# Final Report (STAT 432)

By: Sam Burch (sgburch2), Nathan Nagy (nnagy2), & Nathan Costabilo (nlc3)

12-7-2023

## Project Description & Summary

In this project, we embark on an innovative journey to predict essay scores using a variety of statistical methods. Our focus is on exploring and comparing different analytical approaches, including unsupervised learning (Principal Component Analysis and K-means clustering), various regression techniques (linear regression, K-Nearest Neighbors, and Random Forest), and a classification method (XGBoost). This investigation is rooted in the intriguing premise of analyzing the writing process to forecast essay quality, aiming to bridge the gap between quantitative analysis and qualitative assessment in educational contexts.

In the unsupervised learning segment of our project, we applied Principal Component Analysis (PCA) and K-means clustering to analyze essay data. PCA was used for dimensionality reduction, highlighting the primary components that capture significant variance in the data. This method proved valuable given the high multicollinearity among predictors. K-means clustering further segmented the data into coherent groups, revealing distinct patterns and associations with essay scores. These unsupervised techniques not only reduced data complexity but also offered insights into underlying structures, enhancing our understanding of factors influencing essay quality.

In this project, Linear Regression was used to predict essay scores, aiming to understand the impact of writing style on essay quality. The model employed a 10-fold cross-validation approach for robust training and validation, with an emphasis on evaluating model performance using RMSE, reported as 0.7275. The analysis included plots of actual vs. predicted scores and residuals vs. predicted scores. These plots indicated the model's effectiveness, especially for scores in the 3.0-4.5 range, and demonstrated random distribution of errors, affirming the model's reliability. The K-Nearest Neighbors (KNN) model, a nonparametric regression approach, was used. The key parameter, k, indicating the number of neighbors considered, was optimally set at 27 after extensive tuning. The model achieved an RMSE of 0.659, indicating a good fit with the data. This model's strength lies in its smoothness and reduced variance with an increase in k, though it faces challenges with irregularities at jump points. The prediction range was predominantly between ~2 and ~5.5, aligning well with the common score range of essays.The Random Forest model in this project, chosen for its robustness in handling large datasets with complex relationships, excelled in regression tasks. The model was tuned using 5-fold cross-validation, focusing on the mtry parameter, leading to an optimal mtry value of 7. It achieved an RMSE of approximately 0.672, indicating a good fit. The scatter plot of actual vs. predicted values and the variable importance plot revealed the effectiveness of the model, with variables like 'tot_letters' and 'tot_word_count' being highly predictive. Despite computational limitations, the model performed well, indicating its suitability for datasets with collinearity issues. The XGBoost model, known for its effectiveness in handling complex datasets, was chosen for its advanced ensemble learning capabilities. The model was tuned using a predefined hyperparameter grid, focusing on boosting iterations (nrounds), tree complexity (max_depth), and learning rate (eta). The best performing model had nrounds = 100, max_depth = 3, and eta = 0.01. However, the final model demonstrated limited accuracy, with around 30.8% correct classifications, suggesting challenges in dealing with the dataset's complexity and class imbalance.

## Literature Review

The “Linking Writing Processes to Writing Quality” challenge on the website Kaggle for about two months. The challenge has seen over 22,399 entries from 1,523 competitors striving for the $55,000 prize. As a result of this, several incredibly talented and experienced statisticians and data scientists have attempted to solve this problem. As we are somewhat beginners in the field of model building, it can be incredibly valuable to look into how others solved the problem. The current best model achieved an RMSE of .572 and was created by a user named Lucky Seed. Unfortunately, the model and strategy Lucky Seed used have not been made public. Despite this, there are several other individuals and teams that created very well-fitting models that made their methods and strategies public. Below we discussed the strategies of three high scoring teams

### Best Model Publicly Available (User: AWQATAK, RMSE: .582)

The best scoring publicly shown model we found had an RMSE of .582. It was created by user AWQATAK and was most recently updated on 12/5/2023. His published Python notebook for the Kaggle challenge “Linking Writing Processes to Writing Quality” employs a technically sophisticated approach, centralizing around the use of a Light Gradient Boosting Machine (LightGBM) model, complemented by an extensive set of 165 features. This method showcases a deep engagement with feature engineering, including the calculation of burst features like ‘P-bursts’ and ‘R-bursts’ that quantify typing and editing behaviors, ratios such as the length of the final product relative to the number of keys pressed, and temporal features like keys pressed per second, along with advanced statistical measures. The choice of LightGBM, known for its efficiency with large datasets and feature-rich environments, is a strategic decision, likely involving decision trees as base learners and employing careful regularization to prevent overfitting, especially important given the dataset’s small size of less than 2500 samples. Cross-validation is utilized for robust model evaluation, ensuring the model’s performance is consistent across different data subsets. However, with such a large feature set, there’s a technical challenge in avoiding overfitting, necessitating prudent regularization and possibly model pruning. While the approach demonstrates technical adeptness, particularly in integrating and refining existing methodologies from previous kaggle submissions, it might face challenges in scaling to larger datasets or adapting to different contexts due to the specific nature of the features and model tuning. Overall, the notebook balances technical complexity with the need for a model that generalizes well to new data, yet it remains mindful of potential limitations in scalability and flexibility

### Summarized Model #2 (User: SEAN, RMSE: .613)

In the Kaggle challenge notebook “Linking Writing Processes to Writing Quality,” a technically advanced approach is implemented using a VotingRegressor. This meta-estimator combines three distinct regression models: CatBoostRegressor, XGBRegressor, and LGBMRegressor. The ensemble method leverages the individual strengths of these models, potentially leading to more accurate and robust predictions. For evaluation, the notebook employs a suite of metrics crucial in regression tasks, including Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). These metrics offer a holistic view of the model’s performance, accounting for various aspects of prediction accuracy. The training methodology is comprehensive, involving extensive data preprocessing and feature engineering. Key features derived include counts of writing activities, frequencies of keyboard and mouse events, analysis of text changes, and punctuation usage. Such features are vital in capturing the nuances of the writing process. Hyperparameter tuning, a critical step in model optimization, is performed using Optuna. This framework systematically searches for the best parameters, enhancing each base model’s performance within the VotingRegressor. While this approach is technically robust and likely to yield high-accuracy predictions, it also brings certain challenges. The computational intensity due to model ensemble and extensive data preprocessing could be a drawback. However, this methodology effectively addresses the complex task of linking writing processes to writing quality, showcasing a deep dive into the predictive modeling of textual data

**Summarized Model #3 (User: MAOK-YONGSUK, RMSE: .584)**

In the notebook provided by user MAOK-YONGSUK, the core technical approach is anchored on using LightGBM Regressor, a decision strategically made to handle the challenge's large datasets efficiently. LightGBM, known for its fast execution and relatively low memory consumption, is an ideal fit for the scale of data in this context. The notebook's methodology is deeply rooted in extensive feature engineering, a critical step to boost the model's predictive power by deriving significant features from the data. Alongside this, feature importance analysis is conducted, which is pivotal in shedding light on how different features influence the model's predictions. The model's robustness and ability to generalize well to new data are ensured through K-Fold cross-validation, a technique that helps mitigate the risk of overfitting to the training data. However, this technical approach presents certain challenges. The intensive feature engineering and reliance on LightGBM introduce the potential for overfitting, especially given LightGBM's known sensitivity to hyperparameter settings. This can make the model complex and require careful tuning and validation. Furthermore, while LightGBM is efficient, the computational demands of processing such a large dataset with a sophisticated model are still significant. Despite these challenges, the use of LightGBM in the notebook is a strategic choice, aiming to balance the need for high predictive accuracy with the practicalities of large-scale data processing

**Potential Applications to Our Project**

Incorporating the insights gained from the literature review into our project opens up several promising avenues. Firstly, model choice is pivotal: LightGBM, as employed by AWQATAK and MAOK-YONGSUK, appears particularly effective for its efficiency with large datasets, making it a strong candidate for our model. Similarly, the VotingRegressor approach by SEAN, combining CatBoostRegressor, XGBRegressor, and LGBMRegressor, offers a robust, diversified modeling strategy. This could be particularly beneficial in handling the complex nuances of our dataset. Secondly, model tuning is essential. The successful use of hyperparameter optimization techniques, like Optuna in SEAN's model, suggests that a systematic approach to fine-tuning parameters could significantly enhance our model's performance. Furthermore, the emphasis on extensive feature engineering in all models reviewed provides a clear directive for our project. Developing a rich feature set, particularly those quantifying unique aspects of our dataset, such as specific behaviors or patterns, could be crucial. Lastly, adopting comprehensive evaluation criteria, as seen in the use of RMSE, MAE, and MAPE by SEAN, will be vital. These metrics offer a multi-dimensional view of performance, helping us to understand and improve our model's accuracy and reliability. Overall, by thoughtfully adapting these strategies—balancing model complexity with efficiency, and rigorous tuning with robust evaluation—we can enhance the predictive power and generalizability of our model.

## Data Processing

After going through several of the submissions, we need to process the data. This involves reducing the number of rows to that of one per essay. Then, each essay will have one row of a variety of statistics based on the original columns from the train_logs dataset. These statistics will become the predictors for the score of the essay, so we must be thoughtful about which to include. We have four categories of predictors that we developed – Time, Activity, Punctuation, and Word Length.

Starting with time, we have total word count, which is simply the number of words in the essay. This was chosen because the more detailed someone is for an essay, we'd guess this would lead to better performance. (That is really the idea behind all the time variables.) Next, we have total & average action time, with totals for down and up time as well. Down and up time are the times at which the action is *pressed* and *released* respectively. Action time is the difference of such, so how long the user is actually taking to complete that action. We expect total action time to be important – like we expect for total word count – and average action time to possibly have some explanation of efficiency from the user. The totals for down and up time likely don't mean much but were added just in-case. Continuing, we developed total time, words per minute, and pause time. Total time is defined as the difference between the last up-time and the first down-time. This shows how long someone worked on the essay. Words per minute was simple the total word count divided by the total time, transformed into minutes. Seeing how efficient the writer is should give some indication towards score. Pause time is approximate here, the the amount of total time minus the action time; this is defined differently on Kaggle. Seeing how long people pause and think about their choices should provide some indication as well.

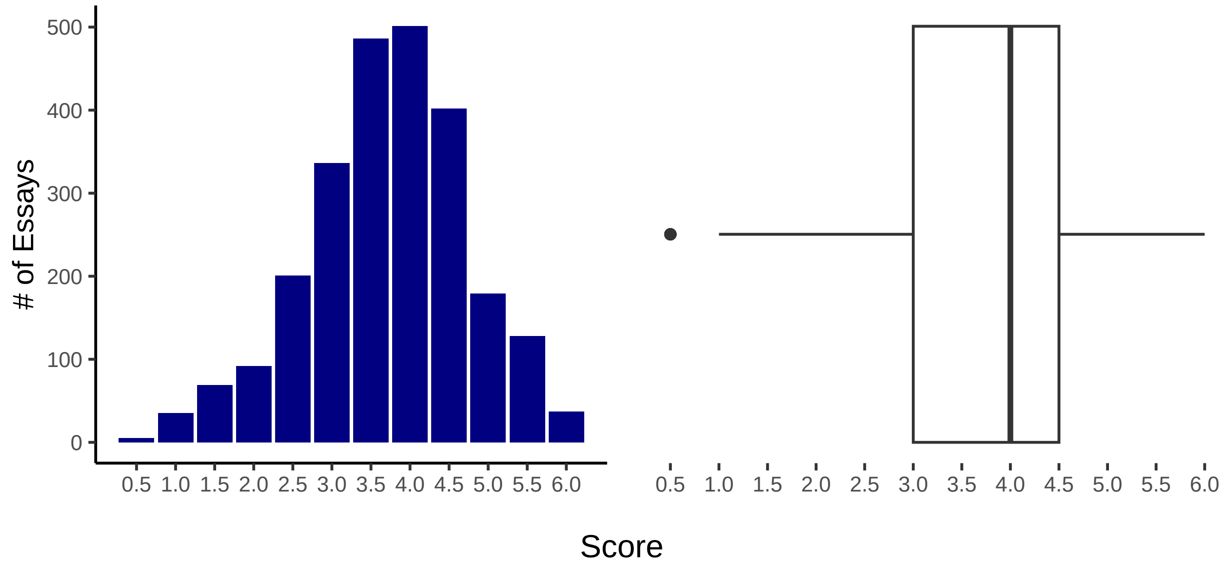
Next, we have the different activities. The six different activity types are as follows: nonproduction, input, remove/cut, paste, replace, and move from one place to another. For each, we gathered the total number of rows that had this activity type and the frequency of that number, with the total number of rows as the denominator. Nonproduction is when "the event does not alter the text in any way" (Kaggle). One who is nonproductive often may be thinking about what to do (helping their score) or they may be wasting time (hurting their score). Input is when "the event adds text to the essay" (Kaggle). Adding a lot of inputs likely will help their score. Remove/Cut is when "the event removes text from the essay" (Kaggle). Having a lot of these should show the person is constantly trying to make their essay better. The last three, paste, replace, and move, will likely show when one is editing their essay often. Like with remove/cut, this should lead to improved essay scores. Before we move on, we noticed that there were several rows in the text change column that said "NoChange" of were NA. We decided this might show some indication for similar reasons to the nonproduction variable. To calculate such, we took the total number of input rows where text change said "NoChange" or was an NA value. Then, we subtracted the number of remove/cut rows that contained the same information. Finally, we divided by the number of words in the essay to get a rate.

After activities, we created punctuation predictors. These are all rates to show how often a certain punctuation type is used. The theory here is that one who uses good punctuation and grammar should have a better score on their essay. We got these rates for the following punctuation types: dash, semi-colon, colon, comma, period, exclamation point, question mark, and bracket. The totals were defined as the sum of inputs where the text change was that punctuation type minus the sum of removes/cuts of the same type. From here, we divided the totals by the total word count for the essay. This is approximate because there are some text changes where it is a long string that includes letters and punctuation. However, these variables should be very close to the truth, with those long strings being fairly infrequent.

Last category dealt with was word length. The goal for this category is to get the necessary components for average word length. To do so, we tried to extract all the letters and divide by word length. This was not easy to do because of the structure of the train_logs data frame. However, we arrived at an approximate count for the letters. The total letters consisted of the input, paste, and replace letters added and the cut letters removed. Input letters were the total input rows with "q" being the text changed; "q" represents an anonymous letter. Cut letters were 90% of the number of characters in the text change column from the remove/cut rows. The reason we use an arbitrary 90% cutoff here was because many of these text change columns included mostly letters, but a few extra characters as well (like punctuation). We also rounded the number of these letters so it is a whole number. For paste letters, we did the exact same thing expect they were from when the activity was paste – obviously. For replace letters, the syntax was slightly different for the text change column. The string for such included an arrow separating what was there before (old) to what was there after (new). Thus, we simply used the number of characters from the new input minus the number of characters from the old. With the four sets of letters we calculated, we now have the number for the total letters. Hence, we can take the total letters and divide by the word length to get the (approximate) average word length for the essay. We would think the larger the average length of words, the more complex the words are. This should lead to a better score. We will also keep the four letters and total letters variables.

### Summary of Data

Our new data frame consisted of 2471 rows with 37 columns – for the id, 35 predictors, and score. The average score is 3.71, with a minimum of 0.5 and maximum of 6. To get a better idea of the score distribution, let us look at a couple plots.

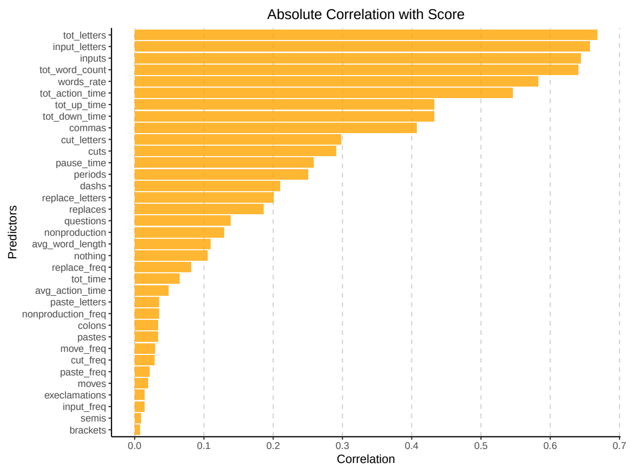


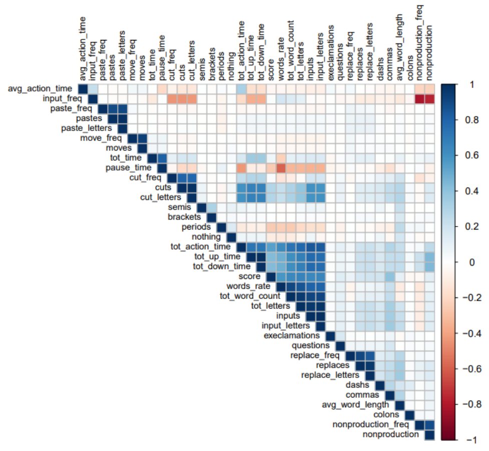
From the histogram and box plot above, we can see that the score of 0.5 are outliers. In fact, only 5 of the 2471 essays (0.2%) received a score of 0.5. While so few of the scores are 0.5, we have decided to NOT remove these scores. With there being so few, this shouldn't impact the models in a major way.

The data looks like a bell curve, which gives way to a normal distribution. Most of the data (69.8%) lie between a score of 3 and 4.5, with most scores being a 4 (20.3%). This may cause many models to be biased towards the average. However, a model that only predicted a 4 would have a decent prediction. Such a model would be exactly correct 20.3% of the time and would have a RMSE of 1.047 – off by 1 score on average.

It is also important to note that there are no 0s in the data. The range of scoring criteria is 0-6, but no scores received such a low score as 0. Hence, if a 0 occurs in the future (or hidden testing data by Kaggle) our model would not predict such. This shouldn't be a major hit towards the model though because no instances of the score 0 have occurred in the 2471 essays observed here.

Now let's look at the different predictors and correlations.





By using the absolute correlation, this shows the most important predictors. The total amount of letters, inputs, and words are all among the top few. Words per minute is the highest rate variable. Next comes totals for action, up, and down time, followed up by commas.

It is interesting to see average word length perform poorly, with a correlation of only 0.11. We thought average word length would be important because of its explanation of complex word usage. Perhaps we should have used certain percentiles of word length to capture this ideal. With that being said, if it wasn't for trying to develop that predictor, we wouldn't have developed total letters or input letters – the top two predictors by correlation at 0.67 and 0.66 respectfully.

Most of the correlations are positive, meaning the larger they get, the higher the score is on average. However, a few variables have a negative correlation. Pause time, periods, and the nothing rate are the top three negatively correlated variables. These make sense as when one is not working on their essay often (high pause time and nothing rate) it means they get lower scores on average. Also, if one uses more periods, their sentences will probably be more basic than one who uses more commas – most positively correlated punctuation variable.

While there are many high correlated variables with score, there also exists a lot of multicollinearity. The graph on the right shows this fact. Take the tot_letters variable for example. Many of the squares are colored in that row-column combination, suggesting collinearity with other predictors. Because of a large amount of multicollinearity, we must be careful when modeling that we either remove issue or choose modeling methods where collinearity can't be an issue.

Now we will move onto our models we developed. Note: The data is randomly split into training (80%) and testing (20%).

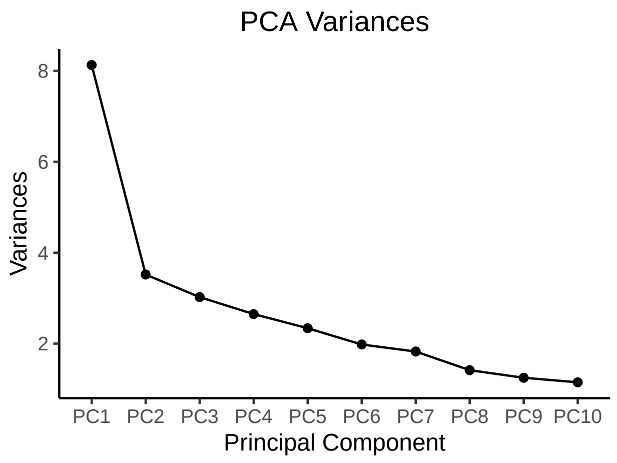
## Unsuprivised Learning

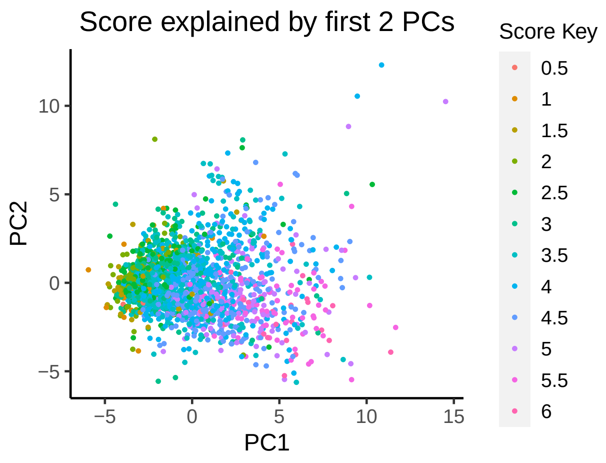
### PCA

The first unsupervised learning model we will apply is principal component analysis (PCA). PCA is used for dimension reduction and is very common. The way the algorithm works is it finds the direction in the data with the least amount of variation. Then, it transforms the original predictors into a new set of uncorrelated variables, called principal components. One must always center the data when doing PCA, but scaling is only used when needed. (Scaling is needed when the variables have much different ranges.) The main downside to PCA is the lost of interpretability. This is a great choice for this dataset because of the large amount of multicollinearity among the predictors.

As we apply PCA to our training data, we must center and scale it. The reason scaling was chosen was due to the difference in predictors. For example, the range of the total amount of letters and words per minute are completely different. Once this is done, the variances among the top-10 components are plotted *on the left*. As we can see, the variances fall off significantly after the first component and continue to fall. This suggests the first component (and a few after) can tell us a lot.

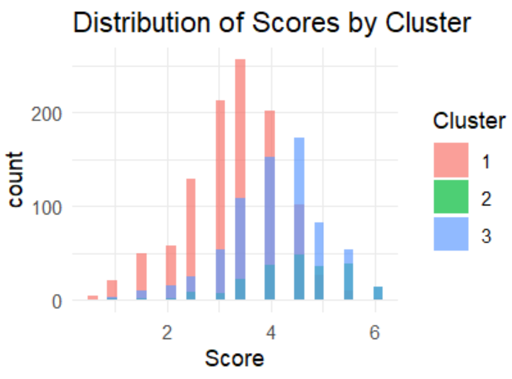
*On the right* we compare the first two PCs. As we can see, there is not a lot of separation here. However, in general, the score goes up the larger PC1 is. Perhaps this component alone can predict score pretty well. In fact, the first three components are at 0.61, -0.21, and 0.11 correlation respectively. Thus, one could get a pretty good model off the first few components – especially when considering the reduction in multicollinearity issues.





### K-Means

The k-means model is a clustering algorithm used in machine learning to group data into a specified number (k) of clusters. It works by assigning each data point to the nearest cluster based on the mean of the points in the cluster. This process iteratively adjusts the cluster centroids until it finds the most cohesive grouping. It's commonly used for market segmentation, pattern recognition, and data analysis but requires the number of clusters to be defined in advance.



The histogram illustrating the distribution of scores across the three clusters reveals distinct patterns. Each cluster represents a unique grouping within the dataset, characterized by varying ranges and concentrations of essay scores. This visual representation offers a clear perspective on how the clusters differ in terms of score distribution, reflecting the diversity in essay quality and characteristics captured by the K-means clustering approach.

Table 1: Summary of Clusters: Mean Scores

| Cluster | Mean Score |
|---|---|
| 1 | 3.294118 |
| 2 | 4.497674 |
| 3 | 4.096377 |

Table 2: Distribution of Scores Within Each Cluster

| 0.5 | 1 | 1.5 | 2 | 2.5 | 3 | 3.5 | 4 | 4.5 | 5 | 5.5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 21 | 49 | 58 | 129 | 213 | 256 | 202 | 102 | 26 | 10 | 1 |
| 0 | 1 | 1 | 2 | 8 | 7 | 22 | 37 | 48 | 36 | 39 | 14 |
| 0 | 3 | 10 | 16 | 25 | 53 | 108 | 152 | 173 | 83 | 53 | 14 |

The analysis of mean scores and distribution within each cluster is summarized in two tables. The first table shows the mean score for each cluster, indicating the average essay quality within each group. The second table provides a detailed breakdown of the distribution of scores within each cluster, offering a granular view of how essay scores are spread across different clusters. This detailed analysis helps in understanding the characteristics and score tendencies of each cluster, further enriching the insights gained from the clustering process.

The clustering results may be useful in many ways for our supervised model, for example:

- Insights and Strategy: Understanding how the data points are grouped can provide insights into different segments of your data, which might be useful in tailoring specific strategies or models for different segments.
- Model Performance Evaluation: Comparing the performance of your supervised model with and without the cluster assignment as a feature can help in evaluating the impact of clustering on your predictive model.

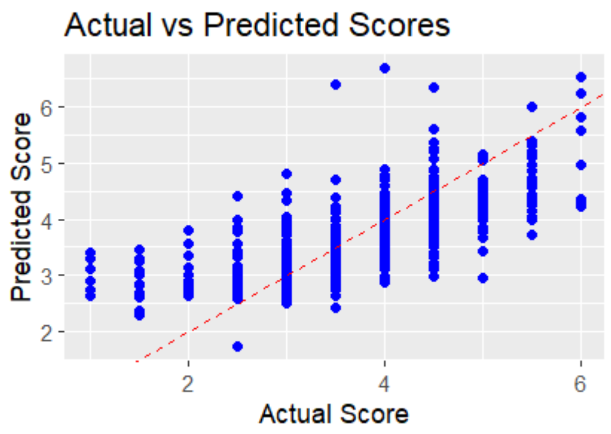
## Regression/Classification Methods

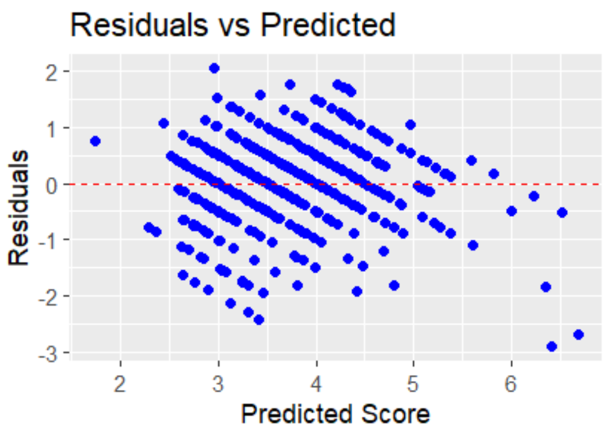
### Linear Regression

Linear regression is a statistical method used for modeling the relationship between a dependent variable and one or more independent variables. The basic idea is to fit a linear equation to the observed data. The primary purpose of linear regression is to understand the relationship between variables and to predict values. In our case, it is used to predict essay scores based on writing styles to understand the impact of writing style on the quality of an essay.

The dataset is divided into 'k' parts or folds, done in a random manner.A 10-fold cross-validation was used, meaning the data was split into 10 subsets. The model is trained on 'k-1' parts of the data. The trained model is then validated on the remaining part (the validation set). This process is repeated k times, with each of the k subsets used exactly once as the validation data.

The primary evaluation criterion used in our analysis is the Root Mean Squared Error (RMSE), which is reported as 0.7275. Additionally, we provided a predicted vs. actual plot and a residuals vs. predicted plot to help visualize the performance of the linear regression.





From this actual vs. predicted plot we can see the relationship between the actual scores and the predicted scores. A good fit in a linear regression model would show the points closely aligned along the diagonal line, indicating that the predicted values are close to the actual values. In this plot we can see that the model is better at predicting scores 3.0-4.5, than is it on the extremes (0.5-2.0 and 4.5-6.0).

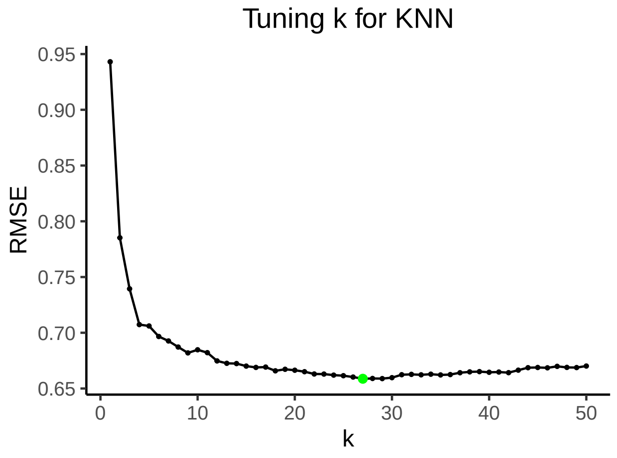
From this residuals vs. predicted plot we can see the residuals on the vertical axis and the predicted scores on the horizontal axis. Ideally, in a well-fitting linear regression model, the residuals should be randomly scattered around the horizontal axis, indicating that the model's errors are random and do not follow a pattern. In our plot we can see that the points are randomly scattered around the line, indicating that the error's are not following a pattern.

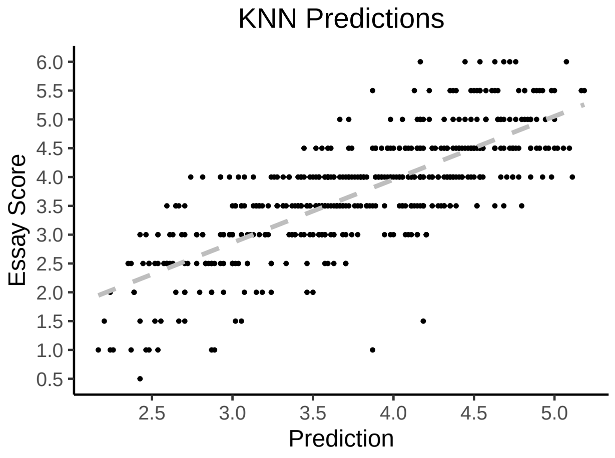
## KNN

K-nearest neighbors (KNN) is the next regression model we will apply. This is a nonparametric model in which it centers the prediction around the local average. There is one parameter we need to tune here – k. This is simply how many neighbors should we pay attention to. As k increases, bias increases and variance decreases. On top of this, the model is smoother and has less *jumps* as k increases. The primary disadvantage for this model is the jumps can cause irregularities at the jump points. On top of this, the interpretability is not very good.

Mentioned above was the fact that k needs to be tuned. To do this, we will simulate the KNN model for k value from 1 to 50. Whichever has the lowest error will be the model that we choose. After running the simulation, we find that 27 is the optimal k value.

The model that we chose has an RMSE of 0.659. This is similar to our other results thus far and represents a good fit of the data. The variance seems constant throughout. There are a couple outliers predicted at ~4 that are actually a 1 and 1.5. We also see that the range of prediction is only between ~2 and ~5.5. With most essays scoring around 3.5-4, this makes sense. That being said, this model seems to be a good fit.





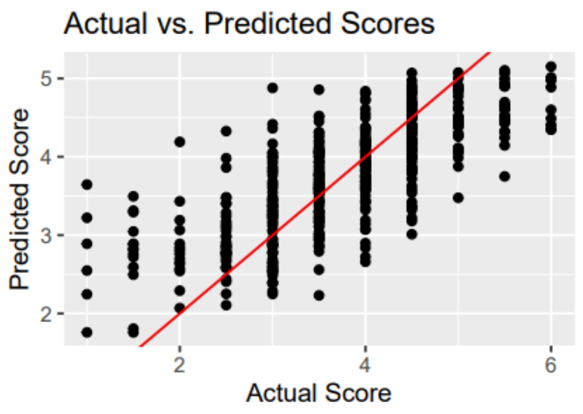
## Random Forests

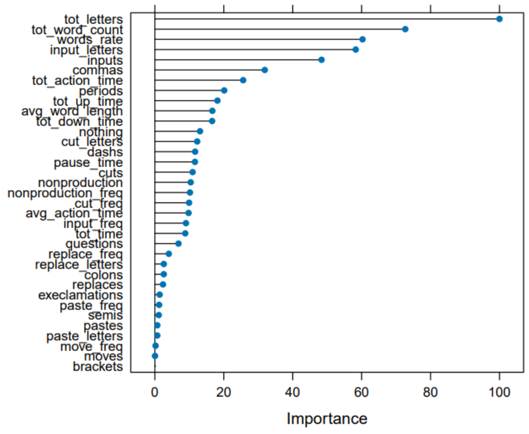
A random forest model is an ensemble learning method that builds multiple decision trees for robust and accurate predictions. It is known for handling overfitting and excelling in both classification and regression tasks, especially with large datasets and diverse features. However, it is computationally demanding, lacks transparency in decision-making, and requires considerable memory, yet is beneficial for datasets with complex relationships and collinearity issues like our current dataset.

As with the other models, the `id` column was removed since it's a unique identifier and not useful for predicting `score`. The data was also again split into testing and training portions. We decided to use all available variables when fitting the model as random forests can effectively handle a large number of predictors, are considerably robust to collinearity, and avoid overfitting well. For parameter tuning, we used 5-fold cross-validation to optimize the `mtry` parameter while RMSE was used as the evaluation criteria. The `mtry` parameter represents the number of variables (or features) randomly sampled as candidates at each split when building a tree. It is generally considered the most impactful parameter in the model. Due to our limited computational power, we focused on tuning the `mtry` parameter while leaving the other parameters as their defaults. We used the sequence of integers from 7 through 17 as the grid of potential values for the parameter. This range was chosen as it is centered around the generally advised starting value for `mtry` (one third the number of predictors, about 12 in our case), while keeping room for tuning. The model was specifically fit utilizing the `randomForest` package with tuning being performed with the `caret` package. Utilizing the best model (lowest RMSE), we then made our predictions on our testing dataset. The model achieved an RMSE of about .672 utilizing an `mtry` value of 7.

The model achieved an RMSE of about .672 on the testing set of data This means that the square root of the average squared differences between the predicted and actual values is about 0.672

To further help evaluate the model's fit, we created a scatter plot of actual vs. predicted values. (We also created a variable importance plot to help assess which specific variables were most impactful in the random forest model.)





Our predicted values generally fall relatively close to the line representing the predictions if they were exactly equal to the actual values. This indicates that the model is fitting well.

Based on the variable importance plot, `tot_letters`, `tot_word_count`, `words_rate`, `input_letters`, `inputs`, and `commas` are the most important (predictive) variables in this model in that order. There is a significant drop off in variable importance starting with `commas`

Overall the random forest model was able to provide a well fitting model despite collinearity issues in the data. If given access to more computing power, we could have also tuned other parameters such as `nodesize` and `sampsize`. This would have likely further improved the model.

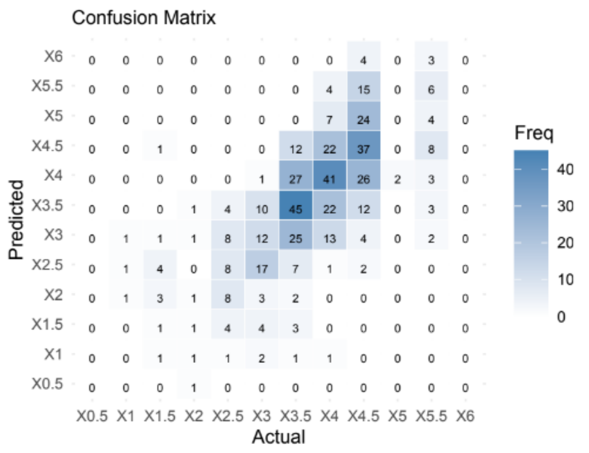
## Gradiant Boosting (XGBoost)

XGBoost is a well optimized, advanced ensemble learning method using gradient boosting to create multiple decision trees, offering precise predictions. It's effective in classification and regression tasks, especially with complex, large datasets. While efficient in handling sparse data and overfitting, XGBoost can still be somewhat resource-intensive and less transparent in decision-making. Its ability to tackle collinearity makes it well-suited for our current dataset with intricate relationships

As with the other models, the `id` column was removed since it's a unique identifier and not useful for predicting `score`. In the tuning process of the model, a predefined hyperparameter grid was utilized that included two values for `nrounds` (100 and 200), which dictate the number of boosting iterations, and two values for `max_depth` (3 and 6), which control the complexity of the trees. Additionally, two values for the learning rate `eta` (0.01 and 0.1) were chosen to determine the step size shrinkage used to prevent overfitting. The rest of the parameters, `gamma`, `colsample_bytree`, `min_child_weight`, and `subsample`, were held constant at 0, 1, 1, and 1, respectively, to simplify the model and ensure efficient computation. The `caret` package's train function facilitated the cross-validation process, methodically working through combinations of these parameters to identify the set that yielded the highest accuracy and best generalization performance as indicated by the cross-validated results. The best performing model had the following tuned parameter values: `nrounds` = 100, `max_depth` = 3, `eta` = .01

The final model achieved an accuracy of about .308 on the testing set of data. This means that the model correctly classified the score of a given essay about 30.8% of the time. This is quite low as it can be very difficult to create accurate classifiers for models with a somewhat large amount of response possibilities like this dataset.

To further help evaluate the model's fit, we created a confusion matrix.



| | X0.5 | X1 | X1.5 | X2 | X2.5 | X3 | X3.5 | X4 | X4.5 | X5 | X5.5 | X6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 3 | 0 |
| X5.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 15 | 0 | 6 | 0 |
| X5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 24 | 0 | 4 | 0 |
| X4.5 | 0 | 0 | 1 | 0 | 0 | 0 | 12 | 22 | 37 | 0 | 8 | 0 |
| X4 | 0 | 0 | 0 | 0 | 0 | 1 | 27 | 41 | 26 | 2 | 3 | 0 |
| X3.5 | 0 | 0 | 0 | 1 | 4 | 10 | 45 | 22 | 12 | 0 | 3 | 0 |
| X3 | 0 | 1 | 1 | 1 | 8 | 12 | 25 | 13 | 4 | 0 | 2 | 0 |
| X2.5 | 0 | 1 | 4 | 0 | 8 | 17 | 7 | 1 | 2 | 0 | 0 | 0 |
| X2 | 0 | 1 | 3 | 1 | 8 | 3 | 2 | 0 | 0 | 0 | 0 | 0 |
| X1.5 | 0 | 0 | 1 | 1 | 4 | 4 | 3 | 0 | 0 | 0 | 0 | 0 |
| X1 | 0 | 0 | 1 | 1 | 1 | 2 | 1 | 1 | 0 | 0 | 0 | 0 |
| X0.5 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Unfortunately, the model demonstrates poor performance in classifying the positive instances of most classes (scores), evidenced by low sensitivity and precision, while it correctly identifies negative instances with high specificity. The highest sensitivity is around 46% for Class: X3.5 (score of 3.5), but for several classes (scores), it's at 0%, showing a particular weakness in detecting true positives. Given the low detection rates and moderate to low balanced accuracy, the model seems to struggle with the class imbalance present in the dataset, indicating a need for model refinement, further feature engineering, or even alternative approaches.

## Appendices

### Data Processing Code

```r
df1 = train_logs |>
  group_by(id) |>
  summarise(# Time
            tot_word_count = last(word_count),
            tot_action_time = sum(action_time),
            avg_action_time = mean(action_time),
            tot_up_time = sum(up_time),
            tot_down_time = sum(down_time),
            tot_time = max(up_time) - min(down_time),
            ## WPM
            words_rate = tot_word_count / (tot_time / 60000),
            ## Close
            pause_time = tot_time - tot_action_time,

            # Activity Freq / Totals
            nonproduction_freq = sum(activity == "Nonproduction") / n(),
            input_freq = sum(activity == "Input") / n(),
            cut_freq = sum(activity == "Remove/Cut") / n(),
            paste_freq = sum(activity == "Paste") / n(),
            replace_freq = sum(activity == "Replace") / n(),
            move_freq = sum(substr(activity, 1, 4) == "Move") / n(),
            cuts = sum(activity == "Remove/Cut"),
            inputs = sum(activity == "Input"),
            nonproduction = sum(activity == "Nonproduction"),
            pastes = sum(activity == "Paste"),
            replaces = sum(activity == "Replace"),
            moves = sum(substr(activity, 1, 4) == "Move"),

            # Punctuation Rates
            ## Not perfect, but close
            dashs = (sum(text_change == "-" & activity == "Input",
                na.rm = TRUE) - sum(text_change == "-" &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,
            semis = (sum(text_change == ";" & activity == "Input",
                na.rm = TRUE) - sum(text_change == ";" &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,
            colons = (sum(text_change == ":" & activity == "Input",
                na.rm = TRUE) - sum(text_change == ":" &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,
            commas = (sum(text_change == "," & activity == "Input",
                na.rm = TRUE) - sum(text_change == "," &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,
            periods = (sum(text_change == "." & activity == "Input",
                na.rm = TRUE) - sum(text_change == "." &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,
            execlamations = (sum(text_change == "!" & activity == "Input",
                na.rm = TRUE) - sum(text_change == "!" &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,
            questions = (sum(text_change == "?" & activity == "Input",
                na.rm = TRUE) - sum(text_change == "?" &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,
            brackets = (sum((text_change == "["  | text_change == "]") &
                activity == "Input", na.rm = TRUE) -
                sum((text_change == "["  | text_change == "]") &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,

            # No Change
            nothing = (sum((text_change == "NoChange"  |
                is.na(text_change)) & activity == "Input", na.rm = TRUE) -
                sum((text_change == "NoChange"  | is.na(text_change)) &
                activity == "Remove/Cut", na.rm = TRUE)) / tot_word_count,

            # (Approx.) Word Length
            input_letters = sum(text_change == "q" &
                activity == "Input", na.rm = TRUE),
            cut_letters = round(.9*sum(nchar(text_change) &
                activity == "Remove/Cut", na.rm = TRUE)),
            paste_letters = round(.9*sum(nchar(text_change) &
                activity == "Paste", na.rm = TRUE)),
            replace_letters = sum((nchar(str_split_i(text_change, ' => ', 2))
                - nchar(str_split_i(text_change, ' => ', 1))) &
                activity == "Replace", na.rm = TRUE),
            tot_letters = input_letters + paste_letters + replace_letters
                - cut_letters,
            avg_word_length = tot_letters / tot_word_count,

            .groups = "drop") |>
  left_join(train_scores)
```

**References**

- AWQATAK's Model
- Kaggle Competition
- MAOK-YONGSUK's Model
- SEAN's Model