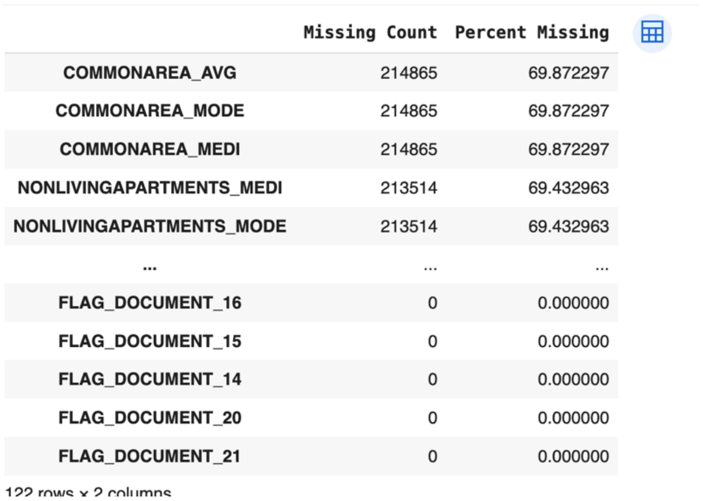
Gabriella Rich 10/22/2025
Assignment 2 Applied Machine Learning

I used the Home Credit Default Risk Dataset to predict whether an applicant will repay a loan, using a wide range of application, credit, and behavioral data. This dataset was used in a Kaggle competition, and the main goal was to unlock the potential of the data using machine learning models to make predictions on clients' repayment ability in order to prevent the population of being taken advantage of by lenders. I only used the application_train.csv file for this project where one row represents one loan from the data sample. The train.csv has the main table with our target. On first glance of the data, I noticed that there was a ton of missing data in every column which is to be expected since it is a natural part of data collection and obviously the collectors could not collect every single person's data. There are 122 columns in this dataset and 307,511 rows so clearly it is a massive data set and there is a lot of missing data. I knew at first glance that the volume of the data would increase the time complexity of my models that I am hoping to train and that it would be tough to extract meaningful insights from this dataset. Additionally, the data is filled with all different data types. Some of the data is numerical in nature, other columns are integer data, and there is also a lot of categorical variables. Categorical variables need to be encoded because the models I am training cannot process the non-numerical data directly. I thought for a while on how to my missing data in the data set and moved on to data preprocessing steps.

In the data preprocessing steps, I had to analyze my data further. I had the idea to figure out which columns could be dropped based off the percentage of how much total data was missing/"NaNs". I designated a arbitrary threshold that if the data had over 60% dropped it could definitely be dropped as it would not be useful to my models even if I dealt with the missing values. First, I identified all columns in my data set that had missing values and then asked Chat

GPT to output a summary table showing how many and what percentage of the dataset's values are missing in each column of the Data Frame.

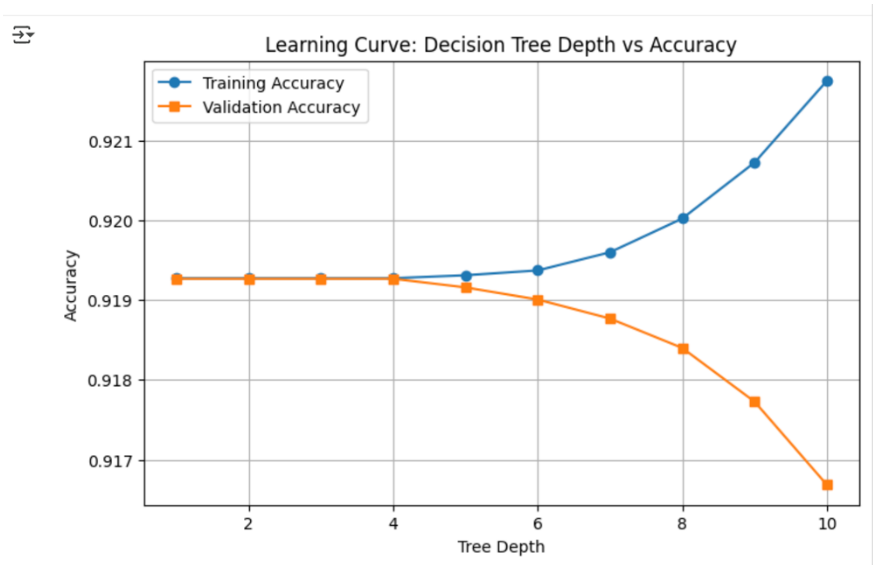


	Missing Count	Percent Missing
COMMONAREA_AVG	214865	69.872297
COMMONAREA_MODE	214865	69.872297
COMMONAREA_MEDI	214865	69.872297
NONLIVINGAPARTMENTS_MEDI	213514	69.432963
NONLIVINGAPARTMENTS_MODE	213514	69.432963
...	...	...
FLAG_DOCUMENT_16	0	0.000000
FLAG_DOCUMENT_15	0	0.000000
FLAG_DOCUMENT_14	0	0.000000
FLAG_DOCUMENT_20	0	0.000000
FLAG_DOCUMENT_21	0	0.000000

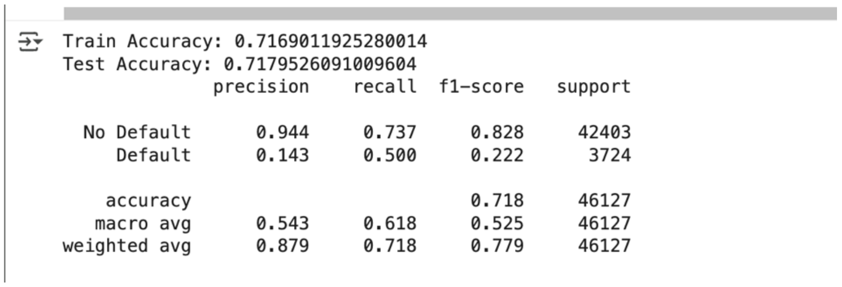
122 rows × 2 columns

Then I could begin dealing with it. After dropping the missing columns, I imputed the missing values depending on their data type. I filled in gaps with 0 or performed imputation of their column medians and filled in the categorical data with "Unknown" . Additionally, I separated the numeric and categorical columns from one another. Chat GPT helped me apply the right imputation method and gave me code to use SimpleImputer to have the numerical columns imputed by their median values while categorical columns were imputed by the most frequent value in said column. Then I split the newly cleaned credit dataset into training, validation, and test sets in order to develop the models. After the train/test/split, I performed the task of one-hot encoding and imputation using Professor Luo's code from class as a skeleton. Originally, I did one-hot encoding column transformation prior to train/test/split but I ran into problems when training my baseline decision tree. Chat GPT advised that one-hot encoding transformation is done after the split because "fitting the preprocessor (like imputation or encoding) before splitting would let the model 'see' patterns from validation/test data — a form of data leakage that inflates performance metrics. By fitting only on the training set, the preprocessing simulates how the model would handle unseen real-world data". After my data was properly encoded and the feature names and X_train shape were the same, I was able to begin training the models on this dataset.

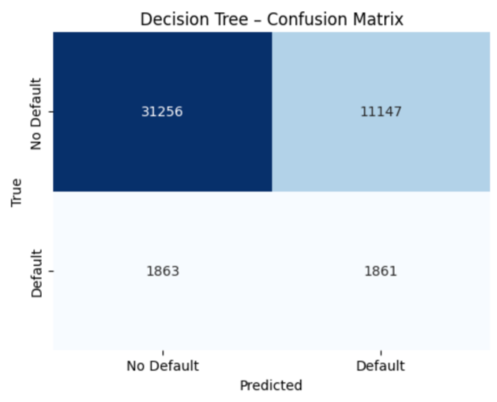
I started with a baseline model: The Decision Tree. I needed to tune the depth, minimum samples, and regularization parameters of the Decision Tree. I decided to work with two different baseline models to see how pre-pruning parameters affected the model's overall performance. The original baseline fit perfectly with the training data which 1 at first thought was a good thing but it is actually not. Having a training accuracy of 1 means that there was a high likelihood of overfitting, meaning that the model just memorized the training data, and it lacks the ability to accurately predict outcomes on new data. For baseline2, I added ccp_alpha which is a cost complexity pruning to control tree size as well as adding more minimum samples to the nodes to split, as well as limiting the depth of the tree to 3. There was less overfitting as the training accuracy was now .91 and the test accuracy was also .91 so it was improved. Additionally, Chat GPT helped me plot the learning curve to show how training and validation scores for this baseline model. The x-axis shows the tree depth, and the y-axis is the accuracy which measures the performance of the model. Below is what the graph of the training accuracy and the validation accuracy looks like for different values of depth.

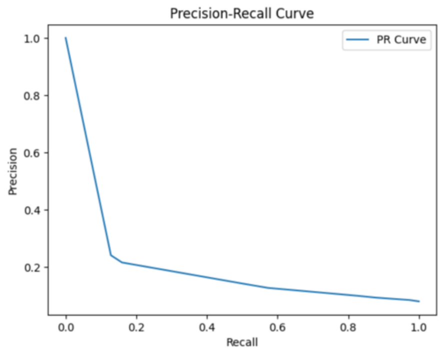


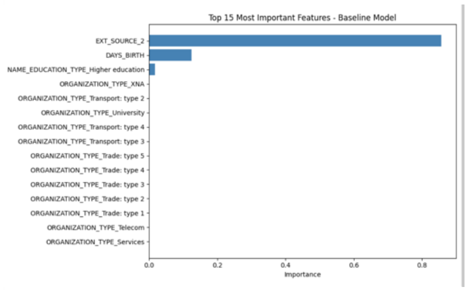
This graph shows that low depth leads to underfitting also known as high bias and the model is too simple to capture the relational data. Around 3-5 is the optimal range of depth for this baseline decision tree. It balances bias and variance and is able to capture the structure in the data without overfitting. For larger depths, you can see that the training accuracy increases while the validation accuracy decreases which means that the model is too complex and there are risks of overfitting. Now using what I found, I was able to create a baseline decision tree model and chose max_depth of 3.



```
Train Accuracy: 0.7169011925280014
Test Accuracy: 0.7179526091009604
              precision    recall  f1-score   support

  No Default      0.944     0.737     0.828     42403
     Default      0.143     0.500     0.222      3724

    accuracy                          0.718     46127
   macro avg      0.543     0.618     0.525     46127
weighted avg      0.879     0.718     0.779     46127
```

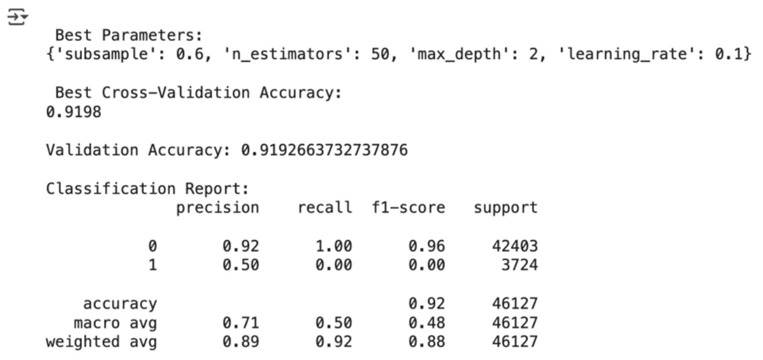




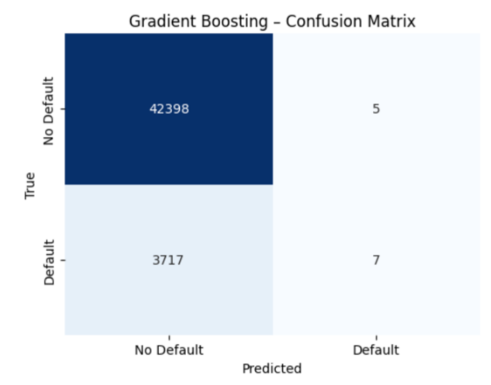


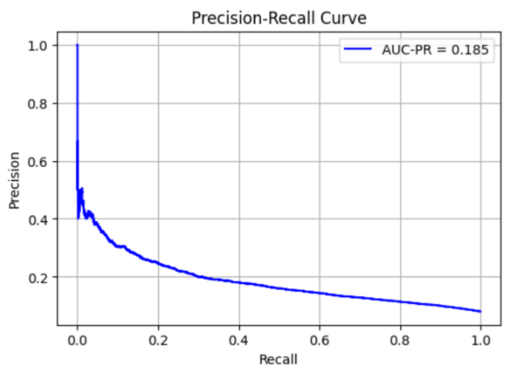
Through these metrics and plots for the decision tree model, the model has a 72% accuracy but has low decision for the default category of the data. The model leans towards predicting "Default" and can find potential risks in the data. As recall increases, you can see that precision decreases by a lot which could mean that the dataset is imbalanced because there aren't a lot of "Default's" being found. The Feature importance bar chart finds that EXT_SOURCE_2 is the strongest predictor. Since EXT_SOURCE_2 examines a person/client's external credit risk, the model can see how low EXT_SOURCE_2 correlates to higher default risk thus making it the strongest predictor in the case of our Decision Tree Model.

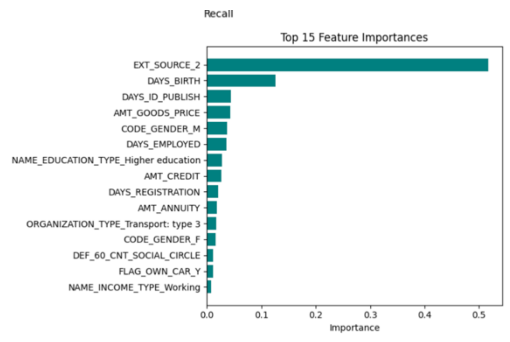
Then I moved to building both a Gradient Boost model as well as an XG Boost model. Boosting trains models sequentially so that each new model focuses on the errors of the previous ones. Gradient Boosting minimizes the loss function by adding weak learners in the direction of the negative gradient. XG Boost is an enhanced version of Gradient Boost that adds improvements to the Gradient Boost model in terms of speed and regularization capabilities. Chat GPT helped me build a Gradient Boost model that includes Randomized Search CV for hyperparameter tuning of the model. It has 2-fold stratified cross-validation which splits our data into 2 parts and ensures each fold preserves the class distributions. Originally, I tried running this code on the entire training dataset and it took over 20 minutes to run. This turned into a big problem for me, so chat GPT advised to take the first 40,000 samples of the data because the original data was just too large. I am aware of that there are certain pitfalls to cutting up the data into smaller sections and it risks missing the best parameters needed. It was a risk I was willing to take.



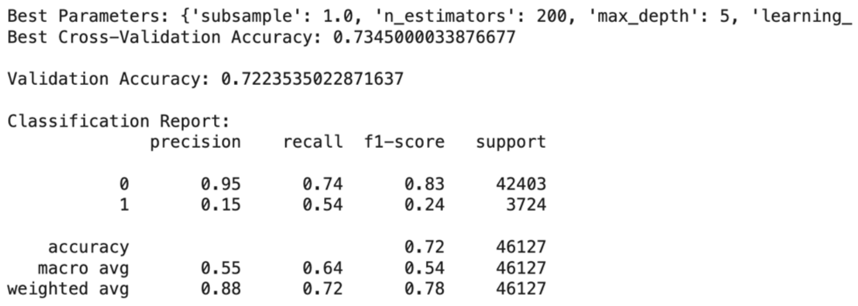
```
 Best Parameters:
{'subsample': 0.6, 'n_estimators': 50, 'max_depth': 2, 'learning_rate': 0.1}

 Best Cross-Validation Accuracy:
0.9198

Validation Accuracy: 0.9192663732737876

Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     42403
           1       0.50      0.00      0.00      3724

    accuracy                           0.92     46127
   macro avg       0.71      0.50      0.48     46127
weighted avg       0.89      0.92      0.88     46127
```





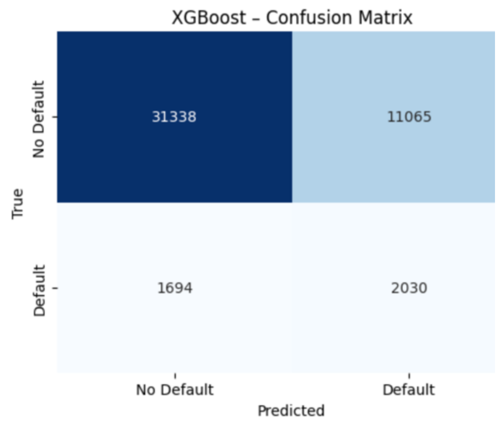


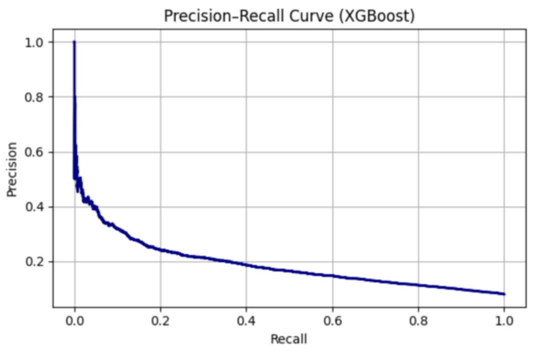
Using the best-found parameters from the Randomized Search CV, I was able to build the best gradient boosted model for the subsection of the data I took. The data has high accuracy scores and for high precision scores for Default(0) but has poor precision for Default(1) which means it almost never predicts Default (which very well could be the case based on how rare Default was). There is very poor recall for the minority class and this model has high bias (Underfitting the data due to small subsection). This model struggles with the Default group and has very weak precision. This model relies heavily on predictors like EXT_SOURCE_2 as well as DAYS_BIRTH which is the age of the borrower (trends in younger borrowers' default more than older borrowers). Once again, I repeated this process using Randomized Search CV on my XG Boost, but I learned from my mistakes made in the Gradient Boosted Model and balanced the weights of the classes for XG Boost.

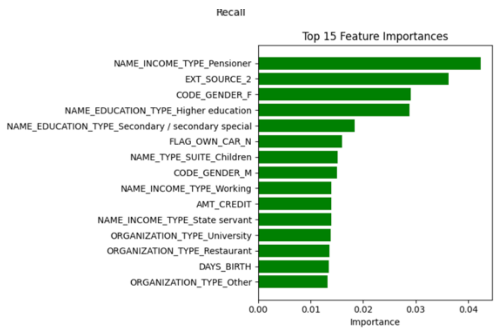


```
Best Parameters: {'subsample': 1.0, 'n_estimators': 200, 'max_depth': 5, 'learning_
Best Cross-Validation Accuracy: 0.7345000033876677

Validation Accuracy: 0.7223535022871637

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.74      0.83     42403
           1       0.15      0.54      0.24      3724

    accuracy                           0.72     46127
   macro avg       0.55      0.64      0.54     46127
weighted avg       0.88      0.72      0.78     46127
```

```
AUCPR:     0.1890
```

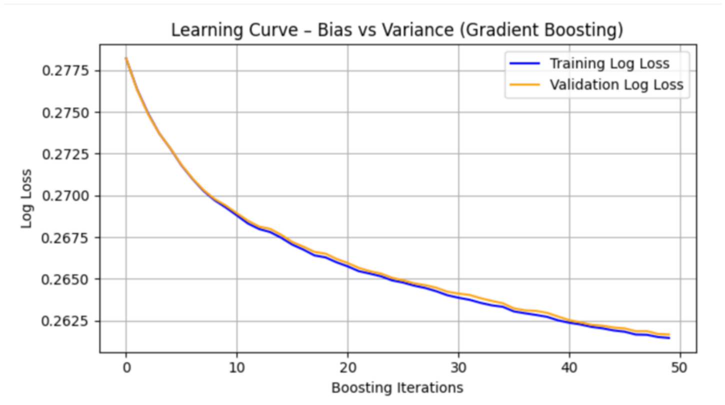


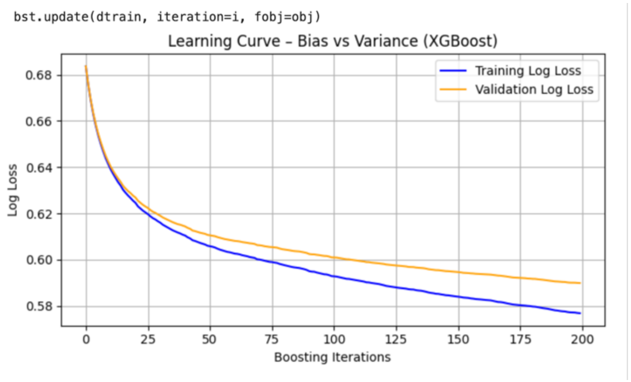




There is a drop in the accuracy for XGBoost but this is actually a good thing. This helped me learn from my mistakes and the Gradient Boost model's mistakes. XGBoost model learned to detect the default amounts to improve recall and fairness across the classes. The Recall for XG Boost now correctly identifies over of the actual “Default” class. Since Precision on “Default” is low this means it is still trapping a lot of false positives which is a downside of this model. The Precision-Recall curve is smoother which indicates that XG Boost handles the imbalanced dataset better. Feature Importance Bar Chart for XG Boost Model reflects that the top features are NAME_INCOME_TYPE_Pensioner, EXT_SOURCE_2 and CODE_GENDER_F. Feature importance is redistributed across multiple variables unlike for gradient boosting and even our baseline Decision Tree. There is less reliance on one singular predictor which is a mistake reflected before.

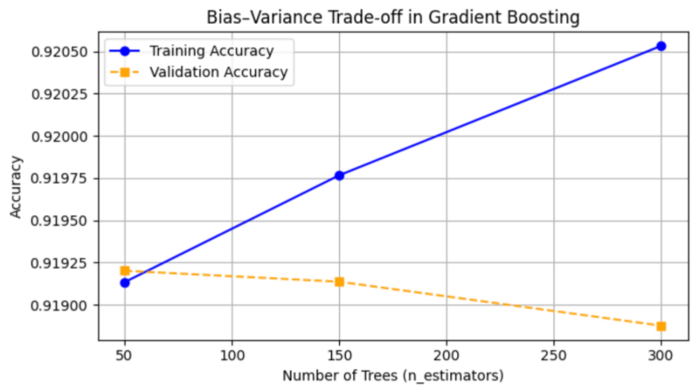
In comparing the Gradient Boosted model and the XG Boost (balanced class weights) model, I was able to analyze Training vs Validation Loss over Boosting Iterations, Effect of Learning Rate, and Bias-Variance Trade-Off.



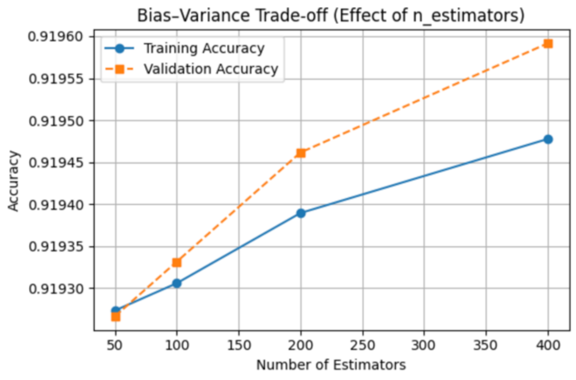


```
bst.update(dtrain, iteration=i, fobj=obj)
```

The Gradient Boost curve shows a simple model with low variance and high bias while the XG Boost curve reflects a well-balanced bias-variance tradeoff. The model takes on a bit more variance to achieve learning performance and higher recall on difficult classes.



```
bst.update(dtrain, iteration=i, fobj=obj)
```



The left graph represents the Bias-Variance Trade-Off for the Gradient Boosted Model while the right graph represents the Bias-Variance Trade-Off for XG Boost. Both the of the Training accuracy increase which shows that they learn more as it becomes more and more complex but the validation accuracy for gradient boosting slowly declines while the accuracy for XG Boost increases which reflects that XG Boost keeps improving while Gradient Boost overfits the data. The best trade-off point for the Gradient Boost is around 100-150 trees while more than 300 trees can be used for the XG Boost. This shows that XG Boost can retain its stability at higher capacities compared to Gradient Boost.

Given the fact that I wanted to learn and reflect on the mistakes made from Gradient Boost to show how different XG Boost is on balanced class weights, it leads to differing conclusions about the cost and interpretability of the models built. XG Boost is much more efficient on larger data sets while gradient boost is slower because it sequentially builds trees. In practice, XG Boost is faster and has more scalable features, but Gradient Boost is easier to interpret because of the shallowness of the trees and fewer interactions. Additionally, feature importance improves predictions because it gives a measure to how much each feature in our data set contributes to reducing error which leads us to understand why the model is predicting if a creditor is going to default and why exactly it is predicting in one direction or another. Boosting improves generalization because it reduces bias by iterating through all the trees and correcting itself while also controlling the variance by regularization and hyperparameter fine tuning to weed out the weak learners.

I used Chat GPT for help in modifying Professor Luo's demo code, asking it to generate its own code for me to use for data preprocessing and building the models up. It also helped me in debugging certain areas such as modifying the data set to get a shorter run time as well as balancing the weights of the classes for the XG Boost so I can see a difference in my predictors for those models.