Credit Risk Prediction Using the German Credit Dataset

CS 4120 | Machine Learning, Data Mining

Midpoint Submission

Ditthi Chatterjee & Mohammed Sahm

1. Updated Dataset Description and Cleaning Notes

The dataset used in this project focuses on credit risk prediction and contains both classification and regression targets, allowing the exploration of supervised learning methods across two predictive tasks.

Data Cleaning: The data had no missing values or duplicate entrees and needed no type conversions.

Preprocessing notes

- **Categorical encoding:** All categorical variables were encoded using a combination of Label Encoding and One-Hot Encoding.
- **Feature Scaling:** All numerical values were scaled using Standard Scaler.
- **Imbalance handling:** The original class distribution was approximately 70 - 30; this was adjusted to a balanced 50 - 50 distribution through resampling to improve classification performance.

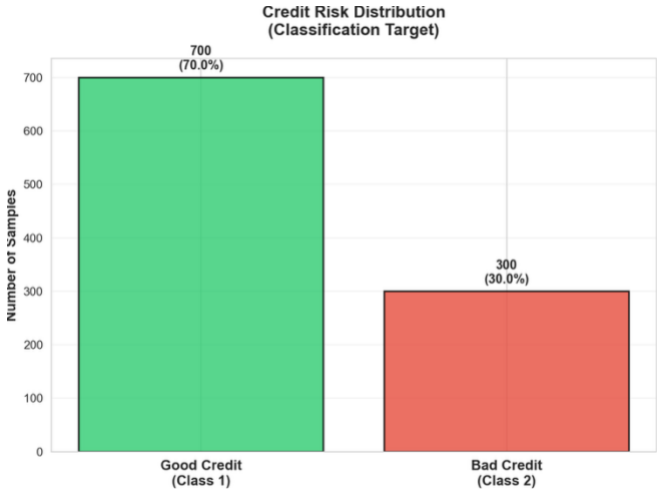
→ Data loading and splitting were handled through *data.py*

---

2. Exploratory Data Analysis (EDA)

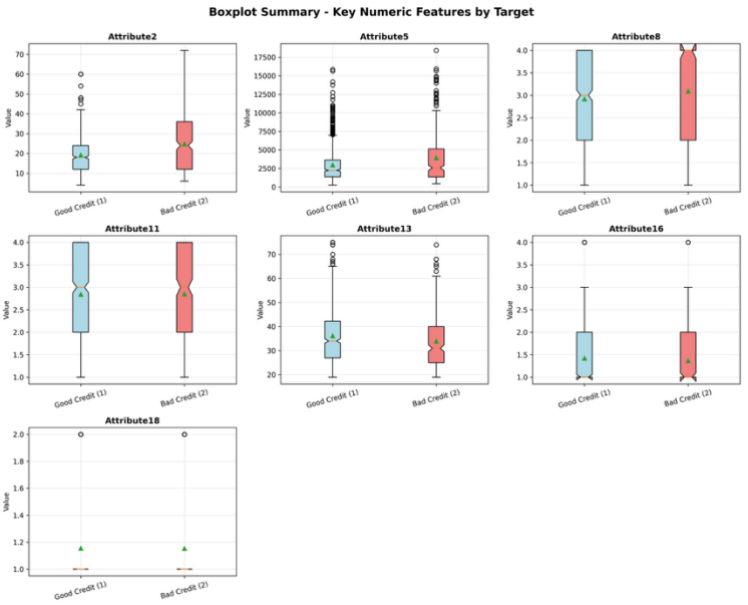
An initial EDA was performed on the raw dataset to understand its structure, distribution, and relationships between features and target variables. The initial EDA and Plot 1 is given by *notebooks/01_basic_eda.ipynb.*

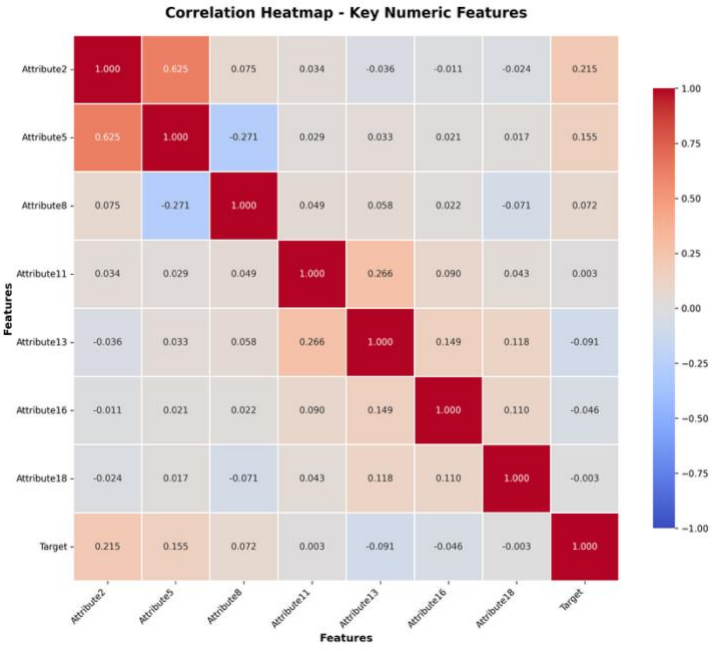
- **Plot 1:** Target distribution for the classification task (showing class counts before and after balancing).



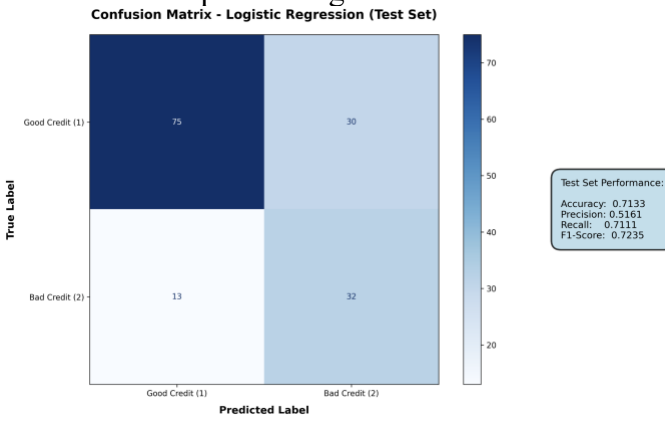
The following visualizations were generated via *evaluate.py*:

- **Plot 2A / 2B:** Correlation heatmap and boxplot summaries of key numerical features to highlight potential relationships.

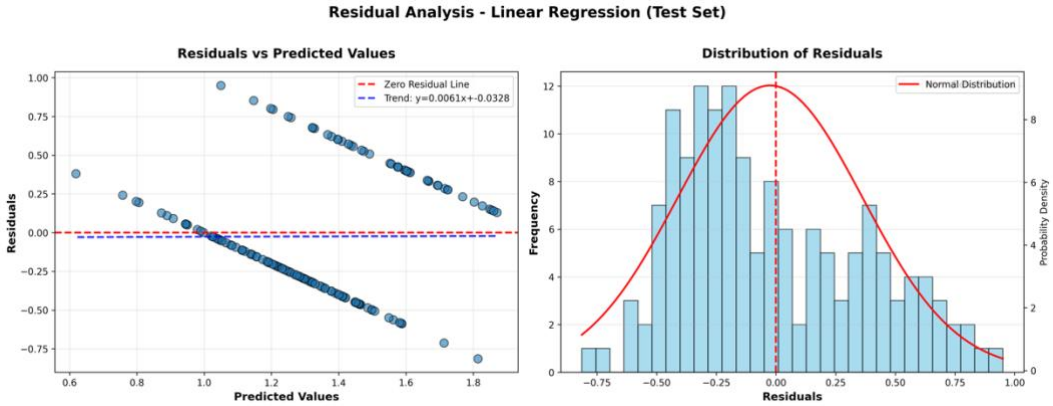




- **Plot 3:** Confusion matrix for the best-performing classification baseline.



- **Plot 4:** Residuals vs. Predicted plot for the best-performing regression baseline.



3. Baseline Models and MLflow Tracking

Baseline models were trained for both tasks, and all runs were tracked using **MLflow** to record parameters, metrics, and artifacts.

Classification Baselines

Classical machine learning models implemented:

1. Logistic Regression
2. Decision Tree Classifier

Regression Baselines

Models used for the regression task:

1. Linear Regression
2. Decision Tree Regressor

Hyperparameter Tuning

Each baseline was trained with default hyperparameters initially, followed by tuning to optimize performance.
[We used tune_decision_tree_regressor.py, tune_linear_regression.py, tune_hyperparameters.py, tune_logistic_regression.py to separately tune hyperparameter values, and then implemented best values into train_baselines.py. These files were used individually for each model implemented, with varying algorithms used to find best values for optimized accuracy]

MLflow Implementation

Each experiment (data split, model training, and evaluation) was logged to MLflow, including model type, configuration, and evaluation metrics.

---

4. Train – Validation - Test Split

A fixed **random seed (42)** was used to ensure reproducibility. The data was split in the following proportions:

Training: [60 %] **Validation:** [20 %] **Test:** [20 %]

5. Results and Discussion

The results of the baseline models are summarized in Tables 1 and 2.

Table 1 – Classification Metrics

Model	Val_Accuracy	Val_F1-Score	Val_ROC-AUC	Test_Accuracy	Test_F1-Score	Test_ROC-AUC
Logistic Regression	0.7133333333333334	0.7210209463157371	0.7837037037037037	0.7133333333333334	0.7234807031136506	0.7790476190476191
Decision Tree Classifier	0.7333333333333333	0.7333333333333333	0.7540740740740741	0.6466666666666666	0.6591738898842672	0.7085714285714286

Table 2 – Regression Metrics

Model	Val_MAE	Val_RMSE	Test_MAE	Test_RMSE
Linear Regression	0.3464930110360525	0.41771899579711197	0.33134529895350406	0.3879268526574735
Decision Tree Regressor	0.34417173940898094	0.4347295773315977	0.3393377867733138	0.4144498010984179

Discussion

Preliminary observations:

- Balancing the dataset improved the classification model's ability to detect minority cases. (class weight)
- Encoding method changes affected performance consistency across runs. (categorial features)
- Hyperparameter tuning led to modest improvements in both accuracy and F1 for classification, and lower RMSE for regression.
- Some failure modes were observed in the Decision Tree Regressor where the model overfit.
- Next steps will involve testing neural networks to handle more complex nonlinear relationships.

---

6. Neural Network Plan

For the upcoming phase, we plan to implement:

- **Classification Task:** A Multilayer Perceptron (MLP) architecture suitable for tabular data.
- **Regression Task:** A parallel MLP model configured for continuous value prediction.

Justification

Since the dataset is primarily tabular and numerical, an MLP is appropriate. It can model non-linear dependencies more effectively than linear or tree-based methods. We aim to compare its performance against classical baselines and analyze generalization improvements.

Next steps will include:

- Normalizing inputs for the NN.
- Experimenting with hidden layer depth and activation functions.
- Logging results and metrics using MLflow for consistency.

---

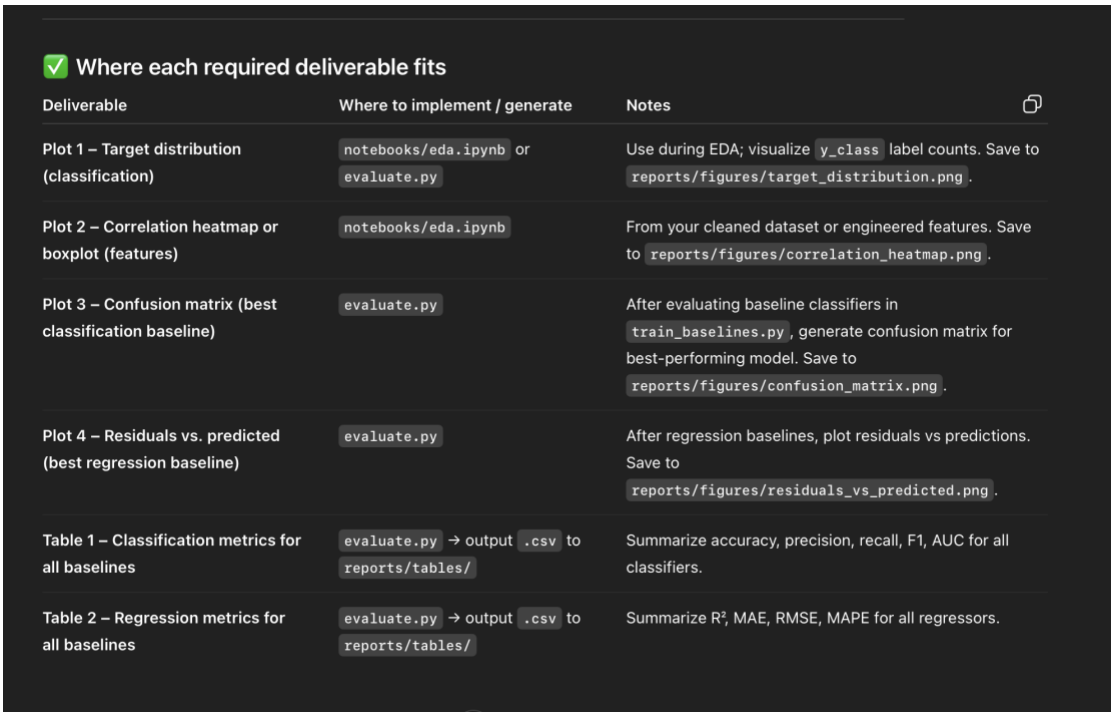
7. AI Disclosure

AI tools (ChatGPT) were used for:

- Writing and tuning code for hyperparameter experiments. (helped program and debug tuning files)
- Creating visualization code for EDA (Seaborn, Matplotlib).
- Debugging logic flow by suggesting where to add print statements.
- Implementing MLflow tracking setup.
- Learning and general question and answer
- Helping structure the repository and the overall program flow.

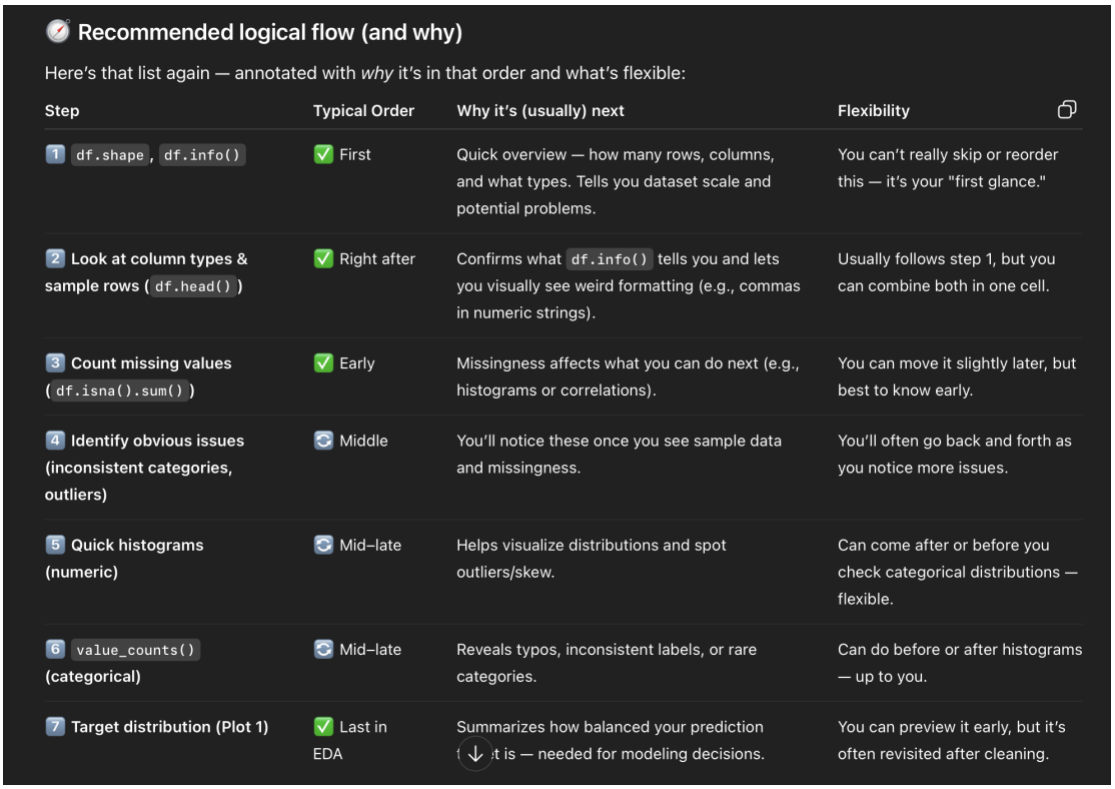
All model designs, dataset processing, and evaluation logic were independently implemented and verified by the group, with the addition of code and processing ideas inspired from Chat GPT.

Some responses provided by ChatGPT that assisted us in the project:



✅ Where each required deliverable fits

Deliverable	Where to implement / generate	Notes
Plot 1 – Target distribution (classification)	`notebooks/eda.ipynb` or `evaluate.py`	Use during EDA; visualize `y_class` label counts. Save to `reports/figures/target_distribution.png`.
Plot 2 – Correlation heatmap or boxplot (features)	`notebooks/eda.ipynb`	From your cleaned dataset or engineered features. Save to `reports/figures/correlation_heatmap.png`.
Plot 3 – Confusion matrix (best classification baseline)	`evaluate.py`	After evaluating baseline classifiers in `train_baselines.py`, generate confusion matrix for best-performing model. Save to `reports/figures/confusion_matrix.png`.
Plot 4 – Residuals vs. predicted (best regression baseline)	`evaluate.py`	After regression baselines, plot residuals vs predictions. Save to `reports/figures/residuals_vs_predicted.png`.
Table 1 – Classification metrics for all baselines	`evaluate.py` → output `.csv` to `reports/tables/`	Summarize accuracy, precision, recall, F1, AUC for all classifiers.
Table 2 – Regression metrics for all baselines	`evaluate.py` → output `.csv` to `reports/tables/`	Summarize R^2, MAE, RMSE, MAPE for all regressors.



🧭 Recommended logical flow (and why)

Here's that list again — annotated with *why* it's in that order and what's flexible:

Step	Typical Order	Why it's (usually) next	Flexibility
1 `df.shape`, `df.info()`	✅ First	Quick overview — how many rows, columns, and what types. Tells you dataset scale and potential problems.	You can't really skip or reorder this — it's your "first glance."
2 Look at column types & sample rows (`df.head()`)	✅ Right after	Confirms what `df.info()` tells you and lets you visually see weird formatting (e.g., commas in numeric strings).	Usually follows step 1, but you can combine both in one cell.
3 Count missing values (`df.isna().sum()`)	✅ Early	Missingness affects what you can do next (e.g., histograms or correlations).	You can move it slightly later, but best to know early.
4 Identify obvious issues (inconsistent categories, outliers)	🔄 Middle	You'll notice these once you see sample data and missingness.	You'll often go back and forth as you notice more issues.
5 Quick histograms (numeric)	🔄 Mid–late	Helps visualize distributions and spot outliers/skew.	Can come after or before you check categorical distributions — flexible.
6 `value_counts()` (categorical)	🔄 Mid–late	Reveals typos, inconsistent labels, or rare categories.	Can do before or after histograms — up to you.
7 Target distribution (Plot 1)	✅ Last in EDA	Summarizes how balanced your prediction t…t is — needed for modeling decisions.	You can preview it early, but it's often revisited after cleaning.