1. Identify which type of machine learning problem is it?

It is a prediction problem with binary classification.

In this scenario, the target variable (i.e., what we want to predict) is whether each applicant is capable or not capable of repaying a loan. This can often be represented as a binary outcome, where an applicant is classified into one of two categories: likely to repay (positive class) or unlikely to repay (negative class).

1. Please suggest models that the team may consider for the problem at hand. Suggest at least three models and discuss their pros and cons.

Logistics regression

Pros:

* Simple and easy to implement.
* Outputs probabilities which can be interpreted as likelihood of loan repayment.
* Fast training and prediction times, suitable for large datasets.
* Provides coefficients that indicate feature importance.

Cons:

* Assumes a linear relationship between features and log odds of the target.
* May underperform if the relationship between features and target is non-linear.
* Sensitive to outliers and irrelevant features.
* Logistic regression may not be accurate if the sample size is too small.

Random Forest

Pros:

* Capable of capturing complex non-linear relationships in the data.
* Robust to outliers and does not require feature scaling.
* Automatically handles feature importance through ensemble learning.
* Less prone to overfitting compared to decision trees.

Cons:

* Can be computationally expensive and slower to train compared to simpler models like logistic regression.
* May be difficult to interpret compared to linear models.
* Hyperparameter tuning is required to optimize performance.

Gradient Boosting - XGBoost / LightGBM

Pros:

* Highly effective for predictive modeling tasks due to boosting (ensemble) technique.
* Handles complex interactions between features and target variable.
* Often outperforms other algorithms in terms of accuracy.
* Can handle missing data effectively.

Cons:

* Requires careful tuning of hyperparameters.
* Prone to overfitting if hyperparameters are not properly tuned or if the model complexity is too high.
* Training time can be longer compared to simpler models.

1. Suggest the metric(s) your team wants to use to compare the models. Justify your logic based on business goals. If needed, define your own cost function and pass it under "scoring" parameter in GridsearchCV().

For a binary classification problem like loan repayment prediction, the choice of metric(s) should be driven by the business goals and the relative costs associated with the different types of errors (false positives and false negatives).

In this context, some relevant metrics to consider are:

1. **Accuracy**: This is the overall fraction of correct predictions. However, accuracy can be misleading when the classes are imbalanced (e.g., if most applicants repay their loans, a naive model that always predicts "will repay" would have high accuracy but be useless).
2. **Precision**: This metric measures the fraction of positive predictions (predicted to be unable to repay) that are actually correct. A high precision is important if the cost of investigating a false positive (denying a loan to someone who would have repaid) is high.
3. **Recall (or Sensitivity)**: This metric measures the fraction of actual positive instances (unable to repay) that are correctly identified by the model. High recall is crucial if the cost of missed detections (false negatives, approving a loan for someone who won't repay) is high.
4. **F1-score**: This is the harmonic mean of precision and recall, providing a balanced measure when both are important.

The choice of metric(s) depends on the relative importance of false positives vs. false negatives from a business perspective. If the primary goal is to minimize the risk of approving loans to defaulters (i.e., minimize false negatives), then recall or F1-score would be more appropriate. However, if the cost of investigating false positives is also high, then a balanced metric like an F1 score is preferable.

**Results:**

Since, the target variable is imbalanced, with the majority of applicants being able to repay their loans.

The metrics reported for each model are:

* Confusion Matrix
* Precision
* Recall
* F1 Score
* Accuracy

Out of these, the F1 score is being used as the primary metric for model selection. The justification for using the F1 score is that it provides a balanced measure between precision and recall, which is important when dealing with an imbalanced target variable.

In a loan repayment prediction scenario, both precision and recall are crucial:

* High precision is desirable to minimize the cost of investigating false positives (denying loans to applicants who would have repaid them).
* High recall is important to minimize the risk of false negatives (approving loans for applicants who won't repay).

The F1 score combines precision and recall into a single metric, making it a suitable choice when both types of errors (false positives and false negatives) have significant costs associated with them.

The results show that the Gradient Boosting model has the highest F1 score of 3.21%, which is why it is chosen as the model of choice. However, it's important to note that all models perform poorly on this task, with very low precision, recall, and F1 scores, likely due to the extreme class imbalance in the data.

1. Identify the data preparation steps required.

Handling Data Imbalance: we identified that there's a significant class imbalance in the target variable, with one class (0) representing 91% of the data. This can lead to biased model training where the model might become overly sensitive to the majority class and overlook the minority class.

Handling Null Values: The table provided shows the number of null values for specific features in the dataset.

|  |  |  |
| --- | --- | --- |
| Column(s) | miss | miss\_pct |
| SK\_ID\_CURR | 0 | 0 |
| REG\_CITY\_NOT\_WORK\_CITY | 0 | 0 |
| REG\_CITY\_NOT\_LIVE\_CITY | 0 | 0 |
| LIVE\_REGION\_NOT\_WORK\_REGION | 0 | 0 |
| LIVE\_CITY\_NOT\_WORK\_CITY | 0 | 0 |
| REG\_REGION\_NOT\_LIVE\_REGION | 0 | 0 |
| AMT\_CREDIT | 0 | 0 |
| REGION\_POPULATION\_RELATIVE | 0 | 0 |
| DAYS\_BIRTH | 0 | 0 |
| DAYS\_EMPLOYED | 0 | 0 |
| DAYS\_REGISTRATION | 0 | 0 |
| DAYS\_ID\_PUBLISH | 0 | 0 |
| REG\_REGION\_NOT\_WORK\_REGION | 0 | 0 |
| AMT\_INCOME\_TOTAL | 0 | 0 |
| REGION\_RATING\_CLIENT | 0 | 0 |
| TARGET | 0 | 0 |
| REGION\_RATING\_CLIENT\_W\_CITY | 0 | 0 |
| HOUR\_APPR\_PROCESS\_START | 0 | 0 |
| CNT\_CHILDREN | 0 | 0 |
| DAYS\_LAST\_PHONE\_CHANGE | 1 | 0 |
| CNT\_FAM\_MEMBERS | 2 | 0 |
| AMT\_ANNUITY | 12 | 0 |
| AMT\_GOODS\_PRICE | 278 | 0 |
| EXT\_SOURCE\_2 | 660 | 0 |
| DEF\_60\_CNT\_SOCIAL\_CIRCLE | 1021 | 0 |

1. Set up preprocessor and estimators using column transformers and pipeline.  While setting up preprocessing make your pipeline robust so that if a new category shows up in future, the pipeline does not break. You must also explicitly state whether you want to the model to impute missing values at the time of inference. If yes, set up your pipelines to handle them.
2. **Pipeline Setup for Preprocessing**:
   * Several pipelines are defined for preprocessing numeric and categorical variables separately.
   * **skewed\_num\_pipe**, **norm\_num\_pipe**, **disc\_pipe**, **nom\_cat\_pipe**, **ord\_cat\_pipe**, and **rare\_cat\_pipe** represent pipelines for different types of variables, such as skewed numeric, normally distributed numeric, discretized numeric, nominal categorical, ordinal categorical, and categorical variables with rare categories, respectively.
   * Each pipeline consists of preprocessing steps such as imputation (using **SimpleImputer**) and encoding (using **OneHotEncoder** for categorical variables).
3. **Specify Variable Lists**:
   * Lists of variable names for different variable types are defined, including nominal categorical variables (**nom\_cat\_vars**), ordinal categorical variables (**ord\_cat\_vars**), numeric variables requiring discretization (**disc\_num\_vars**), normally distributed numeric variables (**norm\_num\_vars**), and skewed numeric variables (**skewed\_num\_vars**).
4. **ColumnTransformer**:
   * A **ColumnTransformer** named **preprocessor** is defined, which applies the appropriate preprocessing pipeline to each type of variable based on the specified lists of variable names.
   * Each pipeline is associated with the corresponding variable type using the **transformers** parameter of **ColumnTransformer**.
   * The **remainder='passthrough'** parameter ensures that any columns not specified in the transformers list will be passed through without any changes.
5. **Model Training and Evaluation**:
   * Three classification pipelines (**modeling\_pipeline\_rf**, **modeling\_pipeline\_lr**, **modeling\_pipeline\_gb**) are defined, each consisting of the preprocessing pipeline (**pre**) followed by a specific classifier (**clf**: Random Forest, Logistic Regression, or Gradient Boosting).
   * Each pipeline is trained on the training data (**X\_train**, **y\_train**) and evaluated on the test data (**X\_test**, **y\_test**).
   * Evaluation metrics such as precision, recall, F1 score, accuracy, and confusion matrix are calculated for each classifier.
6. **Hyperparameter Tuning**:
   * For Logistic Regression and Random Forest models, hyperparameter tuning is performed using **GridSearchCV**.
   * The hyperparameter grids (**param\_grid\_lr**, **param\_grid\_rf**) specify the hyperparameters to be tuned and their corresponding values.
   * Grid search is configured to optimize the F1 score and perform 5-fold cross-validation.
7. **Print Best Parameters and Scores**:
   * The best parameters and cross-validation scores obtained from grid search are printed for Logistic Regression and Random Forest models.
8. **Perform GridSearchCV for Each Model:**
   * Loop over each model specified in the pipelines dictionary.
   * Perform GridSearchCV to find the best hyperparameters for each model using cross-validation.
   * Print the best parameters found and the corresponding cross-validation accuracy.
9. **Identify Categorical and Numerical Columns:**
   * Use the appropriate method (e.g., select\_dtypes) to identify categorical and numerical columns in the dataset.
10. **Define Preprocessing for Numerical and Categorical Data:**
    * Define preprocessing steps for numerical data to impute missing values, scale the data, and remove outliers.
    * Define preprocessing steps for categorical data to impute missing values and apply one-hot encoding.
    * Create a preprocessor with ColumnTransformer to apply these transformations to numerical and categorical columns separately.