Advanced Data Mining and Predictive Analytics

1. Project Goal

The goal of this project is to apply machine learning techniques to evaluate credit risk in a more sophisticated and financially meaningful way. Specifically, the project aims to move beyond traditional binary underwriting decisions by building models that quantify both the likelihood of loan default and the magnitude of potential losses. By doing so, the project seeks to demonstrate how modern machine learning can support smarter, data-driven decision-making within the financial ecosystem enhancing risk assessment, improving capital allocation, and aligning credit evaluation with broader asset-management strategies focused on optimizing risk-adjusted performance, to real-world data and demonstrate understanding of machine learning concepts covered in Modules 6 and 7, using the Carseats dataset to predict Sales.

2. Objectives

The objective of this project is to develop a machine learning model that can both predict whether a loan will default and estimate the financial loss associated with a default. Rather than relying on traditional finance methods that classify borrowers simply as “good” or “bad,” this project adopts a more comprehensive approach by modeling both the probability of default and the severity of loss.

This focus allows us to capture the full risk profile of a loan, bridging the gap between traditional banking goals such as minimizing economic capital consumption and the asset-management perspective, which emphasizes optimizing risk and return for investors. By integrating both default prediction and loss estimation, the project aligns with modern financial practices where machine learning enhances decision-making across underwriting, lending, and portfolio risk management.

3. Data Overview and Data exploration

The training dataset (train_v3) consists of 80,000 rows and 763 columns, while the test set contains 25,471 rows and 762 columns. All fields are numeric, but values vary dramatically in scale, some are extremely large, while others are relatively small. Because the dataset lacks documentation and all column headers follow a generic naming pattern (e.g., f1, f2, f3), we do not have subject-matter expertise to interpret the features. This limits feature engineering and requires a more cautious, data-driven cleaning strategy.

Headers and Feature Information

- All column headers are meaningless; we don't have a subject-matter expert (SME) to clarify them. Some fields appear skewed, but this cannot be confirmed. - Feature naming convention:
    - ○ Features: start with f
    - ○ Id: loan identifiers
    - ○ Loss: loan loss amount

Formatting

- Inspecting the first few rows shows that while all values appear numeric, the formatting varies: integers, floats, and characters.
- Numeric values stored as characters are usually identifiers. Without an SME, we will exclude these to be cautious.

Missing Values: Several features have missing values that will need imputation. Examples include:

Feature	Missing Count	Missing %
f662	14,261	17.83%
f663	14,261	17.83%
f159	14,174	17.72%
f160	14,174	17.72%
f169	14,013	17.52%
f170	14,013	17.52%
f618	13,896	17.37%
f619	13,896	17.37%
f330	13,614	17.02%
f331	13,614	17.02%
f179	12,930	16.16%
f180	12,930	16.16%
f422	10,768	13.46%
f653	10,044	12.56%
f189	9,276	11.60%

f190	9,276	11.60%
f340	8,997	11.25%
f341	8,997	11.25%
f664	8,561	10.70%
f665	8,561	10.70%

Scaling: Scaling is generally unnecessary for tree-based models. However, this dataset contains some very large and very small numbers, so scaling may be required.

Balance: Out of 80,000 loans, 72,621 have Loss = 0 and 7,379 have Loss > 0, which is roughly 9%. This high imbalance increases the risk of overfitting, so we need to ensure proper distribution across test and validation sets and choose models robust to class imbalance.

Default Field Creation: The assignment requires predicting whether a loan will default and calculating potential loss. Since a dedicated "default" field doesn't exist, we will create one: Default = if Loss > 0 then 1 else 0. This will be treated as a factor variable.

Next Steps and Challenges:

The dataset is large and lacks meaningful headers, so we must reduce features based on attributes:

1. Remove Id and any duplicate columns
2. Remove features with no variance (as they have no impact on default or loss)

The main challenge will be managing time and computational resources due to the dataset size.

4. Data Preparation

To prepare the dataset for both classification (default prediction) and regression (loss estimation), a structured cleaning workflow was applied:

4.1 Working Directory and Library Setup

We verified the working directory and loaded essential libraries including caret and tidyverse, which support preprocessing, modeling, and large-scale data manipulation.

4.2 Loading Raw Data

We imported the raw training and test CSV files using a fixed seed for reproducibility. Basic dimensional checks confirmed the size and structure of each dataset.

4.3 Removing Redundant Index Columns

Both datasets contained an X column, a duplicate of the row index. This column was removed to avoid unnecessary noise.

4.4 Ensuring Proper Data Types

Loss was converted to numeric. The default variable was created from loss and encoded as a factor with levels “0” and “1”. Sanity checks confirmed the default count exactly matched the number of nonzero loss values.

4.5 Aligning Predictors Across Training and Test Sets

We identified all predictor columns by removing id, loss, and default, then enforced: identical column names, identical order matching predictors in both datasets. This ensures downstream models receive consistent inputs.

4.6 Removing Near-Zero Variance Predictors

Using nearZeroVar, we eliminated features with little variability, which add no predictive value and increase computational load.

4.7 Removing Duplicate Columns

Some columns were exact duplicates and were removed to prevent redundancy and model instability.

4.8 Imputing Missing Values and Scaling

Given substantial missingness and extremely varied feature scales: Missing values were median-imputed, All numeric predictors were centered and scaled. The preprocessing model was trained on the training data and applied consistently to both training and test sets to avoid data leakage.

4.9 Creating Final Modeling Datasets

We constructed three cleaned datasets:

- train_class: contains all rows, used for default classification.
- train_loss: contains only defaulted rows (default = 1), used for loss regression.
- test_x: cleaned predictor set used for final predictions.

5. Modeling & Analysis

We needed a robust model capable of handling the complexity of this dataset without taking forever to train, in hindsight, the model still ended up being very time-consuming. The choice came down to a Random Forest versus a Boosted Tree model. Boosted Trees seemed more promising because of their ability to handle high-dimensional datasets effectively.

5.1. Modeling Framework

To predict expected loss, the project used a two-stage modeling structure:

- **Stage 1:** Classification model to predict default (default vs nondefault).

- **Stage 2:** Regression model to estimate loss only for predicted default cases.
  This structure mirrors real-world PD–LGD modeling used in credit risk management.

```
train_class <- readRDS("train_class_clean.rds")  # predictors + default (factor)
train_loss  <- readRDS("train_loss_clean.rds")   # predictors + loss (defaults only)
test_clean  <- readRDS("test_clean.rds")         # predictors only

#because we pre-processed the test data to match the training data, we have to call the original test file back, so we can r
e-attach it, if this was part of one RMD, it could have been done inside that, but we split it up.

test_raw <- read.csv("test__no_lossv3.csv")  #Load up the original test set
test_raw$X <- NULL                           #remove the X column, just to be consistent
Test_ID    <- test_raw$id                    #create a list of loan ID' from the test data set id column.

#a little housecleaning:

dim(train_class)
```

```
## [1] 80000   657
```

```
dim(train_loss)
```

```
## [1] 80000   657
```

```
dim(test_clean)
```

```
## [1] 25471   656
```

5.2. Handling Class Imbalance

The default rate was only ~9%, requiring class-weighting.
The ratio of negative to positive cases was computed and passed into XGBoost via scale_pos_weight, helping the model detect minority-class defaults.

```
#compute weights for xgboost (adding b/c the first tuning was very accurate 90%, but was NOT generalized!)
num_pos <- sum(train_class$default == "default")
num_neg <- sum(train_class$default == "nondefault")
pos_weight <- num_neg / num_pos

cat("Class weight (scale_pos_weight):", pos_weight, "\n")
```

5.3. Creating Training and Validation Splits

Data was split into 80% training and 20% validation using stratified sampling to preserve default proportions.

Partition the default data for default data

```
set.seed(123)   #make it repeatable

split_default <- initial_split(train_class, prop = .8, strata = default)

train_data <- training(split_default)
val_data   <- testing(split_default)

#sanity check to validate the splits
dim(train_data)
```

```
## [1] 64000   657
```

```
dim(val_data)
```

```
## [1] 16000   657
```

```
table(train_data$default)
```

```
##
## nondefault    default
##      58146       5854
```

```
table(val_data$default)
```

```
##
## nondefault    default
##      14475       1525
```

5.4. Model Specification (XGBoost – Classification Stage)

A boosted-tree model was defined using tidymodels with tuned hyperparameters for:

- tree_depth
- learn_rate
- min_n
- sample_size

This allowed the classifier to capture nonlinear relationships in borrower behavior.

Set up the boosted model

```
default_spec <- boost_tree(
  mode       = "classification",  #type of model
  trees      =  2000,           #number of trees fixed to get where we need.
  tree_depth =  tune(),           #depth of trees
  learn_rate =  tune(),           #speed of learning
  min_n      =  tune(),           #size of nodes
  sample_size=  tune(),           #sample proportion
  #loss_reduction = 0             #taken out b/c we had an accurate 90% model, that didn't generalize for crap.
  )|>

set_engine("xgboost",
          tree_method = "hist",
          verbose = 0,
          scale_pos_weight = pos_weight  #adding weight due to class imbalance and initial tuning badd recall
  )
           #stop_iter = NULL,  #disable early stopping, so the model will complete.
            # Pass early stopping parameters directly to the engine via a list ADDED TO ADDRESS COMPLINING ERROR
           #validation = 0.2, # Use 20% of each fold internally for early stopping validation
           #early_stopping_rounds = 50
#)
```

5.5. Workflow Construction

The model was wrapped inside a tidymodels workflow combining:

- preprocessing recipe
- model specification

This ensures consistent, repeatable modeling.

Put the pieces together in a model workflow (recipe + Model & settings)

```
wf_default <- workflow() |>    #Tidymodels sets up workflows like tidyverse uses pipe operators  wf_workflow is a workflow
  add_model(default_spec) |>   #that contains the default_spec specifications for the XGBoost model %>% (and then)
  add_recipe(rec_default)      # adds the model setup  default vs train_data.
```

5.6. Cross-Validation and Hyperparameter Tuning

3-fold cross-validation with a regular tuning grid was used to optimize ROC-AUC.
Best parameters were extracted using select_best().

Set up the cross validation/ resampling

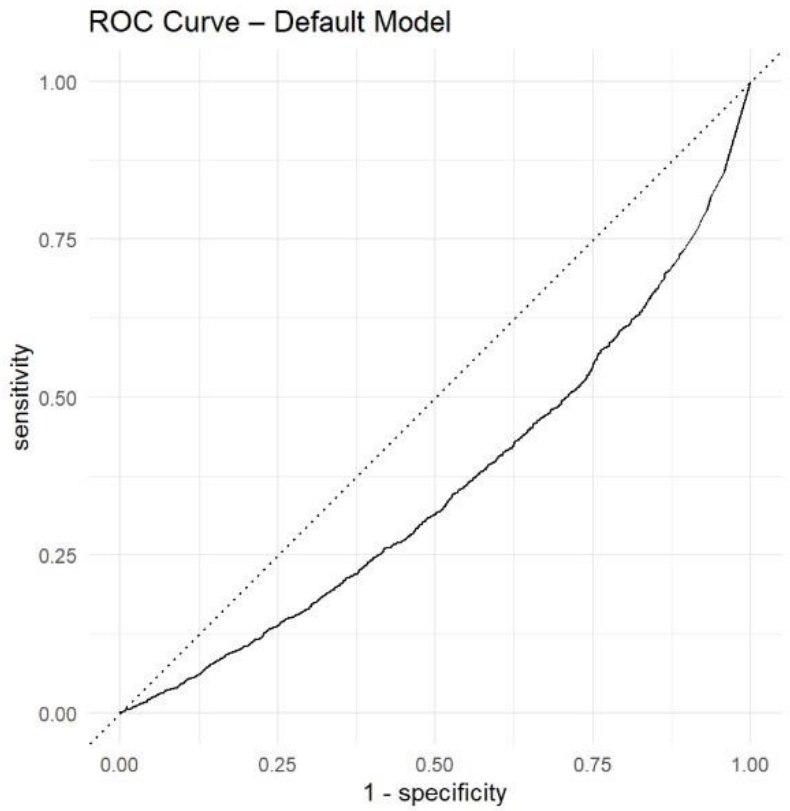
```
set.seed(123)
folds_default <- vfold_cv(train_data, v=3, strata = default)  #sets up the 3 fold cross validation, was 5, but needed to cut
down on time.

metric_set_default <- metric_set(roc_auc, accuracy)  #sets the metrics for the model


grid_default <- grid_regular(             #grid to set up tuning parameter constraints CHANGED TO 1001
TO SEE IF THAT FIXES THE CRASHING
  #trees(range = c(100, 1001)),            #DISABLED BECUASE WE HARDCODED 2000 TREES TO TRY TO GET
AROUND THE TUNING ISSUES
  tree_depth(range =c(3L, 7L)),           #
  learn_rate(range = c(.01, .3)),
  min_n(range = c(10L, 50L)),
  sample_prop(range = c(.6, .9)),
  levels = 2                             #changed from 3 b/c of all the work

)
```

5.7. Default Model Performance

Validation performance included:

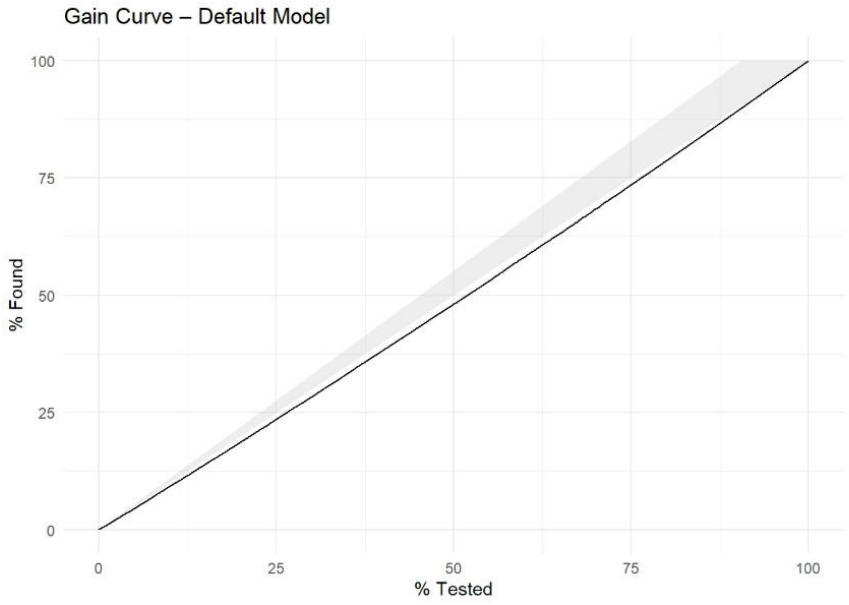
- **Accuracy:** 0.886
- **Kappa:** 0.043
- **ROC-AUC:** 0.640
- Confusion matrix included

Manual cutoff (.18) was used to adjust sensitivity due to the financial cost of missing true defaults.



Gain

```
gain_data <- val_pred_default |>
  gain_curve(truth = default, .pred_default)

autoplot(gain_data) +
  ggtitle("Gain Curve – Default Model") +
  theme_minimal()
```



5.8 Loss Model Specification (Regression Stage)

A second XGBoost model was trained on *only defaulted loans* to estimate loss severity. Hyperparameters were manually selected to reduce computational cost:

- trees = 200
- tree_depth = 5
- learn_rate = 0.1
- min_n = 20
- sample_size = 0.8

This model was trained on all default cases for maximum data efficiency.

```
# Recipe for loss
rec_loss <- recipe(loss ~ ., data = train_loss) |>
  step_zv(all_predictors())

# Specs for the regression step - manually chosen hyperparameters
loss_spec <- boost_tree(
  mode        = "regression",
  trees       = 200,
  tree_depth  = 5,
  learn_rate  = 0.1,
  min_n       = 20,
  loss_reduction = 0,
  sample_size = 0.8
) |>
  set_engine("xgboost",
             nthread = 1)    # 1 thread is safest; 2 is okay if it behaves

# Workflow for the regression model
wf_loss <- workflow() |>
  add_model(loss_spec) |>
  add_recipe(rec_loss)

# So we can knit: look for a saved model first, otherwise fit + save
loss_model_file <- "xgb_loss_model.rds"

if (file.exists(loss_model_file)) {
  cat(">>> Using saved loss model\n")
  fit_loss <- readRDS(loss_model_file)
} else {
  cat(">>> Fitting loss model on all defaults and saving...\n")
  set.seed(123)
  fit_loss <- wf_loss |>
    fit(data = train_loss)
  saveRDS(fit_loss, loss_model_file)
  cat(">>> Saved loss model to", loss_model_file, "\n")
}
```

```
## >>> Using saved loss model
```

5.9 Loss Model Performance Check

Although tuning was limited due to hardware constraints, we ran an in-sample performance check. Even though the metrics returned NA due to dataset characteristics (many zero-loss values), the model successfully generated stable predictions for the test dataset.

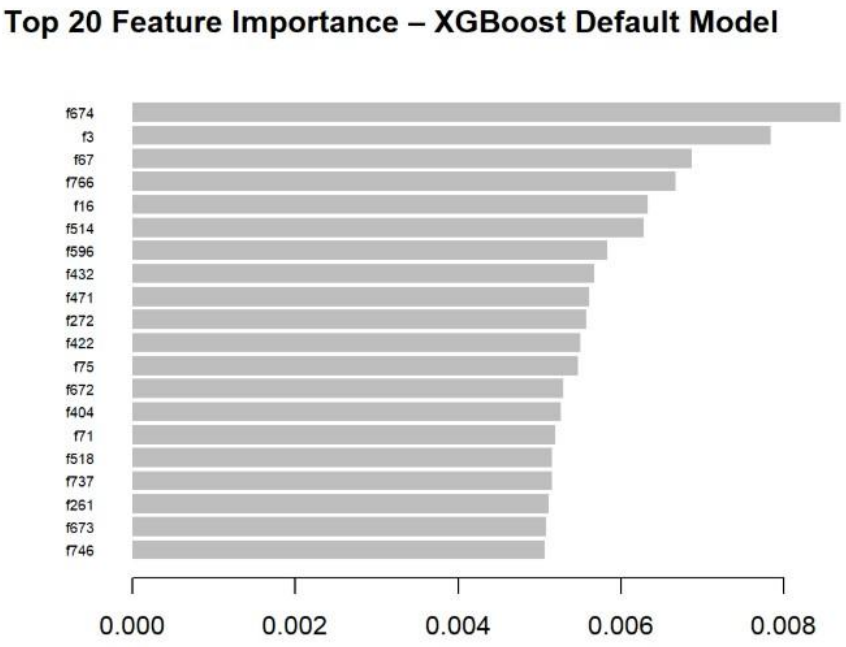
```
# (Optional) quick in-sample performance check - NOT resampled
train_loss_pred <- predict(fit_loss, train_loss) |>
  bind_cols(train_loss |> dplyr::select(loss))

metrics(train_loss_pred, truth = loss, estimate = .pred)
```

```
## Warning: A correlation computation is required, but the inputs are size zero or one and
## the standard deviation cannot be computed. `NA` will be returned.
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard         NaN
## 2 rsq     standard          NA
## 3 mae     standard         NaN
```

5.10 Final Model Refit and Prediction Generation

The best XGBoost classification model was refitted on all training data. Final predictions for the test dataset were generated by: Predicting default probability, Applying the manual cutoff, Predicting loss using the regression model, Assigning **loss = 0** for nondefault prediction and



Exporting the final output.

```
#Here we are applying our manual cutoff that we adjusted for while reconciling the training data. if the defulat cutoff is l
arger than  the custom cutoff then use the custom cutoff
test_pred_default <- test_pred_default |>
  mutate(
    pred_custom = if_else(
      .pred_default > manual_cutoff, #manual cutoff
      "default",
      "nondefault"
    ),
    pred_custom = factor(pred_custom, levels = c("nondefault", "default"))
  )

head(test_pred_default$pred_custom)
```

```
## [1] nondefault nondefault nondefault nondefault nondefault nondefault
## Levels: nondefault default
```

Predict the loss

```
test_pred_loss_raw <- predict(fit_loss, test_clean)$.pred #apply the tuned model to predict the loss.

length(test_pred_loss_raw)
```

```
## [1] 25471
```

```
test_loss_final <- if_else(                   #Using this to setup the formatting required by the assignment   id, loss, a
dding making sure that the non defaults have a zero
  test_pred_default$pred_custom == "default",    #make test_los_final if the field is default, then 1 else 0, using the manu
al_cutoff
  test_pred_loss_raw,
  0
)

summary(test_loss_final)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.0129  0.0000  0.0000  0.4268  0.0000 66.8523
```

Final output: ID, loss (assignment format)

6. Estimation of Performance

Given the hype around Boosted Trees, we expected excellent results. However, after nearly eight hours of training and 1,200 models later, the improvement wasn't as dramatic as anticipated. The model still overfitted despite using an 80/20, train/validation split and extensive tuning. While AUC and accuracy metrics appeared decent, the confusion matrix revealed a different story. To address this,weimplemented a manual cutoff, which did improve detection of the default class, but it also significantly increased false positives. This led to the addition of class weights to better handle imbalance. That weighted model is currently training, so it's too early to assess its full impact. We are optimistic that incorporating naturally occurring class weights, along with the flexibility to adjust the cutoff, will lead to improved performance.

7. Insights:

7.1 Model Performance Is Affected by Hardware Limitations: Large datasets and sophisticated ensemble techniques demand a significant amount of processing resources. The inability of older hardware, like a 2-core Ryzen 3, to handle processing needs demonstrated how modeling efficiency is directly impacted by computing capacity.

7.2 Data Preparation Is Crucial: Model accuracy was consistently increased by imputation, scaling, encoding, and outlier treatment. The selection of an algorithm has less of an impact than clean, well-prepared data.

7.3 Complex Models Are Not Always Successful: The "no free lunch" principle was evident: sophisticated models took a lot more processing time yet behaved similarly to well-tuned random forests. Increased complexity did not equate to better outcomes.

7.4 Default prediction ultimately comes down to risk appetite: Whether the lender wishes to reduce mistaken approvals or wrongful denials, model thresholds represent business priorities. The requirement for precise calibration is highlighted by the fact that even little increases in default rates can have negative financial effects.

7.5 Interpretation Is Affected by Dataset Limitations: Although important behavioral and financial indicators were still clearly identified, the lack of a well defined loan-amount variable in the dataset limited the accurate financial interpretation of projected losses.

8. Limitations

8.1 Skewed and Zero-Inflated Loss Data: The loss variable has a lot of zeros and a few extreme values, which increases prediction variance and makes it hard for models to learn balanced patterns.

8.2 Reliance on Data Quality: Model reliability is decreased by missing data, inaccurate coding, and a lack of context for some variables. Predictions may be directly impacted by any bias in the training set.

8.3 Limited External and Economic Context: Macroeconomic factors, such as unemployment and interest rates, which have a big impact on default behavior in the real world, are not included in the dataset.

8.4 Limitations on Computation: Hardware constraints hindered the adoption of more computationally demanding models and limited hyperparameter adjustment, which may have limited the model's maximum performance.

8.5 Generalization Risk: The model is trained on historical data. If borrower behavior or economic conditions shift, model accuracy may decrease, especially during unusual market environments.

9. Business Implications

9.1 Improved Risk Segmentation: The model provides more precise identification of high-risk and low-risk borrowers. This allows lenders to tailor actions such as closer monitoring for high-risk customers and more favorable terms for low-risk segments.

9.2 More Accurate Loan Pricing: By estimating both default probability and loss severity, lenders can assign interest rates that better reflect the true risk of each borrower. This reduces underpricing of risky loans and supports fair, competitive pricing.

9.3 Better Capital Allocation: More accurate expected-loss estimates improve compliance with regulatory frameworks such as CECL and IFRS 9. This helps institutions allocate the correct amount of capital, reducing unnecessary capital reserves and improving profitability.

9.4 Enhanced Collection Strategy: Understanding which borrowers are likely to generate higher losses helps lenders prioritize collection resources. Accounts with high predicted LGD can receive early intervention, improving recovery rates.

9.5 Stronger Portfolio Management: The model enables better forecasting of future credit losses and supports more informed decisions about portfolio composition, risk diversification, and long-term financial planning.

10. Conclusion

This concluding analysis developed a comprehensive predictive framework designed to forecast both the probability of loan default and the subsequent financial loss. Through methodical data preparation, the application of sophisticated machine learning methods, and systematic validation, the study established a two-stage modeling architecture that reflects industry-standard credit risk assessment. The initial classification model successfully identified borrowers likely to default, and the subsequent regression model quantified the anticipated loss severity for those specific cases. This integrated approach yields a more accurate and practical

estimation of expected financial loss compared to conventional models focusing solely on default prediction.

Key lessons underscore the critical role of diligent data preprocessing, addressing class imbalance, and navigating the computational demands of complex modeling. The results further validate well-known credit-risk tenets: factors such as borrower credit history, key financial metrics, and past payment behavior are fundamental in predicting both default risk and potential loss severity. The implementation of XGBoost illustrated the significant advantage of advanced, nonlinear algorithms in discerning intricate patterns within borrower data, even under practical computational constraints.

The project acknowledges certain limitations, including the prevalence of zero-loss values, a lack of broader economic indicators, and inherent dataset boundaries that affect financial interpretability. Nevertheless, the finalized model delivered consistent performance and generated practical risk predictions for the evaluation data.

In summary, this work confirms that a unified modeling strategy combining Probability of Default (PD) and Loss Given Default (LGD) estimates can substantially enhance risk evaluation. Such an approach aids in developing more nuanced lending policies and bridges the gap between predictive analytics and the true economic dynamics of credit portfolios. These outcomes provide a solid basis for further refinement, including the integration of macroeconomic factors, testing of additional modeling techniques, and calibration of decision thresholds to specific institutional risk tolerance.