Final Report (Tasks 1–6)

Title: Credit Risk Probability Model for Alternative Data — An End-to-End BNPL Risk System
Context: Bati Bank + eCommerce partner (Buy-Now-Pay-Later eligibility and terms)
Goal: Convert transaction behavior into a **risk probability**, a **credit score**, and deployment-ready **risk service**.

1) Business Problem and Why This Matters

Buy-Now-Pay-Later (BNPL) requires the bank to decide whether to extend credit and under what terms. Traditional credit risk models rely on historical repayment/default outcomes. In this project, we start with **alternative behavioral data (transactions)** and build a pipeline that can:

- define a proxy for risk (since we lack observed defaults),
- engineer customer-level features,
- train and evaluate models with tracking and reproducibility,
- deploy the best model behind an API with CI/CD safeguards.

2) Basel II Context and Model Governance

Basel II's emphasis on risk measurement and supervisory review makes **interpretability, documentation, validation, and monitoring** essential. Even if a complex model performs better, we must be able to explain decisions, track experiments, and ensure the system behaves consistently.

Deliverable: `README.md` contains a “Credit Scoring Business Understanding” section covering these trade-offs and proxy-label risks.

3) Data Overview (Xente Transactions)

Dataset summary (raw transactions):
- **95,662 rows × 16 columns**
- **No missing values** and **no duplicates**
- Date range: **2018-11-15** to **2019-02-13** (UTC)
- `CurrencyCode` is constant (**UGX**) and `CountryCode` is constant (**256**)

Key numeric behavior:
- `Amount` includes negatives (credits) and positives (debits), with large outliers
- `Value` is absolute magnitude; **Corr(Amount, Value) ≈ 0.99**

4) Feature Engineering (Task 3)

Because the system must score a **customer**, we convert transaction-level data into a **customer-level feature table** (1 row per `CustomerId`).

Implemented in: `src/data_processing.py`

4.1 Aggregate features per customer
- `TotalTransactionAmount`: sum(Amount)
- `AverageTransactionAmount`: mean(Amount)
- `TransactionCount`: count(TransactionId)
- `StdTransactionAmount`: std(Amount) (filled with 0 when only one transaction exists)

4.2 Time-derived features
From `TransactionStartTime` we derive hour/day/month/year and aggregate them per customer (mean/mode).

4.3 Categorical encoding + numeric scaling (sklearn Pipeline)
- Categorical (mode per customer) → `OneHotEncoder(handle_unknown="ignore")`

- Numeric → median imputation + `StandardScaler`

Runner scripts:
- `python scripts/run_task3.py` → writes `data/processed/processed.csv`

5) Proxy Target Engineering (Task 4 — RFM + Clustering)

We do not have real loan defaults, so we build a **proxy** outcome `is_high_risk`.

Implemented in: `src/target_engineering.py`

5.1 Compute RFM per customer
- **Recency**: days since last transaction (higher = more disengaged)
- **Frequency**: number of transactions (lower = more disengaged)
- **Monetary**: sum(Value) (lower = more disengaged)

5.2 KMeans clustering (k=3, reproducible)
- Scale RFM with `StandardScaler`
- Cluster into 3 segments using `KMeans(n_clusters=3, random_state=42)`
- Select "least engaged cluster" using a combined risk score:
- risk = z(Recency) − z(Frequency) − z(Monetary)

Label:
- Cluster with highest mean risk score → `is_high_risk = 1`
- Others → `is_high_risk = 0`

Runner:
- `python scripts/run_task4.py` → writes `data/processed/processed_with_target.csv`

6) Model Training, Evaluation, and MLflow Tracking (Task 5)

Implemented in: `src/train.py`

6.1 Data splitting
- `train_test_split(test_size=0.2, random_state=42, stratify=y)` for reproducibility and stable class proportions.

6.2 Two model families (with hyperparameter tuning)
We train at least two models and tune with `GridSearchCV(cv=3, scoring="roc_auc")`:

- Logistic Regression (baseline / interpretable)
- Random Forest (non-linear baseline)

6.3 Metrics logged
We compute and log:
- Accuracy, Precision, Recall, F1, ROC-AUC
and log artifacts such as a **confusion matrix** image.

6.4 MLflow experiment tracking + model registry
- Runs are logged under experiment `credit-risk-task5`
- Best model (by ROC-AUC) is logged and registered as `credit_risk_model`

Runner:
- `python scripts/run_task5.py`

7) Deployment + CI/CD (Task 6)

7.1 FastAPI service

Files:
- `src/api/main.py`
- `src/api/pydantic_models.py`

Endpoints:
- `GET /health`: verifies model loading
- `POST /predict`: returns `risk_probabilities`

Model loading is through MLflow:
- `MLFLOW_TRACKING_URI` (default `sqlite:///mlflow.db`)
- `MODEL_URI` (default `models:/credit_risk_model/1`)

7.2 Docker
Files:
- `Dockerfile`
- `docker-compose.yml`

Start API with:
- `docker compose up --build`

7.3 GitHub Actions CI
Workflow:
- `.github/workflows/ci.yml`

On each push to `main`, CI runs:
- `flake8`
- `pytest`

8) Limitations and Risk Notes (Proxy-Based Modeling)

This system predicts risk using a **proxy** (RFM disengagement), not real default outcomes. That introduces:
- proxy-label risk (misalignment with true repayment behavior),
- concept drift risk (customer behavior changes),
- potential bias (some segments transact differently).

Mitigation:
- monitor post-deployment performance,
- update proxy definition as real repayment data becomes available,
- retrain and validate regularly, maintain documentation.

9) How to Reproduce End-to-End

From repo root:

1. Build customer-level features:
- `python scripts/run_task3.py`
2. Create proxy target:
- `python scripts/run_task4.py`
3. Train + log models:
- `python scripts/run_task5.py`
4. Run API:
- `python -m uvicorn src.api.main:app --host 0.0.0.0 --port 8000`

10) Appendix: Artifacts to Attach (Screenshots)

For a complete Medium-style final submission, add screenshots of:
- MLflow experiment list and best run metrics

- CI workflow passing (GitHub Actions)
- Docker container running + sample `/predict` request/response