During the data exploration, as reviewed in class, there were significantly more 5s than any other score. My initial thought was that I had to ensure that I sampled the data equally so that each score had equal numbers when training the model. However, I found that when I tried to sample equally from the dataset, the accuracy rate in predicting 5s decreased significantly up to 22%, and the accuracy in predicting 2,3,4 increased by only 10%. Since, score 5 is the most frequent score, it decreases the accuracy significantly more than the increase in 2,3,4. Next, I wanted to see if there were some empty cells, I should clean before proceeding with feature extraction and if there are some in the text and summary columns. Lastly, I wanted to see if there are reviewers who have more comments than others and if there are reviewers whose patterns of commenting can be easily caught by the model, which is why I created the graph titled 'Top 25 reviewers with The Most Reviews'. From there, I discovered that there are users who have more than 2000 reviews and there are users who have more than 1000 reviews. I also tried to see the least number of reviews a person would give, and I discovered that the least number of reviews left by a user is 5. This is important because, from this, I would get an idea about the distribution of the least to the highest amount of review given by a person.

After that, I extracted a sample of 10,000 rows to test the effects of features and classifier models faster by using a random set of smaller data from my initial dataset. Next, I cleaned the dataset and the sample. I achieved this by removing rows that do not have both text and summary because they do not contribute anything to the model, as we do not have an apparent reason for why that score was given. In addition, I also put the word 'empty' to that summary and empty text. This ensures that there are no NaN. I did not drop them because it allows the model to recognize that some information is still absent rather than exclude potentially useful data entirely. This might help capture the model's understanding of incomplete or brief reviews if that's a factor in score prediction. Lastly, I combined the text field and the summary field rather than treating them as separate entities, as it helps to create a more prosperous and complete representation of the dataset, especially for those with summaries or reviews. It also expedites TF-IDF in the later steps because we apply the vectorizer to only one rather than two columns.

Proceeding with adding features, I inserted a feature called 'Number_of_Reviews_for_Movie'; This column was to count the number of reviews for each movie. This feature enables the model to judge the movie's popularity as the more popular the movies are, the more reviews there are, therefore allowing the model to judge the consistency of the reviews. I also added another feature to calculate the number of reviews per user, ''Number_of_Reviews_by_User', and this feature enables the model to consider if certain rating patterns correlate with certain users. Next, I labeled the top 25 reviewed products and the top 25 users who left the most comments as 1. I did so because labeling them helps the model recognize these products and potentially interpret their scores differently than for niche or less-reviewed items, which may have more polarized ratings.

Similarly, labeling the top 25 most active reviewers provides insight into any biases or rating habits among high-frequency reviewers, who may have more consistent scoring patterns than infrequent reviewers. This feature allows the model to account for possible influences from these users, who could skew average ratings based on their preferences or scoring tendencies. I also made a feature to label the top 25 least reviewed products, and the top 25 users who left the least comments were -1. I did so to help the model recognize products and users with limited reviews that may lead to more polarized or unreliable ratings, allowing the model to adjust predictions based on these potentially less stable data points. I also added a helpfulness feature, which divides the helpfulness numerator and denominator provided in the initial starter code. This ratio enables the model to see how accurate the review is based on how helpful it is. Stemming from this, I inserted a feature that calculates the deviation between each helpfulness compared to the mean, highlighting reviews that stand out as uniquely helpful or unhelpful, which could correlate with more extreme ratings. I also added a feature that calculates the average deviation of helpfulness per user, representing how each user's helpfulness differs from the product's average, indicating whether a user's reviews are generally more or less helpful than the product average, allowing the model to assess user reliability or consistency in providing constructive feedback.

I then added features based on the combined text and summary, where people convey the most emotions and, therefore, where the model would see the most pattern. We first do sentiment analysis using Textblob, which produces a float from -1 to 1, with a score close to -1 being close to purely negative and a score close to 1 being close to purely positive. This step is crucial because, from this feature, the model can detect the emotional tone of the text and summary and predict the score better. I then added a feature that contains the combined text and summary length. This feature is essential because there can be a tendency for a more extended review and summary to indicate the depth or level of detail in the body of the text; this feature allows the model to consider the level of detail as a factor in score prediction, enhancing its ability to interpret the sentiment and intent behind each review. I added a feature that calculates the average sentiment for each product and the deviation of each review's sentiment from this product average, as it helps the model understand whether a review aligns with the typical sentiment for the product or diverges significantly, which may indicate a particularly strong or unique opinion that could affect the review score.

From there, I added a feature that calculates exclamation, question marks, capital letter, and also negation words. Negation words is defined as 'not', 'no', 'never', 'none', "n't", 'neither', 'nor', 'without', 'hardly', 'barely', 'scarcely'. Because these elements often signal the **tone and intensity** of a review. Exclamation marks and capital letters can indicate strong emotion or emphasis, potentially aligning with extreme ratings, while question marks might suggest uncertainty or a critical tone. Negation words can shift the sentiment of a review, impacting the review's overall meaning. By capturing these nuances, the model can interpret each review's emotional weight and intent, enhancing its ability to predict scores accurately. Then, I also added a feature that calculates average exclamation, question, and negation counts per product and their deviation from product averages.

This helps the model gauge whether a review's tone is typical for that product. Compared to the product's average, reviews with unusually high or low counts of these elements may indicate an outlier in emotional intensity or sentiment, which could correspond with more extreme or unique scores. This allows the model to better interpret the tone and emphasis of individual reviews within the context of each product's general review pattern, improving predictive accuracy. Furthermore, I transformed the `Combined_Text` column into a matrix of TF-IDF features, capturing the significance of each word and two-word phrase within the reviews. With `max_features=1000`, the top 1,000 terms were retained to limit dimensionality, excluding common English stop words to focus on more meaningful terms. They enable the model to recognize and quantify word importance and patterns in the review text, which is valuable for understanding the content's impact on review scores. By incorporating this feature, the model can leverage nuanced language cues often correlating with sentiment, tone, or relevance, enhancing its predictive power.

I split data into training and testing sets using train_test_split, with 75% of the data allocated to training (X_train, Y_train) and 25% to testing (X_test, Y_test). The Score column is separated as the target variable, while the remaining columns are used as features. Setting random_state to zero ensures that the split is reproducible each time. I then proceeded to create a list of the features that I would use in training the model and, using this list, select the relevant columns in X_train, X_test, and X_submission, creating the feature sets X_train_select, X_test_select, and X_submission_select for model training, testing, and submission.

I then tested the various available models and started experimenting with **RandomForestClassifier, K-Nearest-Neighbours, and GradientBoostingClasssifer**. Amongst all 3, they respectively produced 56%, 52%, and 57%; all three scored high in predicting the score five correctly, as all three predicted 89%-90% correctly; they also predicted the score zero pr. As the **GradientBoostingClassifier** was the highest, I started to look for other possible implementations of this technique because I was frustrated by the long time it took for predictions, and tweaking its parameters did not lead to a higher score in predicting numbers that are in between. I then found **HistGradientBoostingClassifier**. It is an optimized version of gradient boosting that accelerates training on large datasets by grouping continuous feature values into discrete bins (histograms), significantly reducing memory usage and computational time while maintaining model accuracy. This model can increase accuracy to 58%, significantly reducing computational time. As the other models take up to 1 hour to run, this model takes considerably less. I then did hyperparameter tuning by trying learning_rate = [0.01, 0.05, 0.1], max_iter: [100, 200, 300], max_leaf_nodes: [15, 31, 63], min_samples_leaf: [10, 20, 30]. Then, after finding the best parameters according to these parameters, I manually tested it to see that max_iter=1000, learning_rate=0.05,max_leaf_nodes=80, min_samples_leaf=40, and random_state=42 to produce the best result, as it can produce accuracy up to 60%. I added more features, such as if and the deviation in exclamation, question marks, and negation, to help increase the model's accuracy.

References

TextBlob: https://textblob.readthedocs.io/en/dev/

HistGradientBoostingClassifier:
https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html