1. Setup and Imports

- **Purpose:** This section installs necessary libraries and imports them for use in the notebook.
- **Steps:**
    - Installs `xgboost`, `pandas-profiling`, `joblib`, and `scikit-learn`.
    - Imports various libraries for data manipulation (`pandas`, `numpy`), visualization (`matplotlib.pyplot`, `seaborn`), preprocessing (`sklearn.preprocessing`, `sklearn.impute`), model building (`sklearn.ensemble`, `sklearn.linear_model`, `xgboost`, `sklearn.svm`), and evaluation (`sklearn.metrics`, `sklearn.model_selection`).
- **Potential Results:** Successful installation and import of the required libraries, allowing subsequent code to run.

2. `modelfit` Function Definition

- **Purpose:** Defines a function to train an XGBoost model and evaluate its performance using cross-validation and metrics like accuracy and AUC.
- **Steps:**
    - Takes an XGBoost classifier object, training data, and predictors as input.
    - Optionally performs cross-validation to determine the optimal number of boosting rounds.
    - Fits the XGBoost model to the training data.
    - Makes predictions and calculates prediction probabilities.
    - Prints the accuracy and AUC score on the training data.
    - Plots the feature importances.
- **Potential Results:** Output of the model's performance metrics and a bar plot of feature importances.

3. Encoding

- **Purpose:** Reads the training data and performs one-hot encoding on categorical features.
- **Steps:**
    - Reads `train_x.csv` and `train_y.csv`.
    - Creates duplicate columns for 'Loan type' and 'Occupation type' to perform one-hot encoding.
    - Renames columns for clarity.
    - Assigns numerical values (0 or 1) to the encoded categorical columns.
    - Selects relevant columns and drops the 'ID' column.
    - Defines lists of categorical and numerical column names.
- **Potential Results:** A pandas DataFrame `trainx` with categorical features encoded numerically.

4. Normalizing

- **Purpose:** Scales the numerical features of the training data using Min-Max scaling.
- **Steps:**
    - Initializes a `MinMaxScaler`.
    - Applies the scaler to the numerical columns in `trainx`.
    - Converts the scaled data back to a pandas DataFrame.
    - Creates copies of the features (`X_imp`) and the target variable (`Y_imp`).
- **Potential Results:** A pandas DataFrame `trainx` with numerical features scaled between 0 and 1.

5. Imputing

- **Purpose:** Handles missing values in the data using `IterativeImputer` and `KNNImputer`.
- **Steps:**
    - Separates the numerical columns into `trainx_cont`.
    - Initializes and applies `IterativeImputer` to the numerical data to fill missing values.
    - Converts the imputed numerical data back to a DataFrame, adding back the categorical columns.
    - Initializes `KNNImputer`.
    - Performs KNN imputation on the features.
    - Adds the target variable to the imputed features.
    - Performs a second round of KNN imputation on the combined features and target.
    - Splits the data back into features (`trainx_imp_final`) and target (`y`).
- **Potential Results:** DataFrames `trainx_imp_final` and `y` with missing values imputed.

6. Reverse Normalizing

- **Purpose:** Reverses the Min-Max scaling on the imputed numerical features.
- **Steps:**
    - Creates a copy of the imputed features.
    - Applies the inverse transform of the `MinMaxScaler` to the numerical columns.
    - Converts the reversed scaled data back to a DataFrame.
    - Creates copies of the features (`X`) and target (`Y`).
    - Renames the target column in `Y` to 'Label'.
- **Potential Results:** A pandas DataFrame `X` with numerical features in their original scale and a DataFrame `Y` with the target variable.

7. Correlation Heatmap

- **Purpose:** Visualizes the correlation matrix of the features to identify highly correlated variables.
- **Steps:**

    - Generates a heatmap using `seaborn.heatmap` based on the correlation matrix of X.
- **Potential Results:** A heatmap image showing the pairwise correlations between features.

8. Dropping Dummyvar Trap

- **Purpose:** Removes features that are highly correlated due to the dummy variable trap to improve model accuracy.
- **Steps:**
    - Drops the columns 'Loan_type_B', 'Occupation_type_Z', 'Score5', and 'Age' from X.
    - Creates a copy of X called XY.
    - Adds the target variable 'Label' from Y to XY as 'target'.
    - Creates a copy of XY called `train`.
- **Potential Results:** A pandas DataFrame X and XY with the specified columns removed.

9. Fixing Learning Rate and Number of Estimators

- **Purpose:** Uses the `modelfit` function to find an initial estimate for the number of estimators based on a fixed learning rate.
- **Steps:**
    - Defines the predictors (all columns in XY except 'target').
    - Initializes an `XGBClassifier` with a fixed learning rate and a large number of estimators.
    - Calls the `modelfit` function with the initialized model and data.
- **Potential Results:** Output from the `modelfit` function, including accuracy and AUC score, and the model's internal number of estimators adjusted based on cross-validation.

10. Tuning Max_depth and Min_child_weight

- **Purpose:** Tunes the `max_depth` and `min_child_weight` hyperparameters of the XGBoost model using GridSearchCV.
- **Steps:**
    - Defines a parameter grid with different values for `max_depth` and `min_child_weight`.
    - Initializes an `XGBClassifier` with some fixed parameters.
    - Initializes a `GridSearchCV` object with the model, parameter grid, scoring metric ('roc_auc'), and cross-validation settings.
    - Fits the `GridSearchCV` to the data.
- **Potential Results:** Output showing the best combination of `max_depth` and `min_child_weight` and the corresponding best ROC-AUC score found by the grid search.

11. Tune Gamma

- **Purpose:** Tunes the `gamma` hyperparameter of the XGBoost model using GridSearchCV.
- **Steps:**
    - Defines a parameter grid with different values for `gamma`.
    - Initializes an `XGBClassifier` with previously tuned parameters.
    - Initializes and fits a `GridSearchCV` object.
- **Potential Results:** Output showing the best value for `gamma` and the corresponding best ROC-AUC score.

12. Recalibrating the Classifier

- **Purpose:** Re-evaluates the model performance using the `modelfit` function with the updated hyperparameters (including the tuned `gamma`).
- **Steps:**
    - Initializes an `XGBClassifier` with the updated hyperparameters.
    - Calls the `modelfit` function with the new model and data.
- **Potential Results:** Output showing the model's performance metrics (accuracy and AUC) with the recalibrated parameters.

13. Tuning subsample and colsample_bytree

- **Purpose:** Tunes the `subsample` and `colsample_bytree` hyperparameters of the XGBoost model using GridSearchCV.
- **Steps:**
    - Defines a parameter grid with different values for `subsample` and `colsample_bytree`.
    - Initializes an `XGBClassifier` with previously tuned parameters.
    - Initializes and fits a `GridSearchCV` object.
- **Potential Results:** Output showing the best combination of `subsample` and `colsample_bytree` and the corresponding best ROC-AUC score.

14. Tuning reg_alpha

- **Purpose:** Tunes the `reg_alpha` (L1 regularization) hyperparameter of the XGBoost model using GridSearchCV.
- **Steps:**
    - Defines a parameter grid with different values for `reg_alpha`.
    - Initializes an `XGBClassifier` with previously tuned parameters.
    - Initializes and fits a `GridSearchCV` object.
- **Potential Results:** Output showing the best value for `reg_alpha` and the corresponding best ROC-AUC score.

15. Tuning reg_lambda

- **Purpose:** Tunes the `reg_lambda` (L2 regularization) hyperparameter of the XGBoost model using GridSearchCV.
- **Steps:**

    - Defines a parameter grid with different values for `reg_lambda`.
    - Initializes an `XGBClassifier` with previously tuned parameters (including the tuned `reg_alpha`).
    - Initializes and fits a `GridSearchCV` object.
- **Potential Results:** Output showing the best value for `reg_lambda` and the corresponding best ROC-AUC score.

16. Final Model

- **Purpose:** Initializes and trains the final XGBoost model with all the tuned hyperparameters and a lower learning rate.
- **Steps:**
    - Initializes an `XGBClassifier` with the final set of hyperparameters.
    - Calls the `modelfit` function to train and evaluate the model.
- **Potential Results:** Output showing the performance metrics (accuracy and AUC) of the final trained model.

17. Train-Test Split

- **Purpose:** Splits the training data into training and validation sets for evaluating the model's generalization performance.
- **Steps:**
    - Splits the data `X` and `Y` into training and validation sets using `train_test_split`, with stratification to maintain the proportion of the target variable.
    - Fits the final XGBoost model (`xgb4`) to the training data (`X_train`, `Y_train`).
    - Makes predictions on both the training and validation sets.
    - Calculates the F2 score and accuracy on both sets.
    - Prints the calculated metrics.
- **Potential Results:** Output showing the F2 score and accuracy of the model on both the training and validation datasets.

18. KFold Cross Validation Score

- **Purpose:** Evaluates the model's performance using K-Fold cross-validation to get a more robust estimate of its performance.
- **Steps:**
    - Initializes a `KFold` object for cross-validation.
    - Calculates cross-validation scores (accuracy) using the trained model (`model1`), features (`X`), and target (`Y`).
    - Calculates the mean of the cross-validation scores.
- **Potential Results:** A single numerical value representing the average accuracy of the model across the cross-validation folds.

19. Importing Test Data and Preprocessing

- **Purpose:** Reads the test data and applies the same preprocessing steps as were applied to the training data.
- **Steps:**
    - Reads `test_x.csv`.
    - Performs the same one-hot encoding and renaming of columns as done for the training data.
    - Drops the same columns that were dropped from the training data (`ID_Test`, 'Loan_type_B', 'Occupation_type_Z', 'Age', 'Score5').
    - Converts certain columns to numeric types.
- **Potential Results:** A pandas DataFrame `testx` with the test data preprocessed in the same way as the training data.

20. Predicting the Values

- **Purpose:** Uses the trained model to make predictions on the preprocessed test data and saves the predictions to a CSV file.
- **Steps:**
    - Fits the final XGBoost model (`xgb4`) to the entire training data (X, Y).
    - Makes predictions on the preprocessed test data (`testx`).
    - Creates a pandas DataFrame `y_pred_f` from the predictions.
    - Adds an 'ID' column to `y_pred_f`.
    - Reorders the columns to have 'ID' first.
    - Saves the `y_pred_f` DataFrame to a CSV file named 'pred_y.csv' without the index.
- **Potential Results:** A CSV file named 'pred_y.csv' containing the predicted 'Label' for each ID in the test data.