setup (XS warehouse with auto‑suspend).

A screenshot of a computer

AI-generated content may be incorrect.

Upload CSV to stage (INGESTION) before load.

A screenshot of a computer

AI-generated content may be incorrect.

COPY INTO RAW table (RAW\_CREDIT\_DEFAULT\_RAW).

A screenshot of a computer

AI-generated content may be incorrect.

Row count validation in RAW (30,000 rows).

A screenshot of a computer

AI-generated content may be incorrect.

Table preview of RAW (ID column is present)

## Analysis Discussion

I interpret the model metrics with business decisions in mind. The AUC measures ranking quality across all possible thresholds and is a standard way to compare binary classifiers (Fawcett, 2006). The XGBoost model reaches an AUC of about 0.78, which means that if I randomly select one defaulter and one non defaulter, the model ranks the defaulter higher about seventy‑eight percent of the time. Logistic Regression reaches an AUC of about 0.71. Accuracy depends on a cutoff. Because the classes are imbalanced, I also track precision and recall. Precision tells me, out of the accounts I flag, how many truly default. Recall tells me, out of all defaulters, how many I catch. I then set the threshold to fit capacity: a lower threshold catches more risk but triggers more extra reviews, while a higher threshold sends fewer alerts with cleaner review lists.

I chose a policy threshold of 0.35 for XGBoost. At this cutoff the recall is about 0.80 and the precision is about 0.34. I prefer high recall for an early‑warning use case because missing likely defaulters is costly. If operations need to reduce false positives, I can raise the cutoff toward 0.45 or 0.50, which increases precision and reduces recall. This gives operations a simple lever to tune workload without retraining the model.

I use two models for complementary reasons. Logistic Regression is transparent. Its coefficients map directly to odds ratios so I can explain how each variable moves risk. For example, recent delinquency indicators such as PAY\_0 through PAY\_6 increase risk, while a higher credit limit tends to lower risk. XGBoost is non‑linear and captures interactions that a linear model might miss. It is a scalable gradient boosting algorithm that handles tabular, non-linear relationships efficiently (Chen & Guestrin, 2016). I report feature importances and I can add SHAP values if I need case‑level explanations.

I used a stratified train–test split so that the proportion of defaults is preserved in both sets, which is important because the classes are imbalanced. I trained two models that are appropriate for a binary, tabular problem: Logistic Regression as an interpretable baseline and XGBoost as a non-linear benchmark. For evaluation, I computed ROC AUC, accuracy at a 0.50 cutoff, confusion matrices, and a threshold sweep to support policy selection.

To address class imbalance, I set class\_weight='balanced' for Logistic Regression and scale\_pos\_weight for XGBoost (ratio of negatives to positives in the training set). I fixed a random seed for reproducibility and kept hyperparameters modest to stay within the project’s cost and complexity constraints.

**Calculations and outputs**. Logistic Regression achieved AUC ≈ 0.71 with accuracy ≈ 0.69 at the 0.50 cutoff. XGBoost achieved AUC ≈ 0.78 with accuracy ≈ 0.76 at the 0.50 cutoff. At my selected policy cutoff of 0.35 for XGBoost, precision ≈ 0.339, recall ≈ 0.804, and accuracy ≈ 0.610. The corresponding confusion matrix is TN = 3,889, FP = 3,120, FN = 391, TP = 1,600. These results show that the non-linear model improves ranking power, while the lower threshold recovers more true defaulters for early outreach.

Why these techniques. Logistic Regression is a natural choice for a binary outcome and provides coefficients and odds ratios that I can explain to stakeholders. Advantage: it is transparent and easy to calibrate. Disadvantage: it may underfit when relationships are non-linear or involve interactions. XGBoost is well suited for tabular data with potential interactions and non-linearities. Advantage: it delivers higher AUC and better ranking on this dataset with modest compute. Disadvantage: it is less transparent, which I mitigate by reporting feature importances now and planning SHAP for case level explanations later.

Together, these choices are appropriate for the data and the business need: the baseline supports governance and explanation, and the benchmark improves detection performance.

## D.2 Selected Analysis/CI Screenshots

I include screenshots that show target distribution, the dbt build in CI, preview and counts of the feature view, the model‑metrics summary in GitHub Actions, the use of repository secrets for secure Snowflake access, confusion matrix @ XGBoost 0.35 threshold policy, and AUC ROC of the test set. Lastly, I included screenshots of github with passing pipeline badge.

A screenshot of a computer

AI-generated content may be incorrect.

Target distribution (class balance) used for stratified splits and threshold tuning.

A screenshot of a computer program

AI-generated content may be incorrect.

dbt build in the CI pipeline creating the staging and feature view in Snowflake.

A screen shot of a computer

AI-generated content may be incorrect.

Models.yml file Declarative Tests

A screenshot of a computer

AI-generated content may be incorrect.

Preview of records in the analytics feature view (post‑staging / ID dropped).

A screenshot of a computer

AI-generated content may be incorrect.

Count of records in the analytics feature view (post‑staging / ID dropped).

A screenshot of a computer

AI-generated content may be incorrect.

Model Metrics build-and-analyze summary in Github Actions

A chart with different colored squares

AI-generated content may be incorrect.

Confusion Matrix with XGBoost @ 0.35 policy threshold

A graph of a logistic curve

AI-generated content may be incorrect.

ROC AUC Curve

A screenshot of a computer

AI-generated content may be incorrect.

Github repository with passing pipeline badge as of 10/6/2025

A screenshot of a computer

AI-generated content may be incorrect.

Github Actions tab with passing workflows

## Outcomes and Implications

My research question asked whether recent repayment status, bill amounts, and payment amounts can estimate a customer’s probability of default next month. The analysis shows that they can. The XGBoost model achieves an AUC ≈ 0.78, meaning that if I randomly select one defaulter and one non defaulter, the model ranks the defaulter higher about 78% of the time. In credit risk applications this is a solid level of discrimination, better than the interpretable Logistic Regression baseline (AUC ≈ 0.71) and well within the range used by risk teams for case prioritization and outreach planning. Extremely high AUCs (for example, >0.90) are uncommon for this kind of problem and can indicate data leakage; therefore, 0.78 is a realistic and trustworthy result. This falls in line with large-scale benchmarking in credit scoring, where many real-world models achieve AUCs in the ~0.70–0.85 range and tree-based ensembles often lead performance (Lessmann, Baesens, Seow, & Thomas, 2015

At a policy threshold of 0.35, XGBoost delivers high recall (it finds most likely defaulters) while keeping precision at a workable level for operations. This matches the business need for early warning lists where missing a likely defaulter is costlier than reviewing extra accounts. If staffing or capacity tighten, I can raise the threshold (e.g., to 0.45–0.50) to increase precision and reduce caseload without retraining the model.

Recommended action: I will adopt XGBoost with a 0.35 cutoff to power daily outreach lists and to flag accounts for quick credit line review. I will publish a simple playbook: start at 0.35 for high recall; adjust upward if precision needs to increase. I will also report threshold based metrics (precision, recall, confusion matrix) alongside AUC so that stakeholders see both ranking quality and operational trade offs.

Limitation: The dataset reflects one market and one time period and includes demographic fields. That limits generalizability and requires fairness and policy review before any production use.

**Three directions for future study:**

**Add better signals (and more markets):** I will build richer features—trends, spikes, rolling averages, utilization bands—and, where allowed, bring in similar data from other regions. I’ll standardize the fields in dbt and normalize amounts so currency and scale don’t skew results. Success looks like a clear lift in performance (e.g., +0.02–0.04 AUC or +5–10 points recall at the same precision).

**Score by time horizon (1, 2, 3 months):** I will produce separate risk scores for near term windows. One month risk feeds immediate outreach; two month risk supports reminders or payment plan offers; three month risk drives education and credit health nudges. The goal is the right action at the right time.

**Tune the model for impact:** I will run a structured hyperparameter tune on XGBoost (depth, trees, learning rate, subsample, etc.) and sanity check results on an out-of-time test. I will keep costs low (XS warehouse or free Kaggle resources) and log the chosen settings and metrics in CI so we can show a before/after impact.

These actions and next steps follow directly from the results and can be carried out with the data and pipeline I have in place. And they will improve both decision making and communication around model outputs.

## References

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’16), 785–794. https://doi.org/10.1145/2939672.2939785

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861–874. <https://doi.org/10.1016/j.patrec.2005.10.010>

Lessmann, S., Baesens, B., Seow, H.-V., & Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. European Journal of Operational Research, 247(1), 124–136. https://doi.org/10.1016/j.ejor.2015.05.030

Yeh, I.-C. (2016). Default of Credit Card Clients [Data set]. UCI Machine Learning Repository. [10.24432/C55S3H](https://doi.org/10.24432/C55S3H)