**HO CHI MINH UNIVERSITY OF SCIENCE**

**FACULTY OF INFORMATION TECHNOLOGY**

**REPORT**

**Subject: Intro to Machine Learning Class:** 22KHDL1  
**Group:** 22127224\_22127257

**Students:**

Trương Thuận Kiệt – 22127224 Phạm Minh Mẫn - 22127257

**TP. Hồ Chí Minh, tháng 4, năm 2025**

Contents

[**Student information and task assignments** 3](#_Toc195902050)

[**General EDA** 3](#_Toc195902051)

[**First view of dataset** 3](#_Toc195902052)

[**Categorical columns** 4](#_Toc195902053)

[**Categorical features vs target** 7](#_Toc195902054)

[**Numerical features** 8](#_Toc195902055)

[**Missing values** 10](#_Toc195902056)

[**Numerical features** 11](#_Toc195902057)

[**Categorical features** 13](#_Toc195902058)

[**Data type of each column** 13](#_Toc195902059)

[**Correlation heatmap** 15](#_Toc195902060)

[**Phase 1** 16](#_Toc195902061)

[**Data Preprocessing** 16](#_Toc195902062)

[**Feature engineering** 16](#_Toc195902063)

[**Handling missing values** 17](#_Toc195902064)

[**Encoding values** 17](#_Toc195902065)

[**Log-transformation target** 18](#_Toc195902066)

[**Retyping** 18](#_Toc195902067)

[**Modelling** 19](#_Toc195902068)

[**Trick for training large datasets** 19](#_Toc195902069)

[**Training process** 20](#_Toc195902070)

[**Random Forest** 20](#_Toc195902071)

[**Decision Tree** 21](#_Toc195902072)

[**Evaluation** 21](#_Toc195902073)

[**Phase 2** 26](#_Toc195902074)

[**Data Preprocessing** 26](#_Toc195902075)

[**Feature engineering** 26](#_Toc195902076)

[**Handling missing values** 28](#_Toc195902077)

[**Encoding values** 28](#_Toc195902078)

[**Log transformation on target and new features** 28](#_Toc195902079)

[**Retyping** 29](#_Toc195902080)

[**Feature selection** 29](#_Toc195902081)

[**Modelling** 31](#_Toc195902082)

[**Training process** 31](#_Toc195902083)

[**XGBoost** 31](#_Toc195902084)

[**LightGBM** 32](#_Toc195902085)

[**Catboost** 35](#_Toc195902086)

[**Evaluation** 37](#_Toc195902087)

# **Student information and task assignments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student ID** | **Student Name** | **Tasks Assigned** | **Rate of completion** |
| 22127224 | Trương Thuận Kiệt | * Phase 1 * Phase 2 (EDA, Preprocessing and XGBoost, LightGBM) | 100% |
| 22127257 | Phạm Minh Mẫn | * Phase 2 (Catboost) * Phase 3 | 100% |

# **General EDA**

## **First view of dataset**

A black and white screen

Description automatically generated

* Although, we don’t have access to full meaning of each column but still we can guess some features:
  + **feature\_1**: Marital Status
  + **feature\_3**: Educational Level
  + **feature\_4**: Work Class
  + **feature\_6**: Living Environment
  + **feature\_15**: Frequency of some types of activities
  + **feature\_16**: Type of Residence
* Other features hopefully we can conclude at the end of EDA.

## **Categorical columns**

A group of blue bars

Description automatically generated

* Surprisingly, we can see that all unique values of categorical features have the same number of appearances.
* For more details:
  + **Gender:**
    - **Observation**: The dataset has a slightly higher proportion of Male compared to Female.
    - **Implication**: The imbalance is not severe and is unlikely to cause significant bias in the model.
  + **feature\_1 (Marital Status):**
    - **Observation**: The dataset has a balanced distribution across Married, Divorced, and Single categories.
    - **Implication**: The balance ensures that the model can learn patterns for all marital statuses without bias.
  + **feature\_13 (Education Level):**
    - **Observation**: The dataset shows a higher proportion of individuals with Bachelor's and Master's degrees compared to High School and PhD.
    - **Implication**: The model may perform better for individuals with higher education levels due to their higher representation.
  + **feature\_4 (Employment Status):**
    - **Observation**: The majority of individuals are Employed, followed by Self-Employed and Unemployed.
    - **Implication**: The imbalance may cause the model to focus more on the Employed category.
  + **feature\_6 (Location Type):**
    - **Observation**: The dataset has a balanced distribution across Urban, Rural, and Suburban locations.
    - **Implication**: The balance ensures that the model can learn patterns for all location types effectively.
  + **feature\_7 (Tier-related feature):**
    - **Observation**: The dataset has a relatively balanced distribution across Basic, Comprehensive, and Premium tiers.
    - **Implication**: The model can learn patterns for all subscription tiers without significant bias.
  + **feature\_15 (Frequency of Activity):**
    - **Observation**: The dataset shows a higher proportion of individuals with Monthly and Weekly activity compared to Daily and Rarely.
    - **Implication**: The model may perform better for individuals with more frequent activity due to their higher representation.
  + **feature\_14 (Binary Feature):**
    - **Observation**: The dataset has a binary distribution with a higher proportion of No compared to Yes.
    - **Implication**: The imbalance may cause the model to predict No more frequently.
  + **feature\_16 (Housing Type):**
    - **Observation**: The dataset has a balanced distribution across House, Apartment, and Condo.
    - **Implication**: The balance ensures that the model can learn patterns for all housing types effectively.
* **Observations on Time Feature (feature\_12)**
  + **Observation**: The time-related feature can be decomposed into components such as date, month, year, and sin/cos transformations to capture periodicity.
  + **Implication**: This decomposition can help the model better understand temporal patterns in the data.
* **Encoding Recommendations**
  + **One-Hot Encoding**: Gender, feature\_1, feature\_4, feature\_6, feature\_16.
    - Reason: These features are nominal (no inherent order) and have a small number of unique categories.
  + **Label Encoding**: feature\_14.
    - Reason: This binary feature can be encoded as 0 and 1 for simplicity.
  + **Ordinal Encoding**: feature\_3 (Education Level), feature\_7 (Tier-related feature), feature\_13 (Quality Rating), feature\_15 (Frequency of Activity).
    - Reason: These features have a natural order that can be captured using ordinal encoding.
  + **Frequency Encoding:** All categorical features.
    - Reason: Frequency encoding provides an additional numerical representation of categories, which can be useful for tree-based models.

## **Categorical features vs target**

A diagram of a diagram

Description automatically generated with medium confidence

* **Strong Predictors**
  + Features such as feature\_3 (Education Level), feature\_7 (Subscription Tier), feature\_13 (Quality Rating), and feature\_15 (Frequency of Activity) show clear ordinal trends and are likely to be strong predictors of the target variable.
* **Weak Predictors**
  + Features like Gender and feature\_14 have weaker impacts on the target variable and may have limited predictive power.

## **Numerical features**

A group of blue and black bars

Description automatically generated

* **Age**
  + **Observation**: The Age feature is evenly distributed across a range of values, with no significant skewness or gaps.
  + **Implication**: This feature is well-represented and can be used without additional preprocessing.
* **feature\_0**
  + **Observation**: The distribution is highly skewed to the left, with a large number of smaller values and a long tail of larger values.
  + **Implication**: The skewness may require transformation (e.g., log transformation) to normalize the distribution.
* **feature\_2**
  + **Observation**: This feature has a small number of discrete values (e.g., 0, 1, 2, 3, 4), indicating it is categorical but still has the chance of being numerical like number of days/members of families since the number of unique values is too many.
* **feature\_8**
  + **Observation**: The distribution is highly imbalanced, with most values concentrated at a few specific points (e.g., 0, 1, 4).
  + **Implication**: This feature may have limited predictive power due to its imbalanced nature. We would consider analyzing its correlation with the target variable by heatmap after.
* **feature\_3**
  + **Observation**: The feature has a roughly normal distribution, with values concentrated around the center and tapering off at the edges.
  + **Implication**: This feature is well-suited for modeling and does not require significant preprocessing.
* **feature\_9**
  + **Observation**: The feature has evenly spaced discrete values, indicating it is categorical.
  + **Implication**: Encoding methods such as one-hot encoding or frequency encoding should be applied to this feature.
* **feature\_11**
  + **Observation**: The feature has a small number of discrete values (e.g., 1, 2, 3, 4, 5), indicating it is categorical.
  + **Implication**: Encoding methods such as one-hot encoding or label encoding should be applied to this feature.
* **feature\_10**
  + **Observation**: The feature has a uniform distribution across a wide range of values.
  + **Implication**: This feature is well-represented and can be used directly in the model without additional preprocessing.
* **target**
  + **Observation**: The target variable is highly skewed to the right, with a large number of smaller values and a long tail of larger values.
  + **Implication**: The skewness may require transformation (e.g., log transformation) to normalize the distribution and improve model performance.
  + The presence of outliers in the upper range should be investigated and handled appropriately.
* **Conclusion**: But after many trials of training and testing models, we realized that these features are truly numerical not any one is misclassified as categorical.

## **Missing values**

A screenshot of a black screen

Description automatically generated

* From the general view, we could see that most of missing values are in numerical features. And the number of missing values is too many, therefore, we decided to use imputation method to fill missing values instead of dropping these.

### **Numerical features**

A graph with a line and a red line

Description automatically generatedA graph with a line and a red line

Description automatically generated

A graph with a line and a red line

Description automatically generatedA graph with a red line and a purple line

Description automatically generatedA graph with a line and a red line

Description automatically generatedA graph with a line and a red line

Description automatically generated

A graph with a line and a red line

Description automatically generatedA graph with a line and a red line

Description automatically generated

* **Features with Strong Correlation:**
  + feature\_0 shows a strong negative correlation with the target variable.
  + feature\_8 shows a strong positive correlation with the target variable, especially for higher values.
  + feature\_10 shows a moderate negative correlation with the target variable.
* **Features with Weak Correlation:**
  + Features like feature\_2, feature\_9, feature\_11, and Age show relatively constant average target values, indicating weak or no correlation.
* **Impact of Missing Values (NaN):**
  + Missing values (NaN) consistently show significantly higher or lower average target values compared to non-NaN values.
  + This suggests that missingness itself may carry important information and should be treated carefully (e.g., imputation, creating a separate category, or using missingness as a feature).

### **Categorical features**

A graph of a bar chart

Description automatically generated with medium confidenceA graph of a number of people

Description automatically generated with medium confidence

A graph of a bar chart

Description automatically generated with medium confidence

* **Weak Correlation with Target:**
  + For all three features (feature\_13, feature\_4, and feature\_1), the non-NaN categories show very similar average target values, indicating weak or no correlation with the target variable.
* **Impact of Missing Values (NaN):**
  + Missing values (NaN) consistently exhibit significantly higher average target values compared to non-NaN categories across all three features.
  + This suggests that missingness itself may carry important information and should be treated carefully during preprocessing.

## **Data type of each column**

|  |  |
| --- | --- |
| **Column** | **Data type** |
| Age | Float64 |
| Gender | Object |
| Feature\_0 | Float64 |
| Feature\_1 | Object |
| Feature\_2 | Float64 |
| Feature\_3 | Object |
| Feature\_4 | Object |
| Feature\_5 | Float64 |
| Feature\_6 | Object |
| Feature\_7 | Object |
| Feature\_8 | Float64 |
| Feature\_9 | Float64 |
| Feature\_10 | Float64 |
| Feature\_11 | Float64 |
| Feature\_12 | Object |
| Feature\_13 | Object |
| Feature\_14 | Object |
| Feature\_15 | Object |
| Feature\_16 | Object |

* So, we need to convert all types that have **int64** to **int8** and if object features get **one-hot encoded** we also need to convert **float64** to **float32**.

## **Correlation heatmap**

A blue and red squares

Description automatically generated

* It is quite surprising that not any features that have remarkable correlations with target.
* **Low Correlation Between Features:**
  + Most features have very low or no correlation with each other (values close to 0.00), indicating minimal multicollinearity.
  + This is beneficial for the model, as it reduces redundancy among features.
* **Notable Correlations:**
  + feature\_0 has the strongest correlation with the target variable (-0.03), making it the most important feature in this dataset.
  + Other features like feature\_8 (0.05) show weak correlations but may still contribute to the model.

# **Phase 1**

## **Data Preprocessing**

### **Feature engineering**

* From the EDA above, we knew that **feature**\_12 is **time-related feature**. And after trying removing this feature, we realized that the RMLSE is much higher when we used it.
* But instead of leaving the whole feature, we extracted day, month, year and then create periodic features like sin and cos. Because models can’t understand the cyclical relationship of months and days. For example:
  + Time-related features such as month, day, and day of the week are cyclical in nature:
    - After December (month 12), the cycle resets to January (month 1).
    - After Sunday (day 6), the cycle resets to Monday (day 0).
  + If these features are represented as integers (e.g., 1 for January, 12 for December), the model may not understand their cyclical relationship. For example:
    - The difference between December (12) and January (1) is 11 numerically, but in reality, they are adjacent in the cycle.
* We used **sin** and **cos** for days and months because these map cyclical features onto a unit circle preserving their periodicity, for example:
  + For a 12-month cycle:
    - January (1) and December (12) are adjacent in the cycle.
    - Using sine and cosine:
      * January: sin(2π \* (1 - 1) / 12) = 0, cos(2π \* (1 - 1) / 12) = 1
      * December: sin(2π \* (12 - 1) / 12) ≈ 0, cos(2π \* (12 - 1) / 12) ≈ 1
    - The sin and cos values for January and December are close, preserving their cyclical relationship.
* Using both **sin** and **cos** ensures that the cyclical feature is represented in two dimensions, capturing both the angle and magnitude on the unit circle.
* This two-dimensional representation avoids ambiguity:
  + For example, sin(π/2) = 1 and sin(3π/2) = -1 are distinct points on the unit circle, but their sine values alone would not differentiate them. Adding cosine resolves this ambiguity.
* We did not add any new features since we tried but the results were worse.

### **Handling missing values**

* Since we already knew from the EDA that the number of missing values in numerical features is high and fill them by mean value would make the distribution gathered in center, which will make the values biased.
* Also, for the categorical missing values, we knew that the missing values have a strong relationship with target.
* Therefore, imputation methods that we chose:
  + Numerical features: Fill by **median** values.
  + Categorical features: Fill **Unknown** values.

### **Encoding values**

* 3 main encoding methods that used:
  + **One hot encoding**: Gender, feature\_1, feature\_4, feature\_6, feature\_16.
  + **Label encoding:** feature\_14.
  + **Ordinal encoding:** feature\_3, feature\_7, feature\_13, feature\_15. Since the values of these features appeared to relate to levels, for example:
    - **Feature\_3:** High School < Bachelor's < Master's < PhD.
    - **Feature\_7:** Basic < Comprehensive < Premium.
    - **Feature\_13:** Poor < Average < Good.
    - **Feature\_15:** Rarely < Monthly < Weekly < Daily.

### **Log-transformation target**

A graph of a graph

Description automatically generated with medium confidence

* Since we already knew that the metric was evaluated is RMLSE, therefore using **log-transformation** on target is the best:
  + Alignment with the Metric: RMSLE measures errors in the log space (log(1+y)). By taking the mean in the log space and exponentiating back, you’re finding the value that minimizes the squared logarithmic differences, which is exactly what RMSLE evaluates.
  + Relative Errors: RMSLE cares about relative errors (e.g., a 10% error is the same whether the true value is 100 or 10,000). The log transformation converts relative errors into additive differences, and the mean in this space corresponds to the geometric mean in the original space, which aligns with RMSLE’s focus on relative errors.

### **Retyping**

* From the EDA above, we knew that values range is not too large, if the features are int the values would not be over 256 and float would not get out of range [**-3.4e+38, 3.4e+38].**
* Therefore, we downcast types:
  + Int64 or Int32 -> Int8.
  + Float64 -> Float32.
* By doing this, we would decrease the training time and also the storage unit while training.

## **Modelling**

In this problem, we used only tree-based models, because:

* From the EDA, we recognized that all **features** had **no direct or linear relationships** with target, which could be seen from the heatmap all correlation values were nearly 0.
* Also, these models are **less sensitive to feature scaling** and from above preprocessing, we did a lot of processing of features.
* Tree models are **effective in handling missing values**, although we already imputed missing values, if the imputation was imperfect, these models still provide an additional layer of robustness.
* Last but not least, the **time training** for tree models is much **lower** than other models and the **performance** is much **better**, which can be seen from the evaluation part below.

### **Trick for training large datasets**

We knew that the size of the training set is 1.2M rows, which was pretty large, and in order to train on entire training set but still time-efficient, we chose [distributed training](https://docs.ultralytics.com/guides/model-training-tips/#pre-trained-weights) (9:25 AM 17/04/2025 Other Techniques to Consider When Handling a Large Dataset ).

* Using parallel\_backend with loky ensures efficient and scalable parallel processing for training the RandomForest model. It speeds up training, manages system resources effectively, and provides flexibility for switching backends if needed. This is particularly important for computationally expensive tasks like training ensemble models on large datasets.
  + **Speed Up Training with Parallel Processing**
    - RandomForestRegressor is an ensemble method that trains multiple decision trees independently.
    - By default, RandomForestRegressor supports parallel processing through the n\_jobs parameter, which allows it to use multiple CPU cores to train the trees simultaneously.
    - Using parallel\_backend('loky') ensures that the parallel processing is explicitly managed, potentially improving efficiency and resource utilization.
  + **Efficient Resource Management**
    - The loky backend is designed for process-based parallelism, which is more robust for CPU-intensive tasks like training decision trees.
    - It ensures that the training process does not interfere with other tasks running on the system by managing memory and CPU usage efficiently.

## **Training process**

* Instead of using regular train test split, in this problem we used **KFold** with n\_folds = 5 to **cross-validate** entire training dataset. Because we wanted to let the models learn as much as possible and to validate in the most comprehensive way possible.
* The params that were used for each model, had been hyper-tuned using [**Optuna**](https://viblo.asia/p/tu-dong-dieu-chinh-sieu-tham-so-voi-optuna-va-pytorch-RQqKLe00Z7z)(9:58 AM)**:** 
  + Optuna uses advanced optimization algorithms like Tree-structured Parzen Estimator (TPE) and Bayesian Optimization to efficiently explore the hyperparameter space.
  + Unlike traditional grid search or random search, Optuna focuses on promising regions of the search space, reducing the number of trials required to find the best parameters.
* After evaluating models and retrieving the most powerful models, we combined them by using **Voting Regressor** to combine. Since each model had strengths and drawbacks by taking average predicted values of each model can help utilize the characteristics of them.

## **Random Forest**

After hyper-tuning and trials with many params, we chose the most appropriate and optimal ones:

* **n\_estimators**: More trees can help stabilize predictions, reducing the likelihood of extreme under- or over-predictions, which lowers RMSLE.
* **max\_depth**: Deeper trees might better capture non-linear relationships in the log-transformed target, but too much depth can lead to overfitting, causing large errors on the validation set.
* **min\_samples\_split**: Increasing this can prevent the model from overfitting to outliers in the training data, which might otherwise lead to large RMSLE errors on the validation set.
* **max\_features**: A smaller max\_features increases randomness, which can prevent the model from overfitting to noisy features, potentially reducing RMSLE.

## **Decision Tree**

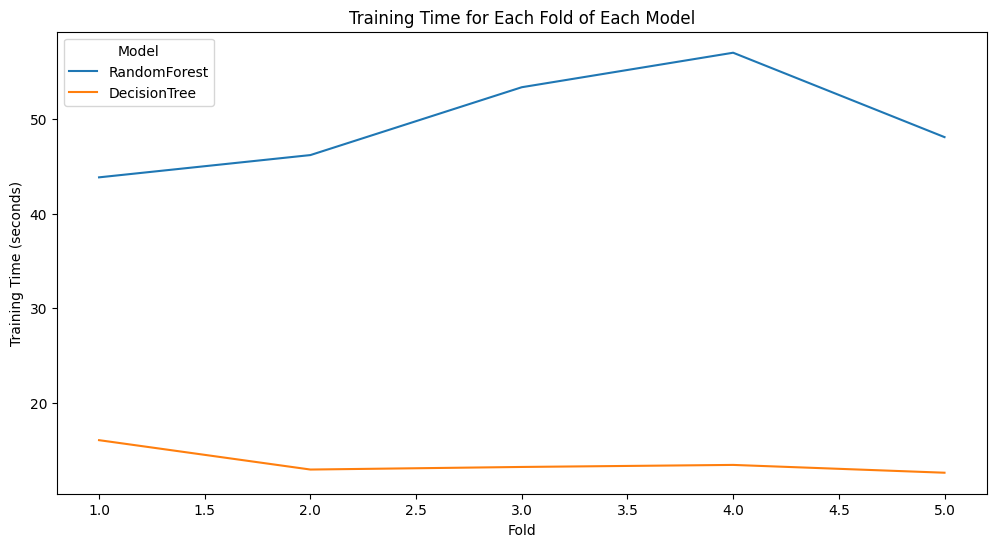
We do the same as Random Forest but the number of params is more basic:

* **max\_depth**: A deeper tree might better capture non-linear relationships in the log-transformed target, but too much depth can lead to overfitting, causing large errors on the validation set.
* **min\_samples\_split**: Increasing this can prevent the tree from overfitting to outliers or noise in the training data, which might otherwise lead to large RMSLE errors on the validation set.
* **min\_samples\_leaf**: A higher value smooths predictions by ensuring each leaf’s prediction is based on more samples, reducing the likelihood of extreme predictions that could inflate RMSLE.

## **Evaluation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Average RMSLE** | **Average RMSE** | **Average R^2** | **Average training time** |
| **Decision Tree** | 1.049114 | 921.879663 | -0.135853 | 13.635407 |
| **Random Forest** | 1.060308 | 931.084165 | -0.158647 | 49.704083 |

#### **Training time**



**Comparison Between Models**

* **Performance:**
  + Random Forest takes significantly longer to train compared to Decision Tree due to its ensemble nature (training multiple trees).
  + Decision Tree is much faster, making it more suitable for scenarios where computational resources or time are limited.
* **Stability:**
  + Decision Tree shows consistent training times across folds, while Random Forest exhibits more variation, likely due to the ensemble's sensitivity to data distribution in each fold.

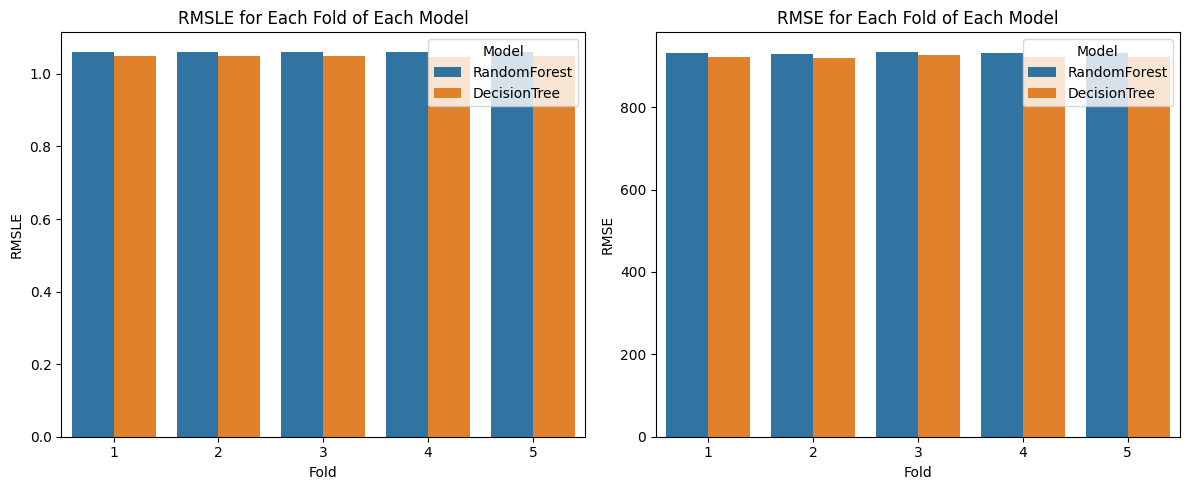
**Key Insights**

* **Random Forest:**
  + The increasing trend in training time across folds suggests that some folds may have more complex data distributions, requiring more computational effort.
  + The slight decrease in Fold 5 could indicate a smaller or less complex training subset for that fold.
* **Decision Tree:**
  + The stable training time highlights its computational efficiency and insensitivity to variations in the training subsets.

**Trade-Off:**

* While Random Forest is slower, it surprisingly provided worse predictive performance due to its ensemble nature.
* Decision Tree is faster but may maintain high accuracy, especially for complex datasets.

#### **RMSLE and RMSE**

****

**RMSLE:**

* Both models have high RMSLE values, indicating poor performance in predicting logarithmic-scaled values.
* Decision Tree slightly outperforms Random Forest in RMSLE, but the improvement is not substantial.

**RMSE:**

* The same to RMSLE both models had quite poor performances

**Key observation**

* **Consistency Across Folds:**
  + Both models show consistent performance across all 5 folds, with minimal variation in RMSLE and RMSE values.
  + This suggests that the models are stable but consistently underperforming.
* **Poor Predictive Performance:**
  + Both RMSLE and RMSE values are high for both models, indicating poor predictive performance.
  + The high RMSLE values suggest that the models struggle with logarithmic-scaled predictions, while the high RMSE values indicate difficulty in predicting the actual scale of the target variable.
* **Decision Tree vs. Random Forest:**
  + Decision Tree slightly outperforms Random Forest in both RMSLE and RMSE, but the difference is negligible.
  + Random Forest, being an ensemble method, is expected to perform better with proper hyperparameter tuning but turns out to contradict the theory.

#### **R2**

A graph of a line

Description automatically generated with medium confidence

**Consistently Negative R2 Values:**

* Both models fail to explain the variance in the target variable, as indicated by the negative R2 values.
* This suggests that the models are underperforming and may not be capturing the underlying patterns in the data.

**Fold 3 as the Worst-Performing Fold:**

* Both models perform the worst in Fold 3, indicating that this fold may have a challenging data distribution, such as outliers or a lack of representativeness.

**Decision Tree vs. Random Forest:**

* Decision Tree slightly outperforms Random Forest in terms of R2 across all folds, but the difference is not substantial.

#### **Limitation Analysis: Erroneous Cases and Their Properties**

**Erroneous Cases:**

* High RMSLE and RMSE: Models struggle with small target values (high relative errors) and fail to capture the overall target distribution, leading to large absolute errors.
* Negative R²: Models perform worse than a mean-based prediction, failing to explain the variance in the target variable.

**Properties of Erroneous Cases:**

* Skewed Target Distribution: The highly skewed target variable makes it challenging to predict extreme values, even with log transformation.
* Outliers: Outliers in the target variable influence predictions, causing overfitting in Decision Tree and instability in Random Forest.
* Feature-Target Relationships: Models fail to capture complex, non-linear relationships, especially with limited Decision Tree depth.
* Feature Importance: Noise or low-predictive-power features degrade model performance.

**Reasons for Inaccuracy:**

* Model Complexity: Decision Tree's limited depth restricts its ability to capture patterns, while Random Forest struggles with skewed distributions and outliers due to reliance on individual trees.

#### **Pros and cons**

|  |  |  |
| --- | --- | --- |
| **Approach** | **Pros** | **Cons** |
| **RandomForest** | Robust, captures non-linear relationships, supports parallelism | High training time, sensitive to hyperparameters, struggles with skewed data |
| **DecisionTree** | Simple, fast training, interpretable | Overfits easily, limited predictive power, struggles with complex patterns |
| **Voting Regressor** | Combines strengths of models, improves stability | Increased complexity, dependent on base models, higher computational cost |
| **Feature Engineering** | Captures complex relationships, handles missing values, adds temporal patterns | Increases complexity, limited effectiveness, time-consuming |
| **Preprocessing** | Reduces skewness, handles categorical and missing data effectively | Imputation noise, encoding complexity |
| **Metrics** | RMSLE penalizes relative errors, RMSE captures absolute errors | Sensitive to small/large target values, fold-specific challenges |

# **Phase 2**

## **Data Preprocessing**

### **Feature engineering**

We did the same to phase 1, but because we were in phase of Gradient Boosting models (which can handle the best at exploring non-linear features), therefore, adding **time features,** **interaction features**, **polynomial features, …** helped increasing the result a lot.

* **Time features**:
  + **Group**:
    - Calculated as (year - 2020) \* 48 + month \* 4 + day // 7, creates a hierarchical temporal grouping. It combines year, month, and a weekly bin (day // 7) into a single feature.
    - This feature aggregates time into a single numerical value, which can help the model identify trends over larger time periods.
* **Interaction features:**
  + Feature\_0\_feature\_5
  + Feature\_0\_Age
  + Group\_Age
    - Since from the heatmap we knew that feature\_0, feature\_5 and feature\_10 have the strongest correlation with target. Therefore, we tried many combinations and concluded that these are the strongest and the most effective features.
    - For **Group\_Age,** after adding Group feature then using model to plot the feature importance, surprisingly when we tested this feature turned out to be the top tier feature that we would see on the feature importance of feature selection phase.
* **Polynomial features:**
  + Feature\_0\_squared
  + Age\_squared
    - As we said before, Age and Feature\_0 have the strongest correlation with target, and after trials, these ones are the strongest features.
* **Nan features:**
  + We knew that missing values have the important relationships with target and Gradient Boosting models like LightGBM and XGBoost can handle that.
  + Therefore, creating nan columns can preserve information about missingness.
* **Frequency features:**
  + As we tested independently, we surprisingly discovered that these features somehow have great importance.
  + Also, maybe the number of unique values of each categorical feature is almost the same. So maybe this kind of encoding can help providing information for models.

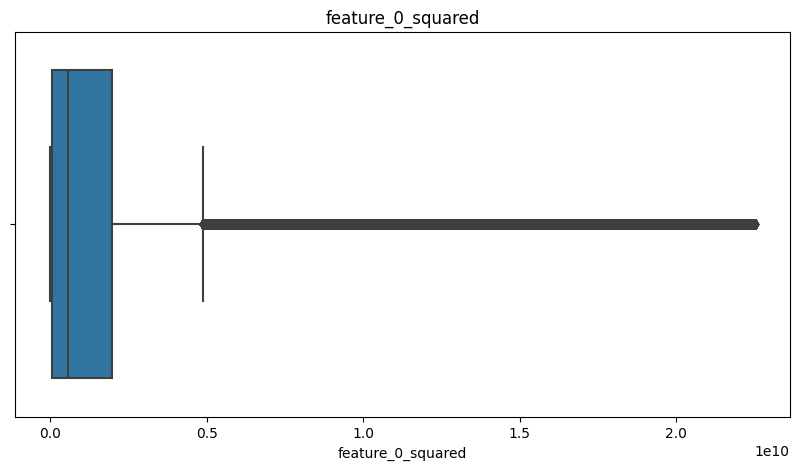
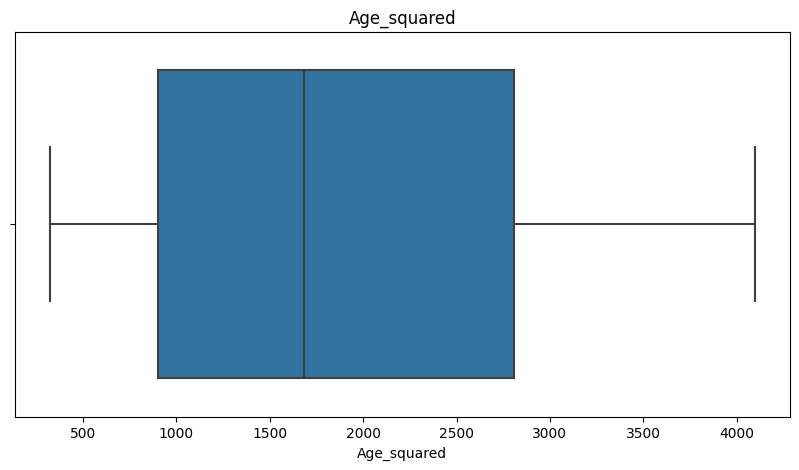
### **Handling missing values**

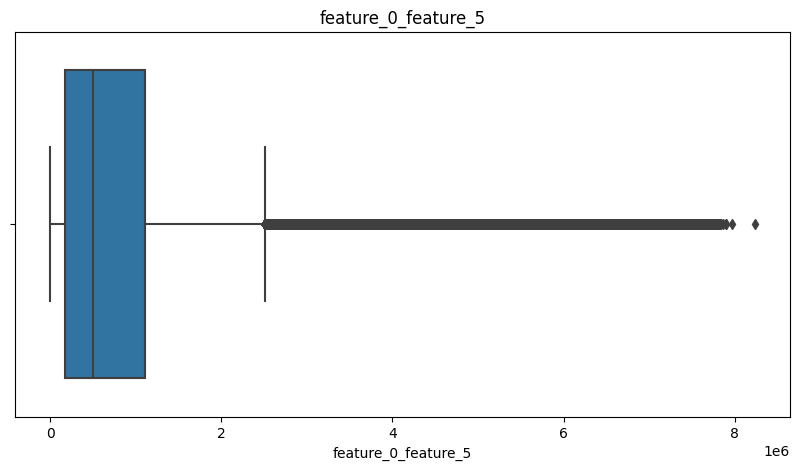
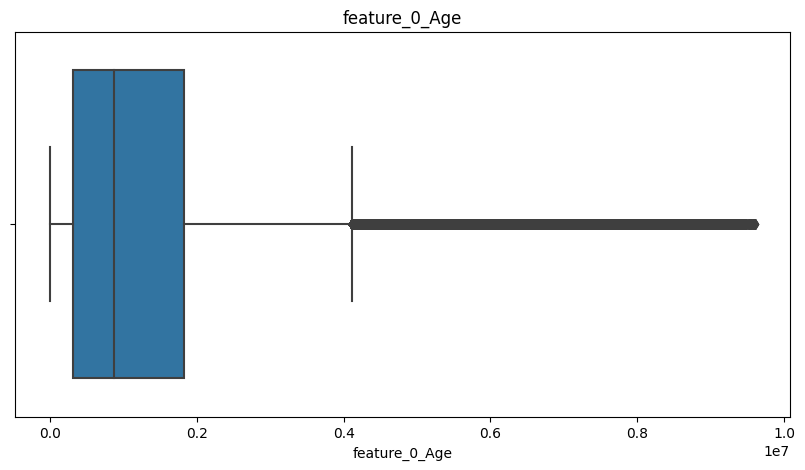
We did the same to Phase 1.

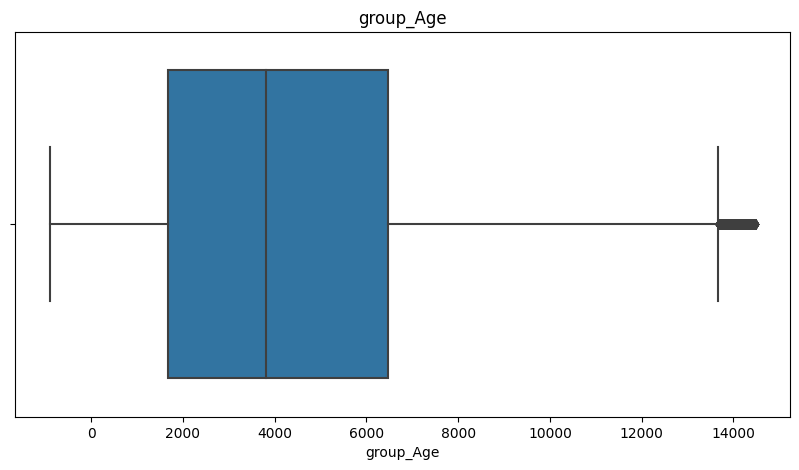
### **Encoding values**

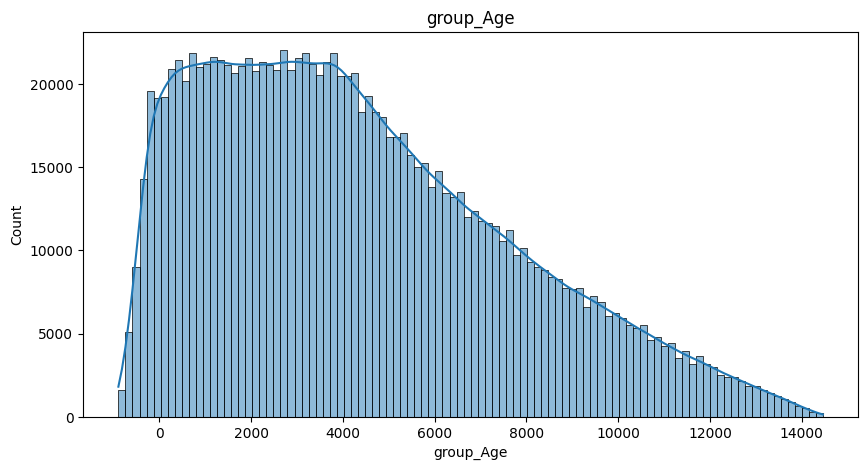
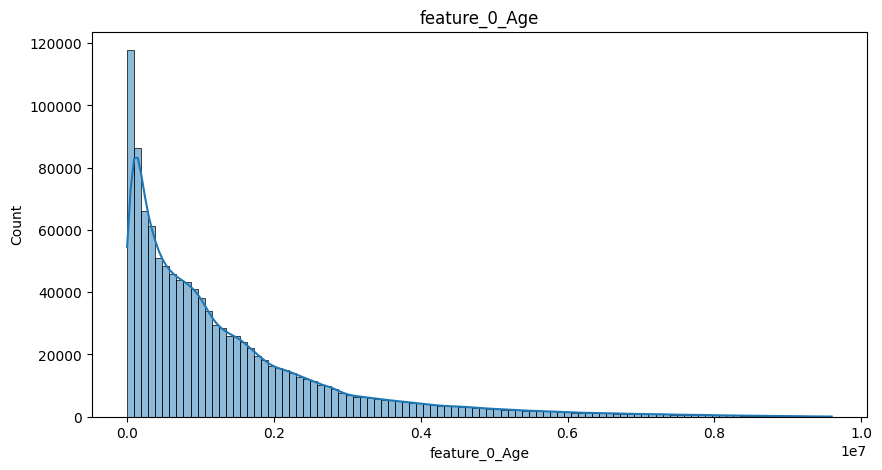
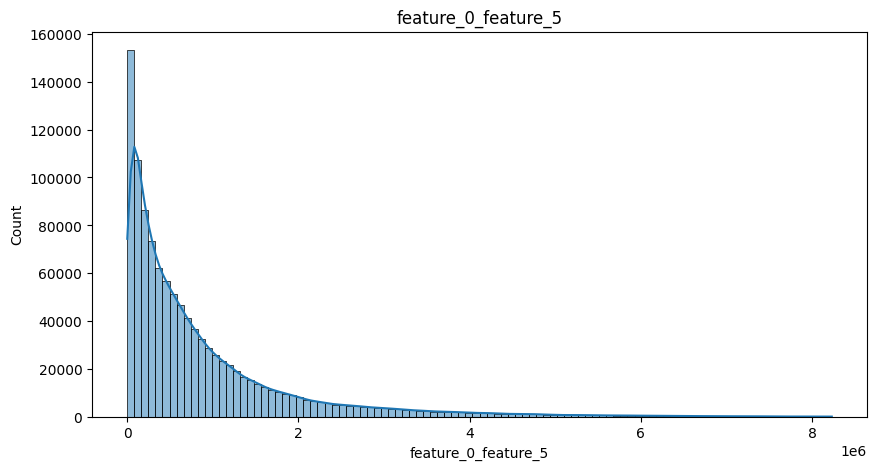
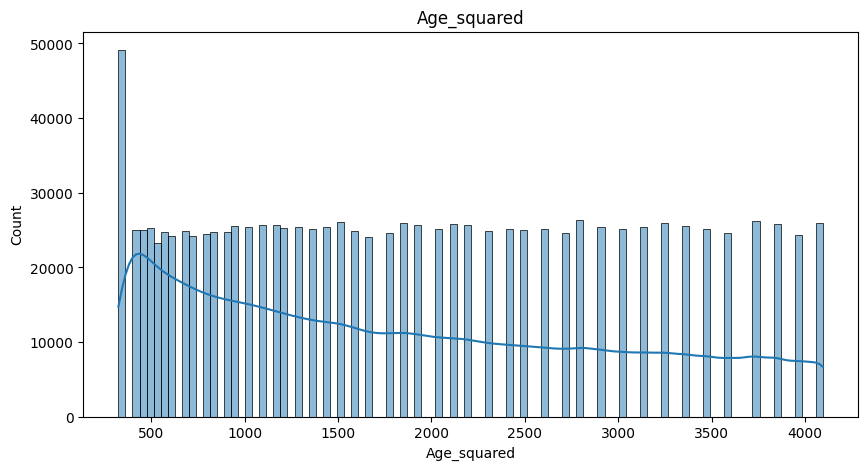
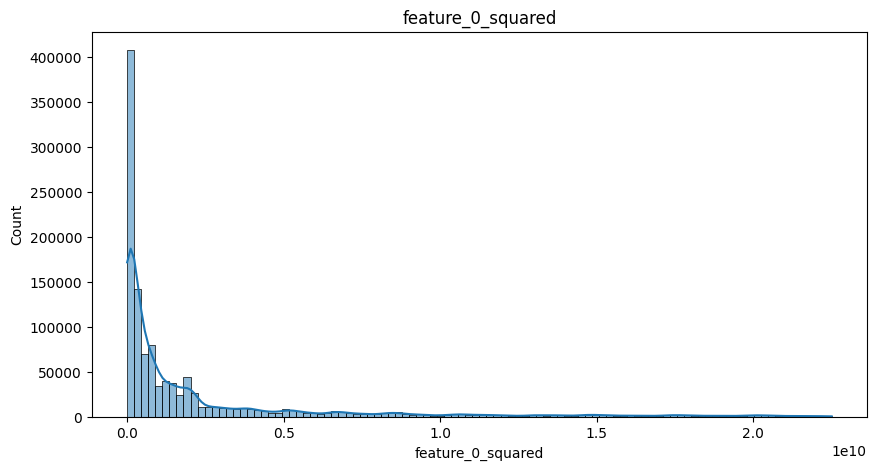
The same process in Phase 1 was applied.

### **Log transformation on target and new features**





* As we could see that these new features have not only outliers but also skewness. But to maintain the variety of target features, we decided not to drop outliers.
* Instead, we only log-transformed these features to reduce skewness.

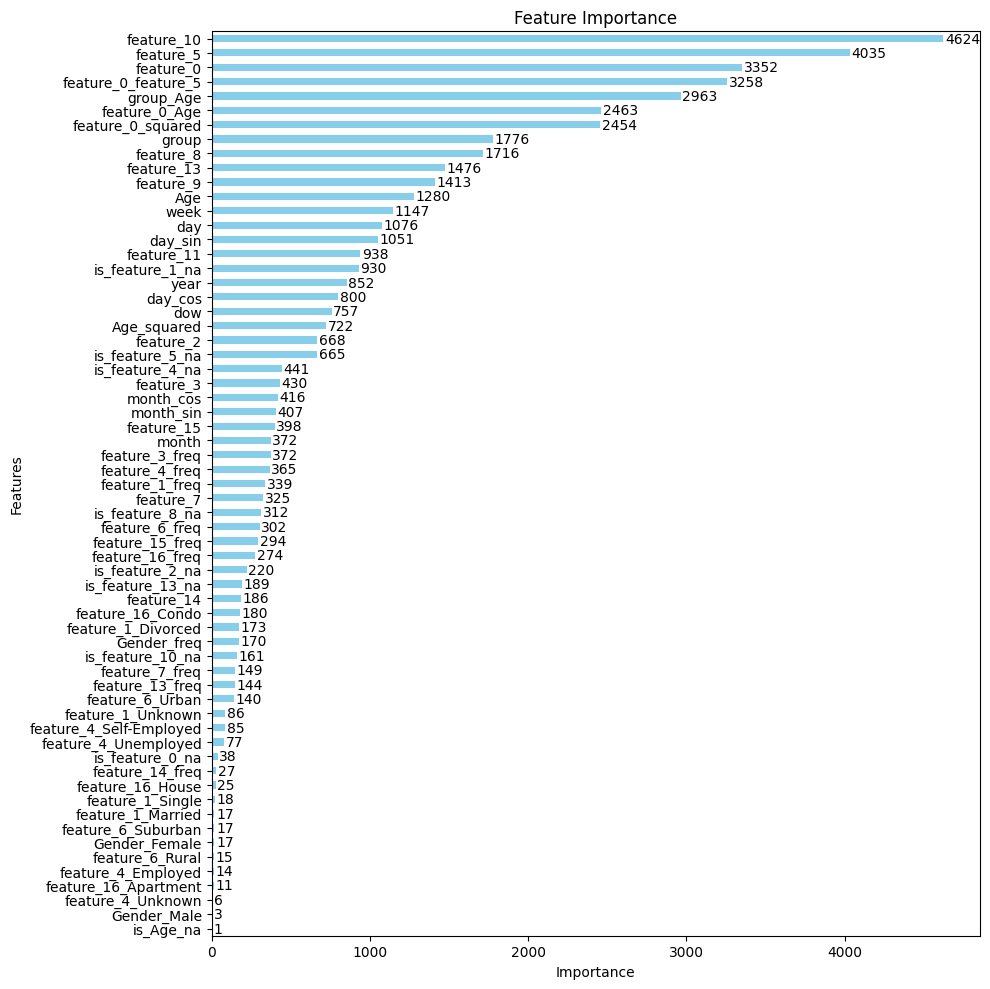
### **Retyping**

The same process in Phase 1 was applied.

### **Feature selection**

From the experience of Phase 1, we realized trying to put too many features into models can cause noises and made models learn the wrong pattern.

Therefore, we used LightGBM (the strongest model in this problem) to check feature importance:



**Interaction Features:** Interaction features like feature\_0\_feature\_5 and group\_Age are among the top contributors, validating the decision to create these features.

**Polynomial Features:** Squared terms like feature\_0\_squared are highly important, highlighting the value of capturing non-linear relationships.

**Frequency Encoding:** Frequency-encoded features (e.g., feature\_4\_freq, feature\_1\_freq) have moderate importance, showing that encoding category prevalence adds predictive value.

**Missing Value Indicators:** NaN indicator columns (e.g., is\_feature\_11\_na) are moderately important, suggesting that missingness itself carries information.

* **Surprisingly, we can see that most of categorical features have low importance. After many trials, we decided that threshold 200 is the optimal one for us to eliminate low important features.**

## **Modelling**

For this phase, we used all Boosting models: XGBoost, LightGBM and Catboost. From the general theory, we knew that these models could capture non-linear relationships of features.

* Instead of choosing one model, we decided to keep all models since each model has its own strengths and weaknesses.
* By combining all models using Voting Regressor as the way we did in Phase 1.

### **Training process**

For the same process of Phase 1, we still do the same process as Phase 1. However, there are some different points during hyper-tuning process:

* For all models, we used **early stopping** that is provided for each model to improve time training performance.
* The details of how early stopping initialized will be described below.

### **XGBoost**

**Brief introduction:**

* A powerful and efficient machine learning algorithm based on the gradient boosting framework.
* XGBoost builds an ensemble of decision trees sequentially, where each tree corrects the errors of the previous ones.
* It minimizes a loss function (e.g., RMSE for regression) using gradient descent.

**Hyper-tuning:**

* To use **early\_stopping in XGB**, we gotta use **DMatrix** and **Train** of XGB because:
  + They offer greater flexibility for early stopping via callbacks.
  + **DMatrix** is more efficient for repeated training (e.g., in cross-validation).
  + They support advanced features like categorical handling and custom evaluation.
  + Early stopping reduces training time for poor hyperparametersets, which is critical during tuning.

[**Params**](https://xgboost.readthedocs.io/en/stable/parameter.html) **(4:22 PM 18/04/2025):**

* **learning\_rate:** A smaller learning rate allows the model to learn more gradually, reducing the risk of overshooting the optimal solution and stabilizing predictions, which can lower RMSLE. However, if too small, it might require more trees, potentially leading to underfitting and higher RMSLE.
* **subsample**: Using a fraction of the data per tree introduces randomness, preventing the model from overfitting to the training data, which can reduce RMSLE. Too low a value might cause underfitting, increasing RMSLE.
* **lambda**: Increasing L2 regularization penalizes large weights, reducing overfitting and preventing large prediction errors on the validation set, which can lower RMSLE. However, excessive regularization might lead to underfitting, increasing RMSLE.
* **alpha**: Increasing L1 regularization encourages sparsity in the weights, which can prevent overfitting to noisy features, potentially reducing RMSLE. Too high a value might oversimplify the model, increasing RMSLE.
* **gamma**: A higher value makes the model more conservative by requiring a larger loss reduction to make a split, reducing overfitting and potentially lowering RMSLE. If too high, it might prevent the model from capturing important patterns, increasing RMSLE.

### **LightGBM**

**Brief introduction:**

* A high-performance, distributed, and efficient gradient boosting framework
* Designed for speed and efficiency, making it particularly well-suited for large datasets and high-dimensional data. Because it is based on decision tree algorithms.

**Hyper-tuning:**

* The same to XGBoost, to use **early\_stopping,** we used **train and Dataset:**
  + It optimizes data storage and processing, reducing memory and computation time during hyperparameter tuning with multiple folds and trials.
  + It is required by **lgb.train(),** the native API used for training, which offers fine-grained control over the training process.
  + It supports early stopping by defining validation sets, allowing the model to stop training when validation RMSE plateaus, saving time and reducing overfitting to minimize RMSLE.

[**Params**](https://lightgbm.readthedocs.io/en/latest/Parameters.html) **(4:25 PM 18/04/2025)**:

* **learning\_rate**: A smaller learning rate allows the model to learn more gradually, reducing the risk of overshooting the optimal solution and stabilizing predictions, which can lower RMSLE. However, if too small, it might require more trees, potentially leading to underfitting and higher RMSLE.
* **subsample**: Using a fraction of the data per tree introduces randomness, preventing the model from overfitting to the training data, which can reduce RMSLE. Too low a value might cause underfitting, increasing RMSLE. (Note: In your case, use\_goss=True, so this parameter is ignored in favor of GOSS sampling with top\_rate and other\_rate.)
* **reg\_lambda**: Increasing L2 regularization penalizes large weights, reducing overfitting and preventing large prediction errors on the validation set, which can lower RMSLE. However, excessive regularization might lead to underfitting, increasing RMSLE.
* **reg\_alpha**: Increasing L1 regularization encourages sparsity in the weights, which can prevent overfitting to noisy features, potentially reducing RMSLE. Too high a value might oversimplify the model, increasing RMSLE.
* **min\_child\_weight**: A higher value makes the model more conservative by requiring more samples to make a split, reducing overfitting to outliers or noisy data and potentially lowering RMSLE. If too high, it might prevent the model from capturing important patterns, increasing RMSLE.
* **max\_depth:** Limits the depth of each tree, controlling the model's complexity and reducing the risk of overfitting, which can lower RMSLE. If the value is too low, the model might underfit and fail to capture important patterns, increasing RMSLE.
* **num\_leaves**: Determines the maximum number of leaf nodes in each tree. A higher value allows the model to capture more complex patterns, potentially reducing RMSLE. However, too many leaves can lead to overfitting, especially on noisy data, increasing RMSLE.
* **colsample\_bytree**: Controls the fraction of features (columns) used for each tree, introducing randomness and reducing overfitting, which can lower RMSLE. If the value is too low, the model might miss important features, leading to underfitting and higher RMSLE.
* **n\_estimators**: Specifies the number of trees in the model. A higher number of trees allows the model to learn more patterns, potentially reducing RMSLE. However, too many trees can lead to overfitting and increased training time.
* **use\_goss**: Gradient-based One-Side Sampling (GOSS) focuses on the most important samples (with large gradients) while down-sampling less important ones, improving efficiency and reducing overfitting.
  + When use\_goss=True, subsample is ignored, and top\_rate and other\_rate control the sampling strategy.
* **top\_rate (GOSS Parameter):** Specifies the proportion of samples with the largest gradients to retain. A higher value ensures that the most important samples are prioritized, reducing RMSLE. If the value is too high, it might lead to overfitting.
* **other\_rate (GOSS Parameter):** Specifies the proportion of samples with smaller gradients to retain. A balanced value ensures that the model does not ignore less important samples, reducing RMSLE. If the value is too low, the model might underfit.
* **objective:** Specifies the loss function to optimize. For regression tasks, regression or regression\_l2 is used to minimize RMSE, which aligns with the goal of reducing RMSLE.
* **force\_col\_wise:** Forces LightGBM to use column-wise data storage, which can improve memory efficiency and speed for datasets with many features.

### **Catboost**

**Brief introduction**

* A gradient boosting framework, designed to handle categorical features efficiently.
* It is based on decision trees and is widely used for classification, regression, and ranking tasks.
* Particularly known for its ability to process categorical data without extensive preprocessing, such as one-hot encoding.

**Preprocessing:**

* Unlike XGBoost or LBGM, CatBoost require categorical features to apply smart encoding (Ordered encoding or Target based encoding), help enhance the performance of the model, categorical features:
  + is\_feature\_5\_na
  + is\_feature\_8\_na
  + is\_feature\_1\_na
  + is\_feature\_4\_na
  + feature\_1\_freq
  + feature\_4\_freq
  + feature\_6\_freq
  + feature\_16\_freq
  + feature\_3\_freq
  + feature\_15\_freq
  + is\_feature\_2\_na
* After initializing categorical features, we would added into fit when training.

**Hyper-tuning:**

* Different from 2 above models that used **Dataset** of their own to optimize time training, memory efficiency and early stopping, instead it has **Pool**, which:
  + Efficient handling of categorical features, reduces memory overhead, speeds up training, and maintains consistency across folds.
  + Simplifies the code and leverages CatBoost's advanced capabilities, making it an essential component for optimizing model performance during hyperparameter tuning.
  + Optimizes data storage and preprocessing, reducing computational overhead during hyperparameter tuning with multiple folds and trials.
  + Supports **early stopping** by providing a validation set (val\_pool) for monitoring RMSE, saving training time and preventing overfitting to minimize RMSLE.

[**Params**](https://catboost.ai/docs/en/references/training-parameters/common) **(4:42 PM 18/04/2025):**

* **Depth:** This controls the maximum depth of each tree (how many split a tree can make), Deeper trees can capture more complex patterns but risk overfitting.
* **Iterations:** Number of boosting rounds, more iterations usually increase accuracy but also increase training time. Therefore number of iterations should be balanced between performance and time.
* **Learning rate:** Controls how much the model updates at each iteration. Lower values = slower learning but more accurate (if iterations is high enough).
* **Verbose:** Catboost pring info after a number of iterations.
* **l2\_leaf\_reg:** L2 regularization term on leaf weights to reduce overfitting. The more value the higher regularization.
* **random\_strength:** Adds randomness to the scoring of splits to reduce overfitting. Higher values introduce more noise, which can help generalization.
* **bagging\_temperature:** Controls the strength of the bagging, higher values = more randomness when sampling data for each tree.
* **border\_count:** The number of splits (borders) CatBoost uses to bucket numerical features.

### **Evaluation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Average RMSLE** | **Average RMSE** | **Average R2** | **Average Training Time** |
| LightGBM | 1.045484 | 922.640655 | -0.137728 | 83.117989 |
| XGBoost | 1.046666 | 923.124509 | -0.138922 | 20.261922 |
| CatBoost | 1.047914 | 925.032019 | -0.143634 | 42.377354 |

#### **Training time**

A graph of a training

Description automatically generated

**Comparison**

* **XGBoost**:
  + Fastest training time (~20 seconds).
  + Ideal for scenarios where computational efficiency is a priority.
* **LightGBM**:
  + Longest training time (~80–85 seconds).
  + Suitable for scenarios where accuracy is more important than speed.
* **CatBoost**:
  + Balanced training time (~40 seconds).
  + A good middle ground between speed and accuracy, especially for datasets with categorical features.

**Key Insights**

* **Consistency**:
  + All three models show consistent training times across folds, indicating stable performance.
* **Trade-offs:**
  + XGBoost is the fastest but may sacrifice some accuracy.
  + LightGBM is the slowest but often achieves the best accuracy.
  + CatBoost balances speed and accuracy, making it versatile for various use cases.

#### **RMSLE and RMSE**

A close-up of a graph

Description automatically generated

**RMSLE**:

* All three models (XGBoost, LightGBM, and CatBoost) have high RMSLE values, indicating poor performance in predicting logarithmic-scaled values.
* LightGBM slightly outperforms CatBoost and XGBoost in RMSLE, but the improvement is minimal.

**RMSE**:

* Similar to RMSLE, all three models exhibit high RMSE values, reflecting poor performance in predicting the actual scale of the target variable.
* LightGBM and CatBoost perform slightly better than XGBoost in RMSE, but the differences are negligible.

**Key Observations:**

* **Consistency Across Folds:**
  + All three models show consistent performance across all 5 folds, with minimal variation in RMSLE and RMSE values.
  + This suggests that the models are stable but consistently underperforming.
* **XGBoost vs. LightGBM vs. CatBoost:**
  + LightGBM slightly outperforms CatBoost and XGBoost in both RMSLE and RMSE, but the differences are minimal.
  + XGBoost, despite being the fastest model, lags slightly behind in accuracy.
  + CatBoost performs on par with LightGBM but has slightly higher training times compared to XGBoost.

#### **R2**

A graph of a line graph

Description automatically generated with medium confidence

**Consistency in Negative R2:**

* The same pattern observed in Phase 1 is evident here, with all three models (XGBoost, LightGBM, and CatBoost) consistently showing negative R² values across all folds.
* This indicates that models can hardly explain the variance in the target variable effectively.

**Fold 3 as the Worst-Performing Fold:**

* The same pattern in Phase 1 was also seen here.
* The inability of the models to generalize well to this fold highlights potential issues with the data distribution or feature-target relationships in this subset.

**Model Comparison**

* **LightGBM:**
  + Performs the best among the three models, with the least negative R2 values across all folds.
  + This suggests that LightGBM is slightly better at capturing patterns in the data, though the improvement is minimal.
* **XGBoost:**
  + Performs moderately well, with R2 values slightly worse than LightGBM but better than CatBoost.
* **CatBoost:**
  + Has the most negative R2 values, indicating the poorest performance in explaining the variance in the target variable.

#### **Limitation Analysis: Erroneous Cases and Their Properties**

**Properties of Erroneous Cases:**

* **Skewed Target Distribution:** Models struggle with extreme values, leading to high RMSLE and RMSE despite log-transformation**.**
* **Outliers:** Significant outliers distort predictions; removing them worsened generalization.
* **Feature-Target Relationships:** Non-linear and interaction patterns are not fully captured or enough, resulting in poor predictions.
* **Missing Values:** Imputation strategies may introduce noise, leading to inaccurate predictions for missing data.
* **Temporal Features:** Models struggle to generalize temporal patterns, especially for rare or unseen intervals.

**Reasons for Inaccuracy:**

* **Model Complexity**: Ensemble models struggle with skewed distributions and outliers, failing to generalize for extreme cases.
* **Limited Data Representation**: Rare categories and extreme values are underrepresented, leading to high errors.
* **Overfitting to Noise**: Noise in low-importance features reduces generalization despite feature selection.
* **Metric Sensitivity:** RMSLE and RMSE highlight struggles with both small and extreme target values.
* **Fold-Specific Challenges:** Fold 3 contains challenging data (e.g., outliers, rare cases), leading to consistently poor performance.

#### **Pros and Cons**

|  |  |  |
| --- | --- | --- |
| **Approach** | **Pros** | **Cons** |
| **XGBoost** | Efficient, robust to overfitting, customizable, supports parallelism. | Sensitive to hyperparameters, struggles with skewed data and outliers. |
| **LightGBM** | Fast, scalable, memory-efficient, supports categorical features. | Can overfit, struggles with outliers, less interpretable. |
| **CatBoost** | Handles categorical features natively, robust to overfitting, easy to use. | Slower training, resource-intensive, struggles with skewed data. |
| **Feature Engineering** | Captures complex relationships, handles missing values, adds temporal patterns. | Increases complexity, limited effectiveness, time-consuming. |
| **Metrics** | RMSLE penalizes relative errors, RMSE captures absolute errors. | Sensitive to small/large target values, fold-specific challenges. |