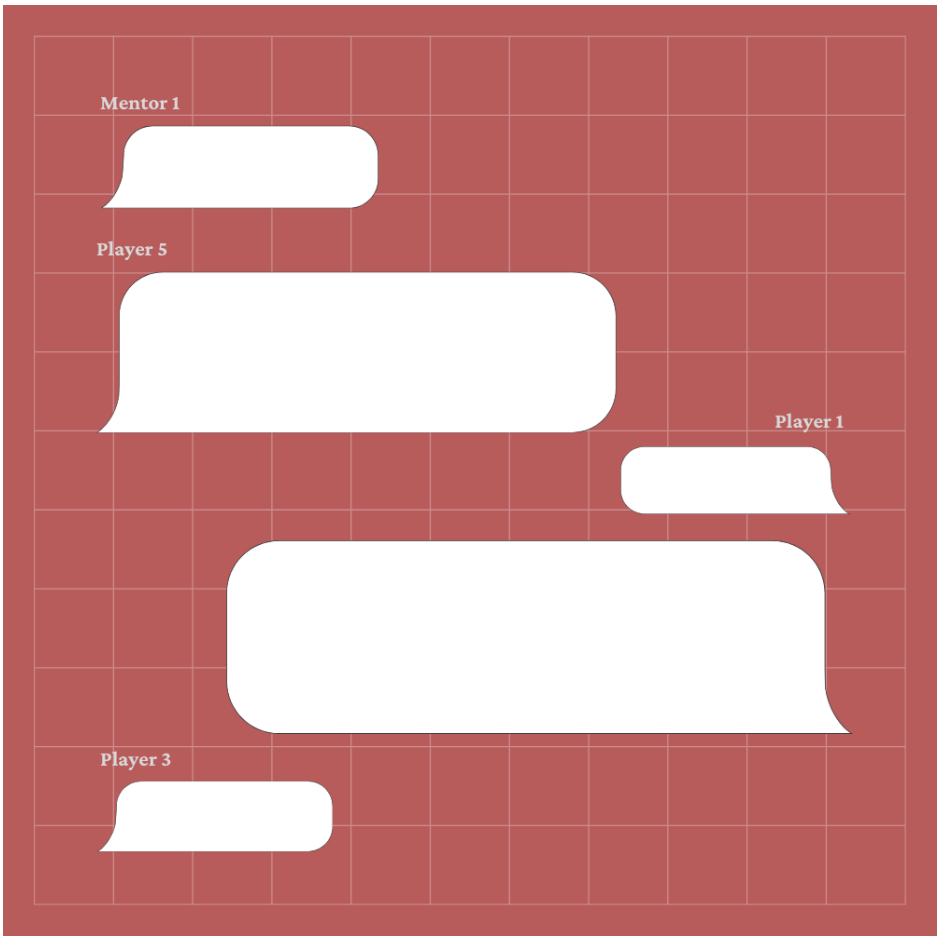
ADS2001 | Data Challenges 3

Virtual Internships

An investigation into the most prominent factors influencing interns' performance



Report completed by:
Adam Choong [33154384]
Alex Huang [33051070]
Emma Kelsall [32592086]
Pooja Kampli [33154759]

TABLE OF **CONTENTS**

EXECUTIVE SUMMARY

As more industries adopt a model whereby the majority of their projects and errands are completed remotely, it has become increasingly evident that more attention must be paid to the assessment of how a new generation of interns fares in this emerging professional environment. In this inquiry, our team became particularly interested in the key factors influencing interns' performance because these findings can be used as indicators of future interns' performances. The analysis was conducted on a fictional virtual internship program, Nephrotex, funded by the University of Wisconsin-Madison and National Science Foundation. Nephrotex is part of a series of simulated internships targeted at freshman college students, intended to assess their potential performance in a controlled environment.

Nephrotex's internship program involved allocating students to different teams to collaborate on the creation of a product that tackled an issue relevant to kidney dialysis. They were provided with guidelines to produce prototypes and given several opportunities to communicate with both external consultants and mentors to acquire technical knowledge, receive feedback and affirm their understanding of the requirements of the internship. During the internship, students could engage in activities such as experimental testing of prototypes and making abstract choices about the design of their products. Students were also required to justify their decisions in the context of the problems they encountered to encourage them to think critically about their design process. By the end of each design process, interns received a result, in the form of an outcome score, graded from 0-7, with a higher score indicating better performance.

The data was collected over a single internship with 15 different implementations and primarily took the form of text messages sent through the messaging system, WorkPro. Each text message was allocated to a given category based on their content. Both the content of messages and their categorical allocations were analysed to detect potential influential factors. The inquiry produced findings that suggested that no outstanding factor influenced the outcome score of interns. Categorical factors had a minimal impact on predicting interns' scores whilst both the content messages sent by interns and how frequently they maintained correspondence with other interns and mentors, seemed to have a higher degree of influence on their performance. This seems to suggest that the quality of each interns' work may be somewhat related to their ability to meaningfully communicate with others in a concise, clear and effective manner. An interesting finding was that consultations from external sources seemed to have little effect on the performance of interns.

The findings of this report can be extended to other medically related industries concerned with product design because the processes involved in the development of interns' products are similar to those of general product design. More generally, the processes applied during this inquiry can be reapplied to any type of virtual internship that involves data extracted from interpersonal communication. It is recommended that similar studies be carried out on other types of virtual environments across several industries to highlight more noticeable insights than the findings that were produced during this inquiry.

Introduction

Background

The virtual internship, set against the backdrop of a fictional medical corporation, Nephrotex, allows students to develop knowledge and skills in biomedical engineering whilst working in teams to design a prototype device that assists patients with kidney failure. Students are required to conduct background research, understand stakeholder needs, design prototypes, test and evaluate prototypes, and justify their design decisions. This project aims to model the performance on the final design report using data collected from the chat records of 15 implementations of the internship and identify the most prominent factors influencing interns' performance.

There are two main parts to this project which are categorical analysis and text analysis involving natural language processing. Categorical analysis involves the investigation of the presence or absence of key concepts in engineering discourse that relate to particular design moves and justifications. Through analysis, the aim is to be able to determine correlations between different variables and the quality of the final design report in order to identify the factors that contribute to successful performance. Therefore, categorical analysis is an essential step in achieving our research objective of modelling the relationship between intern discussions and performance in the virtual internship.

Text analysis and natural language processing (NLP) are fundamental as they enable a deeper analysis to be conducted on text messages which can uncover patterns and themes that may not have been readily apparent through categorical analysis. NLP techniques are used to analyse the language expressed by the interns and can identify key concepts and themes from their discussions, which in turn assists in forming an understanding of their reasoning, decision-making and justifications of design choices. The results acquired can then be compared to the outcomes achieved by each intern for the final design report to analyse which types of themes are associated with higher quality reports and gain a more comprehensive understanding of the factors that contribute to successful performance in the Nephrotex virtual internship.

This report highlights the contributions of each team member. Adam Choong completed data analysis, focussing upon the categorical variables, he built decision tree and random forest classification models. Alex Huang performed data wrangling and conducted multivariate linear regression upon the categorical variables. Emma Kelsall specialised in textual analysis and modelling, producing a final model to find the most important features. Pooja Kampli prepared background research and implemented a support vector classification model.

Supplied Data

For this project, the supplied data consisted of a single data set containing chat messages from 15 Nephrotex Virtual Internship implementations. Each chat message was evaluated and labelled. Firstly, each chat message was assigned a userID (which was the unique identifier of the intern or mentor who wrote the message), an implementation letter (denote the implementation that the

intern or mentor was a part of), a group ID (the team which the intern or mentor was talking to), a role name (whether the person was a mentor or player (student/intern)), a room name (the assigned activity the sender was participating in when sending their message), the sender's outcome score (quality of final design report) and finally the word count of the message. In addition, each message was assigned six dummy variables to indicate various characteristics, such as whether the person talked about making design moves (examples include experimental testing, making design choices, or asking questions) or design justifications (examples include, referring to stakeholder requests, performance parameters, and experimental results, or facilitating communication among engineers).

Data Wrangling and Processing

Data Quality

For this project a clean dataset was provided where not a lot of data imputation was required before analysis and modelling could be performed. To begin, shallow data analysis was performed. The first thing examined was the descriptive statistics of the data set shown in figure 1. From this, it could be determined that the provided dataset contained 19180 rows, with each one representing a chat message. Some other useful insights that were highlighted included the minimum, mean and maximum of the word count and outcome variables. These insights would prove to be useful later in the project.

Figure 1 : A statistical description of the given dataset

	userIDs	Line_ID	group_id	m_experimental_testing	m_making_design_choices	m_asking_questions
count	19180.0000	19180.0000	19180.0000	19180.0000	19180.0000	19180.0000
mean	202.4332	9592.7938	3.9167	0.0287	0.1029	0.1870
std	118.3552	5537.8007	1.3979	0.1670	0.3038	0.3899
min	1.0000	1.0000	2.0000	0.0000	0.0000	0.0000
25%	96.0000	4796.7500	3.0000	0.0000	0.0000	0.0000
50%	204.0000	9593.5000	4.0000	0.0000	0.0000	0.0000
75%	317.0000	14388.2500	5.0000	0.0000	0.0000	0.0000
max	393.0000	19183.0000	6.0000	1.0000	1.0000	1.0000

j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	OutcomeScore	wordCount
19180.0000	19180.0000	19180.0000	19180.0000	19180.0000
0.0181	0.0522	0.0211	3.7416	12.4895
0.1335	0.2225	0.1436	1.4648	14.1172
0.0000	0.0000	0.0000	0.0000	1.0000
0.0000	0.0000	0.0000	3.0000	4.0000
0.0000	0.0000	0.0000	4.0000	9.0000
0.0000	0.0000	0.0000	4.0000	17.0000
1.0000	1.0000	1.0000	8.0000	1032.0000

In addition to exploring the descriptive statistics, it was important to determine whether the dataset contained any missing values. After performing a simple '.isnull().sum()' on the data, it was discovered that the 'RoleName' column contained 3 missing values. As the percentage of missing data was very small and 'RoleName' was a categorical variable, it was chosen to remove these rows before performing further analysis.

Data Wrangling and Processing

In order to get the dataset into an appropriate state for performing analysis and modelling, it was necessary to drop columns that wouldn't be required. For the categorical analysis of this project, the columns 'ChatGroup', 'Content', 'RoomName' and 'Line_ID' were all dropped, as it was decided that these variables were not relevant to the goal of predicting outcome scores. Chat messages were grouped by 'implementation', 'userID' and 'group_Id' to make it easy to analyse individual players (students) and how scores differed depending on the students' attributes. Finally for the categorical analysis, only chat messages where the 'RoleName' was 'Player' were included. Chat Messages by the mentor were omitted as the outcome score for every mentor was 4. This would create a bias when trying to predict the outcome score.

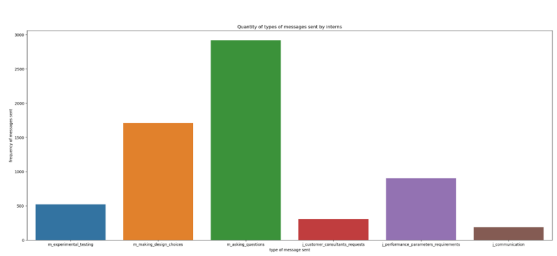
For the natural language processing section of the project, all columns except 'Content', 'OutcomeScore' and 'WordCount' were dropped. This section of the project looks directly at the language and words used in the messages, rather than how it is categorised. Thus it was only necessary to keep variables that were directly related to this. It was then important to process the chat content into a format that would be most appropriate for performing natural language processing techniques. A 'preprocess_message(text)' function was created to perform this task on every chat message. The first step performed was tokenization. This involved splitting sentences into individual words and removing any non-word characters such as punctuation and white space. The next step involved lemmatization using the 'WordNetLemmatizer' function from the 'nltk' library and converting all text into lowercase. Lemmatization is the process of reducing inflectional forms and derivatively related forms of a word to a common base form. An example of this is the reduction of organising, organisation, and organises to organise. This stage is important in this process as it helps to normalise the words in the data and reduces the dimensionality of the feature space thus reducing the number of unique words in the dataset. Morphological analysis is undertaken by the lemmatizer to accurately achieve this. This can aid in improving accuracy when creating models. The final step of processing the chat message data involves the removal of stop words. Stop words are commonly used words that are not considered useful for language analysis due to their high frequency and lack of semantic meaning. Examples include "the", "a" "to" and "are". As these are very common, they dominate frequency distribution of words and thus make it difficult to determine relevant and meaningful words in the chat messages. These words were removed simply to simplify and reduce the noisiness of the data, making it easier to analyse. After running the chat messages through this function, a new column was created to store the cleaned text of each chat message. In addition to this preprocessing, new features were also extracted from the chat messages such as the number of messages sent by each user, as well as a number of other features such as keyword frequency, sentiment score and topic probabilities (extracted through Natural Language Processing) which will be explained and analysed in the textual analysis section of this report.

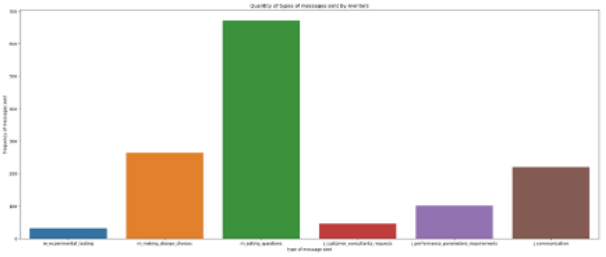
Having completed the processing stages of the dataset, the data is now in a condition ready to be used for analysis and modelling.

Exploratory Data Analysis

In this project, all the data was used albeit in different sections. For example, the subsequent figures 2 and 3 were generated using a quantitative data set, created by the processes used during data wrangling to generate a categorical data set. For example, a bar graph was generated from the cumulative totals of the types of messages sent by interns and mentors.

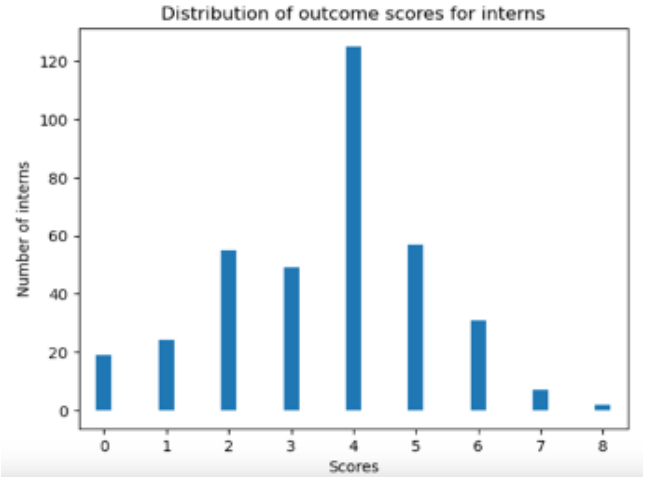
Figures 2 and 3: The distribution of the types of messages sent by interns (left) and mentors (right)





As observed in figure 2, the distribution of the types of messages sent, interns most frequently asked questions. This is likely because interns could only raise concerns with their mentors through WorkPro, in the form of questions to seek clarification on different aspects of their project that they were uncertain of. Interns also expressed their design choices very frequently through the WorkPro applications, perhaps because they could subsequently receive responsive feedback on their choices from mentors. Interns least commonly sent out messages intended to communicate with engineers because perhaps, they were not required to facilitate activities involving the engineers or address their concerns. They also seemed to rarely ask for advice from customer consultants, possibly because of the limited access that they had to consultants or their lack of willingness to engage with consultants. Both reasons deduced from the analysis of the relationship between consultants and interns may help explain the overall poor performance of interns, indicated in figure 4 . For example, the lack of correspondence between consultants and interns may have left interns under informed about what constitutes an effective product for kidney dialysis, causing performance issues with the prototypes they may have designed. Regarding the distribution of messages sent by mentors, the most common type of content contained in messages was questions which may imply that majority of the contact between interns and mentors was in the form of providing hints through posing potential questions that may have assisted interns. For example, a “question” type message sent by a mentor was “How did you choose the best surfactant?” which was intended to encourage interns to think about their thought process more critically when deciding on the best surfactant, rather than revealing the surfactants which they should consider. The least common type of messages sent by mentors was the consultant type, likely because mentors rarely had a presence in interns’ discussions with consultants apart from organising the logistics of the sessions between consultants and interns. Moreover, mentors sent significantly less messages than interns. This may be because one mentor had to manage multiple groups and had limited ability to sustain a heavy amount of communication with any single group, thus reducing the number of messages they were able to send.

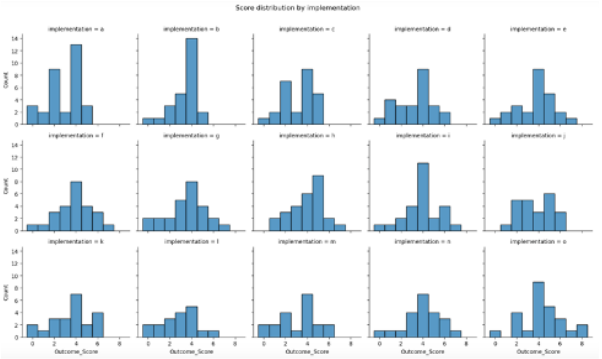
Figure 4 : The distribution of scores achieved by interns

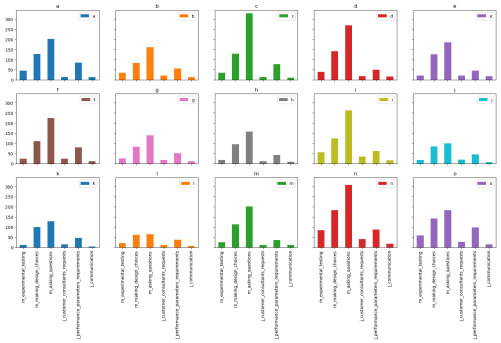


In figure 4, the mode and average score is clearly 4. As previously mentioned during the data wrangling process, each mentor's assigned score was 4 but these mentors' messages were excluded from the data frame before the analysis was conducted, avoiding any errors with regards to this insight. In addition, figure 4 shows that the scores are mildly skewed to the left therefore suggesting that the internship program at Nephrotex has an above average degree of difficulty because the majority of students either performed below average or at an average level.

Furthermore, because each implementation represents a 'round' of internships, the distribution of each internship can be observed for both the scores and frequency of messages sent to determine the presence of a potential relationship between interns' performance and the types of messages they sent at an increased depth.

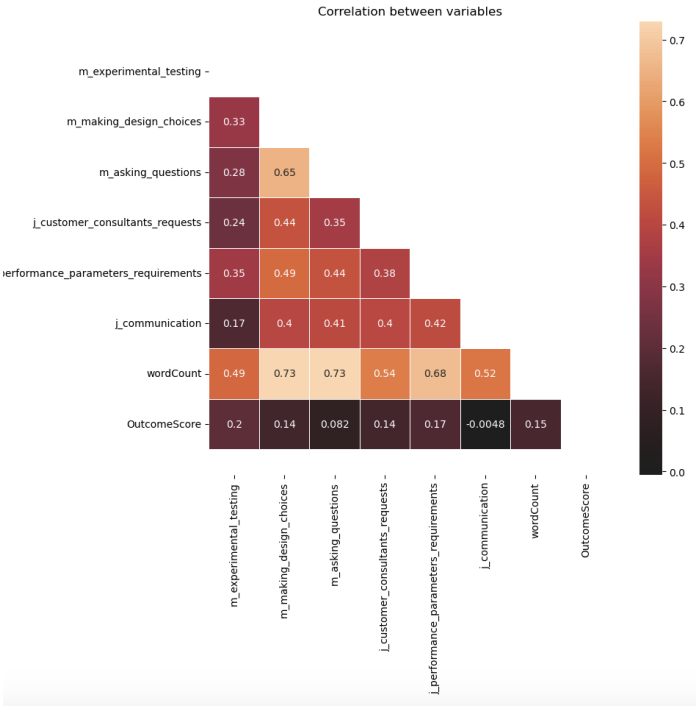
Figures 5 and 6 : The distribution of scores achieved after each round of internships and the distribution of the amount of messages they sent.





In both figures 5 and 6, there seems to be a lack of a visible relationship between the distribution of the outcome scores achieved by interns and the distribution of the types of messages they sent because the relative distribution of messages seems consistent but the distribution of scores for different implementations vary. However, more concrete findings are shown in the form of a correlation matrix at figure 7.

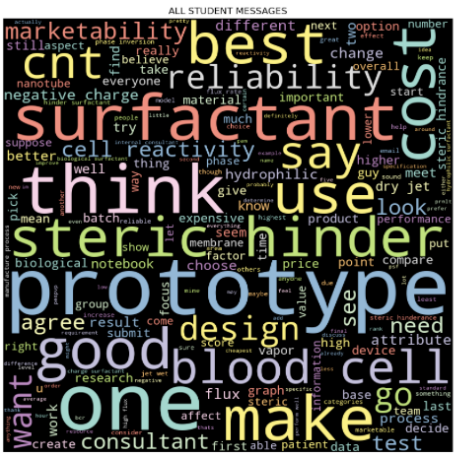
Figure 7: A more concrete measure of correlation between different variables



The observations to pay attention to in figure 7 are the correlation scores between outcome score and the rest of the variables. All these scores are very low which indicates the lack of a definite relationship between any other variable and the outcome score of interns. Since the correlation scores between some variables are reasonably high, for example, the correlation score between m_asking_questions and m_making_design_choices, there will be some issues with multicollinearity later, meaning that when modelling is performed, the importance of both features in terms of their impact on outcome score may become ambiguous which could hinder the process of determining distinctive factors that affect outcome score.

Moreover, another important component in the analysis was the use of the techniques provided by natural language processing to determine the frequency of words used by both interns and mentors to determine whether high-performing students used more of certain key words and conversely whether low-performing students used certain key words frequently or did not use them. A concise visual is provided in figures 8 and 9 to show which words were commonly associated with the content that interns talked about using WorkPro.

Figures 8 and 9 : Word clouds displaying the words that interns and mentors most frequently used in their messages, represented by size.



From figure 8, scientific words and phrases like “blood cell” and “surfactant” seem to have a mild presence, perhaps suggesting that they may not have influenced the outcome score as much as commanding words like “think” and “hinder” or words associated with the designing process, like

“prototype”. A reason why “command” words may have had an impact on score is that commands tend to affect team dynamics and therefore may influence individual interns’ performance as a result. “Design” words could affect the outcome of an individual’s performance because the use of such words may alter how the thought processes of interns are expressed. For example, if interns frequently refer to their prototypes during the process of assessing their products, this may imply that they are relying heavily on the insights produced by their prototypes, therefore significantly influencing outcome score. The results from using similar natural language processing techniques could suggest that a series of features may not be influencing outcome score, but rather, the content of the messages being sent could be the defining trait in predicting outcome score.

In figure 9, further analysis done on mentors’ responses to the messages sent by interns, using the same mechanism as before, shows that action and command words such as “write”, “share” and “submit” were the most frequent type of words contained within the messages sent by mentors, perhaps implying that mentors approached interns in a mainly instructive capacity, providing them with required tasks to fulfil rather than advising them of potential ways to improve. There was a significantly lesser presence of messages of a technical nature as words like “surfactant” and “graph” were used a lot less than other phrases, perhaps suggesting that mentors were either restricted in providing technical knowledge to interns or that interns did not probe mentors for such information. This may lead to interns producing results that may be lacking with respect to the technical aspect of their project. In addition, a significant proportion of words and phrases, such as “make sure” and “deliverable list” were intended to clarify requirements for the project. This may imply that interns’ main source of assistance were their mentors but because mentors were either restricted in the information they were allowed to give or the very nature of their role was to provide instructions, the feedback they gave to interns when asked for further clarification may not have been as helpful. This could potentially explain the poor performance of the majority of interns.

Predictive Modelling

Overview

This problem was tackled as both a regression and classification problem although once it was found that regression algorithms produced illogical results, the project was viewed as a classification problem whereby each outcome score was assigned as a potential target label. Two different approaches were taken, one which looked at the categorical analysis of the text messages and another which looked at the text analysis which focused more on the semantics and lexicology of the messages. The models used on the categorical data set were linear regression, support vector machines classifier, random forests classifier and decision trees classifier. All features were examined in hopes of determining which features, and which combination of features produced the most accurate results for predicting outcome score. Moreover, to ensure that a fair comparison was made between models, the split between training and testing sets was identical, using a 80-20 train-test split.

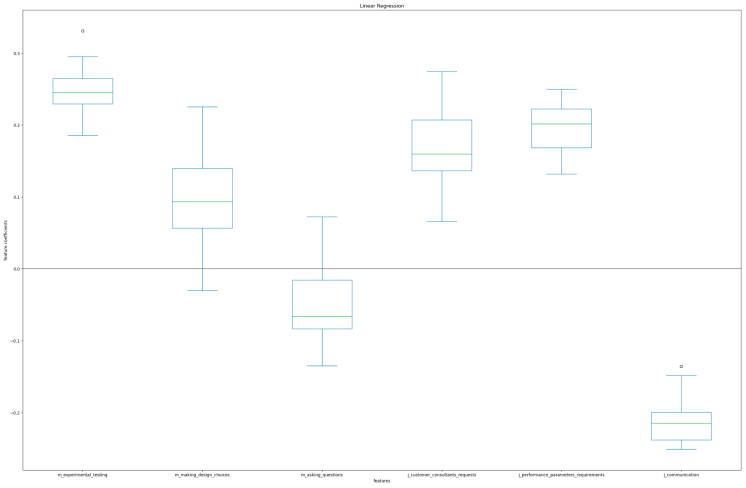
It should also be noted that modelling for categorial analysis was completed taking into account all features except word count because many messages contained noisy data in the form of excess words which did not meaningfully affect the performance of interns. However, the text analysis performed accounted for the majority of the meaningless data through processes elaborated on later and utilised the text messages as its input data rather than the qualitative counts of the types of messages sent.

Categorical Analysis

Multivariate Linear regression

The first model utilised was multivariate linear regression because the model would have demonstrated whether it was appropriate to tackle the project's problem using regression algorithms. There was no need to tune any hyperparameters. Instead, regression analysis was done on coefficients to determine the importance of features. To ascertain whether the model was appropriate to use, the R2 score was used and to determine the accuracy of the model, the RMSE and MAE were used. Before performing regression analysis, the data's feature variables had to be normalised to ensure that they had the same measurement system. The normalisation technique used was the z-score method whereby the average of every column was subtracted from every entry in the feature variables' data set then that result was divided by the standard deviation of all entries in that column. The results of the coefficient analysis are shown in figure 10.

Figure 10: The results of coefficient analysis



As observed in figure 10, all coefficients have a moderate variability. It appears that j_communication has the worst impact on outcome score because all its coefficients lie in the negative region. However, because of the low correlation between j_communication and outcome score, as shown in the correlation matrix at figure _, this may not significantly affect outcome score. Furthermore in figure _, m_experimental_testing has the highest correlation with outcome score and seems to have a positive effect on this variable since all its coefficients are in the positive range. The accuracy metrics for R2, RMSE and MAE were -0.113, 1.624 and 1.372 respectively. The

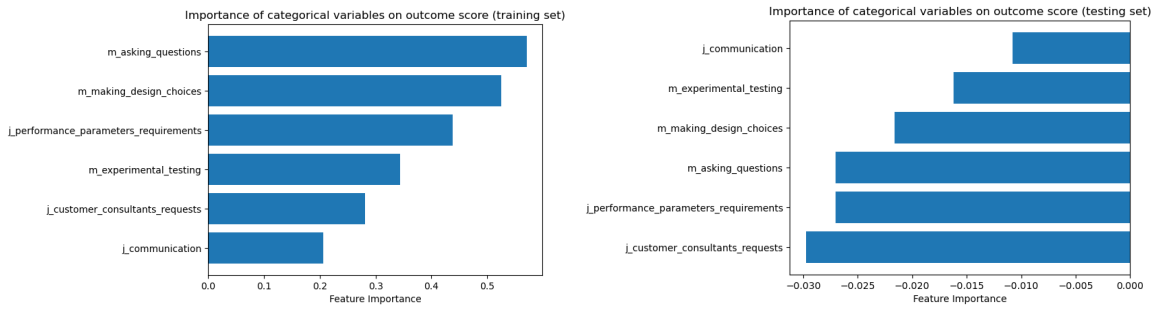
presence of a negative R2 score indicates that the features variables and target labels most likely have a nonlinear relationship. In addition, regression methods may not be appropriate because the target labels are discrete. Therefore, the choice to approach the project using regression models was discounted.

Support vector machines (SVMs)

One of the models used in the project was Support vector machines (SVMs) which are a type of supervised learning algorithm that can be used for classification and regression tasks. They work by finding a hyperplane that separates different classes of data, and they use kernel functions to transform the data into a higher-dimensional space where it can be more easily separated. This model was used due to its effectiveness in dealing with multidimensional data sets and its ability to work with non-linearly separable data.

In the implementation of SVM, the radial basis function (RBF) kernel was used as it is highly effective for non-linear classification tasks. The optimisation method used was 'GridSearchCV' from Scikit-learn with 3-fold cross-validation, allowing a range of possible hyperparameters to be tested to produce the best results on the testing data. The kernel, gamma and the cost parameter C were tuned during cross validation. Gamma determines the shape of the decision boundary and C controls the 'hardness' of the boundaries generated; it can be tuned to accept different levels of misclassification. The method selected for hyperparameters and optimisation can have a significant impact on the performance of the SVM model. The model was tuned to achieve the best performance overall with regards to accuracy and produced the following hyperparameters: 'C' to 10 and 'gamma' to 1. The accuracy score obtained from the SVM model generated was 0.297. This indicates that the SVM model had limited success in accurately classifying instances to outcome scores.

Figures 11 and 12: The feature importance of different variables as decided by applying the permutation function to both the training and testing sets

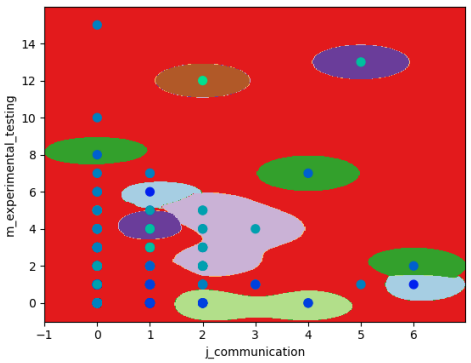


To ascertain the feature importance of the SVM, the permutation function had to be used since the model utilised the radial basis function and was fitted on nonlinear data. The graphs in figures 11 and 12 were generated when permutation was applied to both the training and testing set to assess the degree of overfitting that the SVM had. The differences between the ranks of the

variables, with regards to feature importance, were extreme, indicating that the model was severely overfitted (Jensen, 2022).

Given the low accuracy score and the extreme differences between the results produced by the training and testing sets, it is clear that the SVM approach did not provide reliable predictions for the determining the influencing factors of outcome score. Therefore, alternative modelling techniques should be explored to improve the accuracy and effectiveness of predicting the quality of the final design reports.

Figure 13: The decision boundary generated from the two most important features acquired through permutation on the testing set



Based on the results of the decision boundary generated in figure 13, several insights can be derived. It is important to note that the SVM model achieved an accuracy of nearly 30%, indicating that it was able to classify some instances correctly based on the "j_communication" and "m_experimental_testing" variables. However, the overlapping decision regions suggest that the model faced challenges in accurately separating the different outcome score categories. This can be attributed to the extreme degree of overfitting that the model had potentially because of the presence of noisy data or the fact that the model was not appropriate to use on the data set.

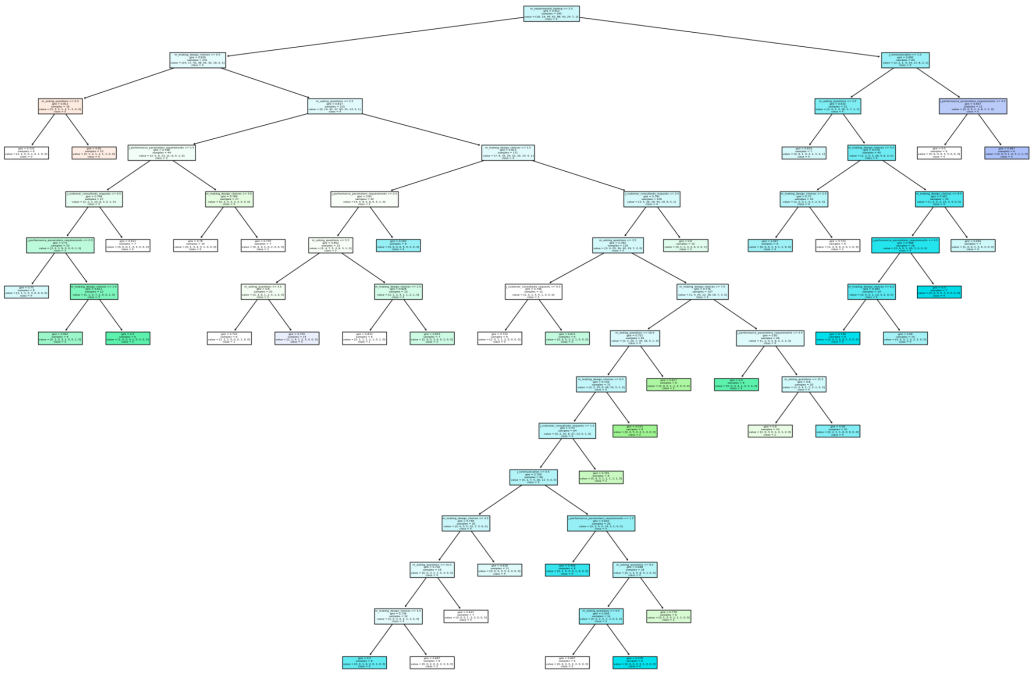
The ‘complex’ decision boundary graph implies that the relationship between the selected variables and the outcome scores is not straightforward or easily discernible. It is possible that other important factors or variables not included in the analysis may contribute significantly to the outcome scores. Consequently, relying solely on the "j_communication" and "m_experimental_testing" variables may not provide a comprehensive understanding of the outcome score prediction.

To improve the performance, several strategies can be considered. Firstly, further exploration of alternative machine learning algorithms could be beneficial, as different models may better handle the complexities present in the data. Additionally, feature engineering techniques could be applied to extract more informative features from the variables, potentially enhancing the model's predictive capabilities. Furthermore, collecting more comprehensive and diverse data may help uncover additional patterns and improve the overall performance of the model.

Decision Trees

Decision trees was the next simple approach to be taken. It involves splitting data at nodes to classify them at different levels of 'purity'. The gini score or index was used as the criteria for splitting because it quantified the randomness present in the data after each split which was the best choice to assist in determining feature importance. (Ceballos, 2021) Since decision trees operate on a relative scale, no normalisation of features needs to occur. This model was chosen because of the need to choose a classifier that could express feature importance. The parameters were optimised using the typical GridSearch cross validation method. The parameters chosen to be optimised were the minimum number of samples required for a split to occur and the maximum depth. The minimum number of samples taken before a leaf is reached (min_samples_leaf) was chosen as a tuning parameter because the goal of this project was to find the factors that affected outcome score the most and setting this parameter could assist in the selection process by optimising the amount of coverage of the data since decision trees are sensitive to the amount of data they handle. Additionally, the data set contained heavy biases towards the average score so this issue had to be tackled. Although the accuracy is affected by the depth at which a decision tree terminates, this parameter was not considered after the grid search produced a result because it recommended a maximum depth of 1 which would not have assisted in producing useful results. The decision tree generated is shown in figure 13 and displays the features that had a prominent effect on score. The accuracy of the algorithm when applied to the data set had an accuracy of approximately 36%. Since this algorithm is a supervised learning type, it requires a separated training and testing set which was randomly split in the same way it had been for other models to maintain the fairness in the comparison between other models.

Figure 14 : A visual representation of the decision tree used to classify interns



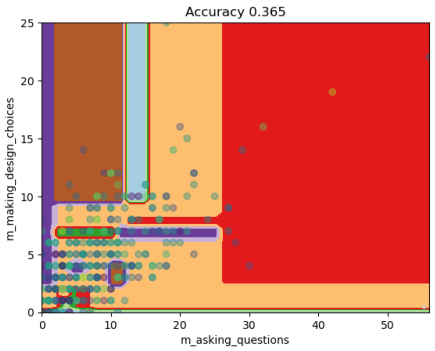
In figure 14, the important features were determined based on how low the Gini scores at each node were. Therefore, by this reasoning, some important features could be m_experimental_testing and m_making design_choices (as deduced from the decision tree). The gini scores are still extremely high in the decision tree plot, indicating strong signs of misclassification. To affirm that this is the case, a decision boundary plot with the accuracy shown can be generated. Additionally, there is no clear factor that explains the variance in outcome score, citing an inappropriate result for determining the most prominent variables influencing the target variable. The next step in this procedure was to gather the importance of all features as shown in figure 14.

Figure 15: The rankings of features by how much variance in the data they could account for.

Feature	Importance
m_making_design_choices	0.288
m_asking_questions	0.26
j_performance_parameters_requirements	0.182
j_customer_consultants_requests	0.099
m_experimental_testing	0.098
j_communication	0.069

The quantity of 'importance' of a feature was measured by what proportion of variability each feature could 'explain' in the data. The most important features were m_making_design_choices and m_asking_questions. Although all features had little effect individually on outcome score, both previously mentioned features had to be considered as they had relatively high feature importance. However, because of the sensitive nature of decision trees to noisy data, this assertion is not guaranteed.

Figure 16: A plot decision boundary for the decision tree classifier used



In figure 16, the model is very inaccurate because there are strong signs of overfitting. Evidence of this can be seen through the relatively small coloured regions. This suggests that perhaps, no relationship between the frequency of the types of messages sent exists at all.

A potential way to improve this model would be to allow GridSearchCV to cross validate over a wider range for each parameter, or to increase the number of cross validations. However, this would not necessarily be effective because of the extremely sensitive nature of decision trees to noise in the data set, leading to increasing randomness. The next sensible choice would be to utilise random forests to combat the primary issue of sensitivity to noisy data.

Random forests

The random forests classifier model was chosen to overcome the issues encountered by decision trees, such as sensitivity to noisy data and having a high variance. A random forest classifier algorithm with bootstrapping enabled, operates analogously to a series of repeated decision tree classifiers under different randomised conditions and retains the aggregate result of the repeated operations as the output. This also combats the issue of bias because the repetition of running the decision tree classifier causes the coverage of the data set to be even greater. The method chosen to tune the hyperparameters was also grid search cross validation because of the need to maintain consistency. The hyperparameters chosen for tuning were the number of estimators, maximum depth and minimum number of samples required before reaching a leaf. These hyperparameters were chosen to retain some consistency when comparing the performance of the random forests classifier algorithm with the decision trees algorithm. The parameter n_estimators, was included in the cross validation because it was a significant parameter that affected the learning ability of the model so it could not be dismissed (Fraj, 2017). The order of feature importance is shown in the table at figure 17.

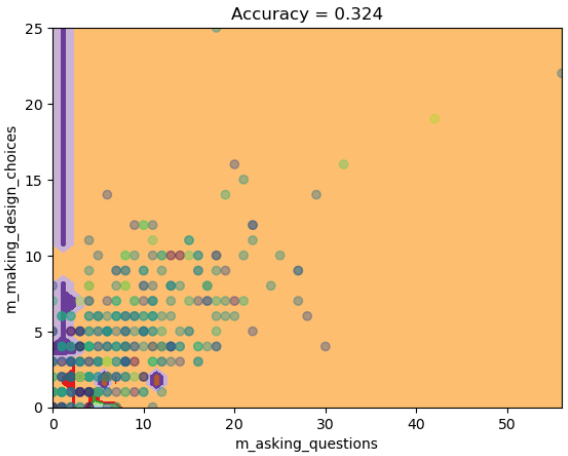
Figure 17: A table displaying importance of features

Feature	Importance
m_making_design_choices	0.239
m_asking_questions	0.218
m_experimental_testing	0.195
j_performance_parameters_requirements	0.185
j_customer_consultants_requests	0.1
j_communications	0.063

As observed in figure 17, all features do not seem to account for a significant proportion of variance in the data set which may be an issue when attempting to produce a meaningful insight. An issue with using random forests was that majority of interns' scores were now classified into a single target label as shown in figure 18, which likely occurred because of the repeated nature of random forests, as opposed to the non-iterative nature of decision trees, which naturally decreased variance in the potential targets that interns could be assigned to. As a result, accuracy

is reduced. Therefore, although random forests significantly reduced variance and overfitting, it produced unfruitful results in terms of classification accuracy.

Figure 18 : A plot decision boundary for the random forest classifier used



The model can be greatly improved by exploring more hyperparameters to tune during cross validation rather than performing a comparison between the results of a single decision tree classifier and the aggregated results of multiple decision trees. Altering the random state could have increased accuracy but undermined the fairness of the comparison between the results. A possible way to increase the accuracy of the model might also be to increase the number of cross validations during the optimisation process to stabilise the results of the GridSearchCV function.

Textual Analysis using Natural Language Processing

Natural language processing (NLP) is a field of AI that allows computers to interpret and understand human language through the use of computational linguistics and machine learning. NLP enables computers to perform tasks such as language translation, speech recognition and text summarisation. In this project, various NLP algorithms have been used including sentiment analysis, term frequency, topic modelling and part-of-speech tagging. These processes were conducted to determine whether the elements extracted could lead to predicting 'OutcomeScore' with high levels of accuracy.

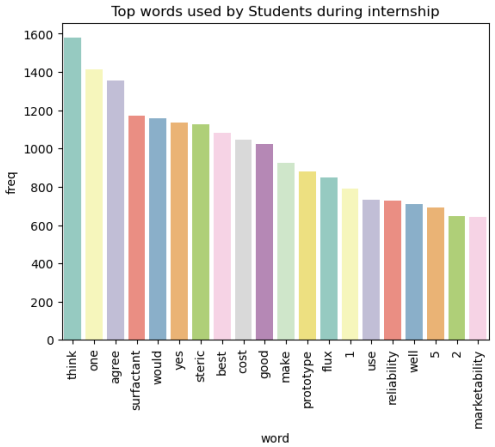
The first form of NLP that was explored was vectorisation. Vectorisation is the process of converting raw text data into numerical values that can be used in Machine Learning models. 4 different vectorisation methods were examined to see which method would achieve the highest level of accuracy. These methods were TF-IDF (Term Frequency-Inverse Document Frequency) vectorisation, Hashing Vectorisation (uses hash functions to convert each word into a fixed size vector), Count Vectorisation (turns each word into a frequency of counts) and finally doc2vec which is a model from the Gensim package. This model uses neural networks to create vectors that preserve similarity relationships (Hamdaoui, 2019). Using these 4 different methods, the first models were created. Firstly logistic regression was attempted resulting in relatively low accuracy

scores. Using logistic regression the doc2vec method produced the best results with a 0.356 in accuracy, 1.583 in RMSE and finally an R^2 of -0.0505. The RMSE and R^2 values were considerably better than the next best method which was TF-IDF which while had the same accuracy it had worse RMSE and R^2 scores. Random Forest classification was then attempted with n_estimator=50 and a max depth of 10. Using doc2vec, the random forest classifier produced better results than the logistic regression model with an accuracy of 0.358, RMSE of 1.578 and -0.0434 for the R^2. Given these results the random forest classifier was used for further calculations, along with the doc2vec method when text vectorisation was required.

The next technique that was explored was sentiment analysis. Sentiment analysis involves analysing text in order to determine the sentiment or emotional tone of a text. In this project sentiment analysis is used to classify a student's chat message as positive, negative or neutral. It assigns a numerical score ('sentimentScore') between -1 and 1, with a score >1 being positive sentiment , while < 1 is negative and a score around 0 is neutral (MonkeyLearn, 2018) . Performing some basic analysis it was identified that higher sentiment scores led to higher 'OutcomeScores' in general. Packages spaCy and textblob were used to create a pipeline that could be used to determine Sentiment Score for each chat message (Pattakos, 2021) . Using a random forest classifier again, a model was produced to predict 'OutcomeScore' based on the sentiment score. Given this model an accuracy of 0.350, an RMSE of 1.613 and an R^2 of -0.0899 was achieved. All these scores were worse than the scores achieved using the previous method. Both methods were then combined to see how this affected the model; the scores were much worse when the model was produced indicating that vectorisation and sentiment scores were not the best features that could be used to predict 'OutcomeScore'.

After sentiment analysis, the presence of keywords was then examined to evaluate whether they had an impact on predicting 'OutcomeScore'. To determine which words would be used and which were thought to possibly have greatest importance, POS-tagging (part-of-speech) was used to find the top 10 nouns used by students (Fardeen, 2021). Given the previous analysis of the highest frequency words in the text (figure. 19), after POS-tagging the words think, agree, surfactant, steric, cost, prototype, flux, use, reliability and marketability were all determined to be the most frequent nouns in the text. These words all appeared to be scientifically relevant to the internship topic and thus were the ones chosen for further analysis. Using a random forest classifier again to create our model, we achieved similar results to the sentiment analysis model. An accuracy of 0.345 was achieved along with a RMSE of 1.60 and R^2 of -0.0766. Using feature importance it was identified that the words 'surfactant' and 'use' had the highest importance when predicting 'OutcomeScore'. While the RMSE and accuracy were much the same for both models, the R^2 score was slightly better indicating that this model fits the data, while not well, better than using sentiment analysis.

Figure 19: Top total word frequencies



The final technique that was explored was Topic Modelling. The Latent Dirichlet Allocation (LDA) technique was used to extract topics from the chat messages. This technique groups together statistically similar words. To do this, a document-word matrix must first be created. After creating the matrix, the LDA model converts it into 2 other matrices: Document-Term matrix and a Topic-word matrix (Seth, 2022). For each message, topics are then weighted to assign topics. In this model, the lda_model was trained to have 10 topics. An example of one of the topics was topic 4 which contained words such as 'steric', 'think', 'best', 'vapour', 'hinder', 'would', 'prototype', '2', '4' and 'phase'. Using the random forest classifier, where the feature set consisted of the 10 topics and their probabilities for each chat message, a higher accuracy score of 0.358 was achieved than in previous attempts. Despite this higher accuracy, the RMSE was much higher than our previous models at 1.756. This is a tradeoff however as the R^2 score is much closer to 0 (the score achieved was -0.382) in this model than in previous models. It was identified that topic 6, which included many words that were used in the keyword frequency analysis such as 'surfactant', 'agree', 'cost' and 'reliability', was the most prominent influence on outcome score compared to the rest of the topics.

Having examined all these features separately, it was important to examine them together in order to create a more accurate model. In the feature set, features such as 'wordCount', 'messages_sent' by the student, 'sentimentScore', keyword frequencies and the topic probabilities were included. Vectorisation was originally included however it was discovered that this technique tended to impact the models in a negative way, as it created a very noisy feature set. As a result it was not included in any final modelling. Again the random forest classifier was used to predict the 'OutcomeScore'; using this model, an accuracy of 0.387 was achieved with these features in the feature set. This is much improved on all the above models. A better RMSE of 1.5419 and a positive R^2 score of 0.0035 was achieved, which was an improvement on previous models, although the R^2 score was still quite low. The decision tree classifier was used on previous models in an attempt to get better accuracy scores than the random forest classifier, to no success. A decision tree model was tried with the same feature set and achieved drastic improvements on the random forest classification model. Using this model, an accuracy score of 0.499 was achieved, however the increase in accuracy did not improve the RMSE (1.575) or the R^2 score (-0.0393); in fact they worsened. This indicated that the difference between the true score and the predicted were worse,

despite the accuracy being better. The importance of features was then examined and it was discovered that the variables that carried the highest importance were 'wordCount' with 0.025 and 'messages_sent' with 0.816, which was quite surprising as they had very little correlation with 'OutcomeScore' in prior analysis. Upon further analysis of the messages sent variable it can be seen that interns with low message counts (<20 messages sent) had low to average outcome scores, while those who sent between 20 and 50 messages produced the best Outcome Scores. Students that sent over 60 messages tended to get average scores (~3-4).

Given the outcomes of the discussed models, it can be concluded that 'OutcomeScore' cannot be predicted accurately using the features that were extracted. Negative R^2 scores were consistently produced which indicated that the model is performing worse than a simple linear model. The better accuracy values produced by the decision tree classification model, however worse RMSE and R^2 values indicate that they are overfitting the data. Despite this it was conclusive that the 'messages_sent' produced the most accurate model giving an accuracy of 0.513 when examined in a feature set of its own.

Final Model (Categorical & Textual Features)

Having taken 2 separate approaches to modelling thus far, one looking at the categorical features given in the original dataset, while the other extracting features directly from the chat messages. It was decided to combine the 2 approaches to produce a final model. Given the success of the decision tree classification model in predicting 'OutcomeScore' from the textual features, this model was used again and compared to a Random Forest Classification. When producing the Random Forest model we achieved accuracy, RMSE and R^2 results of 0.390, 1.536 and 0.0119 respectively which were the best scores that were able to be achieved using this model type. When the decision tree classification model was used, the results were much like the previous model where only the textual features were examined. Accuracy was much better at 0.507, however the RMSE and R^2 of 1.596 and -0.0680 respectively, slipped. Again examining feature importance it was clear that again 'messages_sent' along with 'word_count' were the most significant in predicting 'OutcomeScore'. In this model however, they were closely followed by 'sentimentScore'. The categorical variables had very little importance to the model.

Having completed the modelling, it is evident that improvements need to be made in order to produce more accurate models. Some improvements that could be made could include regularisation, using an L1 or L2 regularisation for example, which would reduce overfitting of the model. Cross validation methods such as GridSearchCV could also be used in order to determine the most optimal hyperparameters; this was used in previous models . Further feature engineering could also be performed in order to extract more features that may be optimal for predicting 'OutcomeScore'. Some potential features could include Emotion detection, counting the number of grammatical errors that a student made or looking further into POS-tagging.

Conclusion and Evaluation

It was found that no individual feature was directly responsible for influencing the performance of interns. Furthermore, all results produced contained high degrees of uncertainty therefore all insights generated from this report are to be received with a significant level of discretion. According to the results of the multivariate linear regression analysis, all other quantifiable variables have a moderate effect on outcome score suggesting that perhaps none of the feature variables observed had a strong correlation with interns' outcome. The decision trees and random forests models' results implied that making design choices and asking questions influenced interns' performance the most. However, since there was no major influencing factor on outcome score, this assertion was not guaranteed. Regarding the results produced by the support vector machine, it was found that communication was the most important variable whilst customer consultation requests had a minimal impact on outcome score, implying that communication amongst interns was a significant influence on their performance and that consultations did not have much of an impact on interns' ability to perform. However, these results were highly inconsistent because during the training of the model, it was found that asking questions had the most profound effect on determining outcome score whilst communication had the least effect on interns' performance, directly contradicting the results deduced during testing. Thus, the conclusions drawn from the support vector machine are highly ambiguous, due to this contradiction. The conclusions drawn from performing vectorisation were not considered because of the extreme presence of noisy data affecting its ability to produce insightful outcomes. Through keyword examination, it could be seen that the frequency of the noun 'surfactant' had the greatest relevance in predicting 'OutcomeScore'. This related directly to the Latent dirichlet allocation modelling (topic-modelling) which produced results that, although comparably insignificant, implied that the most influential topic on outcome score, relatively speaking, was topic 6 which consisted of words such as 'agree', 'surfactant', 'reliability', 'cost', 'reactivity', 'blood', 'best and 'cell'. Surfactant was included in this topic which indicates that intern's use of surfactant could be fairly significant. Combining all the above features into one final model, it was concluded that the most significant features in predicting 'OutcomeScore' were the number of messages sent, word count, 'SentimentScore' and the topics. The categorical features held very little importance when these other features related to textual analysis were used. This was surprising as it was originally thought that these categorical features would have the biggest impact, while it was thought that 'WordCount' would have very little impact.

Regarding the overall findings of the report, it was concluded that there were no outstanding categorical variables which affected the outcome of interns' performances. Upon conducting analysis on the content of the messages sent between interns, it was deduced that a significantly influential factor was the frequency of the usage of technical terms, relevant to kidney dialysis production. This implies that the performance of interns perhaps depended on their ability to retain and apply technical knowledge when communicating with mentors and interns in the process of creating a product. Another conclusion that could be made was that the quantity of messages influenced the performance of interns in such a way that interns who communicated less tended to achieve lower scores whilst interns who communicated with moderate frequency achieved better outcomes which implies that frequency of correspondence is indeed a key factor in determining performance. Finally, interns who expressed lengthy messages tended to achieve

average scores perhaps because these messages were convoluted or largely unreadable in the sense that fellow interns and mentors had neither the time nor the ability to comprehend and respond to these messages in such a way that was effective.

The models used could be improved in various ways. For the support vector machine, decision trees and random forests classifiers models, this included extending the range of each of the parameters explored during grid search cross validation and perhaps increasing the number of cross validations performed to stabilise the resultant parameters produced. Another suggestion to improve these models could be to include more hyperparameters. For all models, a suggestion for improvement could be to add more features to the feature set. Some features could include the evaluation of emotion, determining the number of grammatical errors made by an intern or looking at the different parts of speech that are most significant. Looking at the number of stop words used could also pose an interesting feature. Looking further into each intern as an individual may also present some questions to help predict their Outcome score.

References

Ceballos, F. (2019, February 23). *Scikit-Learn Decision Trees Explained*. Medium; Towards Data Science. https://towardsdatascience.com/scikit-learn-decision-trees-explained-803f3812290d

Fardeen, A. (2021, August 12). *Tutorial on Spacy Part of Speech (POS) Tagging*. MLK - Machine Learning Knowledge. https://machinelearningknowledge.ai/tutorial-on-spacy-part-of-speech-pos-tagging/

Fraj, M. B. (2017, December 21). *In Depth: Parameter tuning for Random Forest*. All Things AI. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d#:~:text=n_estimators%20represents%20the%20number%20of

Hamdaoui, Y. (2019, December 10). *TF(Term Frequency)-IDF(Inverse Document Frequency) from scratch in python* . Medium. https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558

Jensen, T. (2022, September 23). *Feature Importance for Any Model using Permutation*. Medium. https://medium.com/@T_Jen/feature-importance-for-any-model-using-permutation-7997b7287aa

Linguistic Features · spaCy Usage Documentation. (n.d.). Linguistic Features. https://spacy.io/usage/linguistic-features

Monkeylearn. (2018, June 20). *Sentiment Analysis: Nearly Everything You Need to Know | MonkeyLearn*. MonkeyLearn. https://monkeylearn.com/sentiment-analysis/

Nephrotex. (n.d.). Www.virtualinterns.org. Retrieved May 21, 2023, from https://www.virtualinterns.org/nephrotex/

Overfitting and Underfitting. (2017, November 26). Winder.ai. https://winder.ai/overfitting-and-underfitting/

Pattakos, A. (2021, February 28). *Aspect-Based Sentiment Analysis Using Spacy & TextBlob*. Medium. https://towardsdatascience.com/aspect-based-sentiment-analysis-using-spacy-textblob-4c8de3e0d2b9

Selvaraj, N. S. (2022, October 5). *Hyperparameter Tuning Using Grid Search and Random Search in Python*. KDnuggets. https://www.kdnuggets.com/2022/10/hyperparameter-tuning-grid-search-random-search-python.html

Seth, N. (2021, June 28). *Topic Modeling and Latent Dirichlet Allocation (LDA) using Gensim*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/part-2-topic-modeling-and-latent-dirichlet-allocation-lda-using-gensim-and-sklearn/#:~:text=Latent%20Dirichlet%20Allocation%20(LDA)%20is

ADS2001-ReportAppendix

May 28, 2023

1 Appendix

1.0.1 Import required Library and Packages

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,recall_score,precision_score
from sklearn.preprocessing import MinMaxScaler

from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_curve
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVC
from sklearn.metrics import r2_score, mean_squared_error,␣
 ↪mean_absolute_error,accuracy_score,recall_score,precision_score,confusion_matrix
from sklearn.model_selection import cross_validate, RepeatedKFold, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso # models we are␣
 ↪going to use
from sklearn.inspection import permutation_importance

import statsmodels.formula.api as sm

from logitplots import plt_confusion_matrix, plt_decision_boundaries,␣
 ↪plt_correlation_matrix

import spacy
from spacytextblob.spacytextblob import SpacyTextBlob
```

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.corpus import wordnet
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer

stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.porter.PorterStemmer()

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

import gensim
from gensim import corpora, models
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize

import warnings
warnings.filterwarnings("ignore")#ignore warnings

RS=42#set consistent random state for all model testing
```

1.0.2 Dataset Exploration

[ ]:
```
df=pd.read_csv('VI_data.csv',encoding='latin1')
df=df.drop(df.columns[0],axis=1)
chatData = df
```

[ ]:
```
# Explore the data types of all our columns

chatData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19180 entries, 0 to 19179
Data columns (total 16 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   userIDs                               19180 non-null  int64
 1   implementation                        19180 non-null  object
 2   Line_ID                               19180 non-null  int64
 3   ChatGroup                             19180 non-null  object
 4   content                               19180 non-null  object
 5   group_id                              19180 non-null  int64
 6   RoleName                              19177 non-null  object
 7   roomName                              19180 non-null  object
 8   m_experimental_testing                19180 non-null  int64
```

```
 9   m_making_design_choices                19180 non-null  int64
 10  m_asking_questions                     19180 non-null  int64
 11  j_customer_consultants_requests        19180 non-null  int64
 12  j_performance_parameters_requirements  19180 non-null  int64
 13  j_communication                        19180 non-null  int64
 14  OutcomeScore                           19180 non-null  int64
 15  wordCount                              19180 non-null  int64
dtypes: int64(11), object(5)
memory usage: 2.3+ MB
```

[ ]:
```
# Examine dimensionality of the raw data

print(chatData.shape)
```

```
(19180, 16)
```

[ ]:
```
# Any missing values? If yes -> must delete or impute

chatData.isnull().sum()
```

[ ]:
```
userIDs                                0
implementation                         0
Line_ID                                0
ChatGroup                              0
content                                0
group_id                               0
RoleName                               3
roomName                               0
m_experimental_testing                 0
m_making_design_choices                0
m_asking_questions                     0
j_customer_consultants_requests        0
j_performance_parameters_requirements  0
j_communication                        0
OutcomeScore                           0
wordCount                              0
dtype: int64
```

Since the proportion of empty values is insignificant, dropping these values seems appropriate.

[ ]:
```
# Descriptive Statistics

stats=chatData.describe().round(4)
stats
```

[ ]:
```
           userIDs     Line_ID    group_id  m_experimental_testing  \
count  19180.0000  19180.0000  19180.0000              19180.0000
mean     202.4332   9592.7938      3.9167                  0.0287
std      118.3552   5537.8007      1.3979                  0.1670
```

```
min        1.0000      1.0000      2.0000                  0.0000
25%       96.0000   4796.7500      3.0000                  0.0000
50%      204.0000   9593.5000      4.0000                  0.0000
75%      317.0000  14388.2500      5.0000                  0.0000
max      393.0000  19183.0000      6.0000                  1.0000

       m_making_design_choices  m_asking_questions  \
count                19180.0000          19180.0000
mean                     0.1029              0.1870
std                      0.3038              0.3899
min                      0.0000              0.0000
25%                      0.0000              0.0000
50%                      0.0000              0.0000
75%                      0.0000              0.0000
max                      1.0000              1.0000

       j_customer_consultants_requests  j_performance_parameters_requirements  \
count                      19180.0000                               19180.0000
mean                           0.0181                                   0.0522
std                            0.1335                                   0.2225
min                            0.0000                                   0.0000
25%                            0.0000                                   0.0000
50%                            0.0000                                   0.0000
75%                            0.0000                                   0.0000
max                            1.0000                                   1.0000

       j_communication  OutcomeScore   wordCount
count      19180.0000    19180.0000  19180.0000
mean           0.0211        3.7416     12.4895
std            0.1436        1.4648     14.1172
min            0.0000        0.0000      1.0000
25%            0.0000        3.0000      4.0000
50%            0.0000        4.0000      9.0000
75%            0.0000        4.0000     17.0000
max            1.0000        8.0000   1032.0000
```

1.0.3 Variables in the Nephrotex Dataset:

The dataset contains the chat records from 15 implementations of Nephrotex. They are labelled with the presence and absence of various concepts in engineering discourse relating to design moves and design justifications. We want to model a students perfomance on the report based on what they have discussed within the chat throughout the internship.

- `UserIDs` = a unique id for each student
- `implementation` = id for each implementation of nephrotex (there were 15 implementations)
- `Line_ID` = unique Id for the chat utterance
- `ChatGroup` = id for the team the chatter was in. (String values are the teams from the first half of the internship, numeric are the teams from the second part of the intership - interns

swapped teams halfway through)
- `content` = content of the chat utterance
- `group_id` = numerical id for the team the chatter was in
- `RoleName` = Chatter was a **Mentor** or **Player**
- `roomName` = the internship activity that the chatter was participating in
- `OutcomeScore` = A mark from 0 -> 8 indicating quality of the chatter's final design report. (Not everyone in the same group got the same score)
- `wordCount` = no. of words in chat content

Design Moves (Dummy Variables) - `m_experimental_testing` = talks about using experimental techniques to understand the technical features of the design - `m_making_design_choices` = talks about choosing a specification or characteristic for a design. - `m_asking_questions` = asks a question

Design justifications (Dummy Variables) - `j_customer_consultants_requests` = states whether they should meet or exceed stakeholder requests - `j_performance_parameters_requirements` = refers to performance parameters or experimental results - `j_communication` = refers to how it can facilitate communication among the engineers

Description of room names Each room name represents the phases of a team's project. A brief description of each of them is given below in the chronological order of the project's workflow.

Introduction and Workflow Tutorial with Entrance Interview: This room serves as an induction for interns and allows both mentors and interns get to know each other

Background research on dialysis: This room's purpose is for interns to discuss the optimal existing treatments for patients' kidneys so that the team's product can be inspired by these ideas for treating kidney failure. Dialysis describes the treatment used to act as a replacement for a patient's kidneys with regards to functioning, in the event of kidney failure.

Choose consultants to analyze: This room enables interns to discuss which external advisors are the most effective at conveying information about patients' condition with regards to their physiological state, so that their products can best meet patients' needs.

Summarize internal consultant requirements: This room enables interns to discuss how they can satisfy patients' needs based on the advice they have been given by the consultants they had previously chosen

Graphing Surfactant Data: This room's purpose is to visualise the effects of different types of surfactants on patients' kidneys. The general role of a surfactant is to reduce the surface tension of blood vessels in the lungs and kidneys so that pathogenic fluids and waste can be removed from the body.

Reflection team discussion of surfactants: This room allows the team to discuss the results of the surfactants' effecitveness in improving kidney function.

Individuals design 5 prototypes: This room serves as an ideation space for individuals to brainstorm potential ideas. By the end of this phase, individuals are required to realise at least 5 ideas in the form of prototype designation.

Individual analysis of first batch: Individuals analyse and reflect upon the advantages and disadvantages of the their prototypes.

Team designs batch using 1 material: This room contains the team's final design to be submitted as part of the first round of designs for Nephrotex.

Reflection team discussion of first batch results: This room allows the team to discuss the effectiveness of their prototype submission and how to improve in later attempts.

1.0.4 Data Wrangling & Processing

[ ]:
```
chatData=pd.read_csv('VI_data.csv',encoding='latin1')
```

[ ]:
```
def swap_columns(df, col1, col2):#define misc. function to swap columns in␣
 ↪dataframe
    col_list = list(df.columns)
    x, y = col_list.index(col1), col_list.index(col2)
    col_list[y], col_list[x] = col_list[x], col_list[y]
    df = df[col_list]
    return df
```

[ ]:
```
#creating a player only data frame
chatData=swap_columns(chatData,'OutcomeScore','wordCount')
chatData1=chatData.drop(chatData.columns[0],axis=1)#drop unamed column
chatData1=chatData1.
 ↪drop(['ChatGroup','content','roomName','Line_ID'],axis=1)#remove text data␣
 ↪('content')

mask=chatData['RoleName']=='Mentor'#set condition to mentor data only
chatData1=chatData1.drop(chatData[mask].index)#drop all mentor data only

chatData1=chatData1.dropna()#drop the Nan value

teams=chatData1.groupby(['userIDs','implementation','group_id']).sum()#group␣
 ↪players by id and get sum of each quantitative stat

scores=chatData1.
 ↪drop_duplicates(subset=['userIDs','implementation','group_id'])#get score of␣
 ↪each player
teams['Outcome_Score']=scores['OutcomeScore'].values#add outcome score column␣
 ↪in data set so each player has a score from 0-7 instead of cumulative
teams=teams.drop(teams.columns[7],axis=1)#drop original cumulative outcome␣
 ↪score column
teams=teams.rename(columns={'Outcome_Score':'OutcomeScore'})

#creating mentor only data frame repeating same steps
chatData2=chatData.drop(chatData.columns[0],axis=1)
chatData2=chatData2.drop(['ChatGroup','content','roomName','Line_ID'],axis=1)
mask=chatData2['RoleName']=='Player'
chatData2=chatData2.drop(chatData2[mask].index)
```

```
chatData2=chatData2.dropna()

men_teams=chatData2.groupby(['userIDs','implementation','group_id']).sum()
men_teams=men_teams.drop(["OutcomeScore"],axis=1)
```

[ ]:
```
teams.head()
```

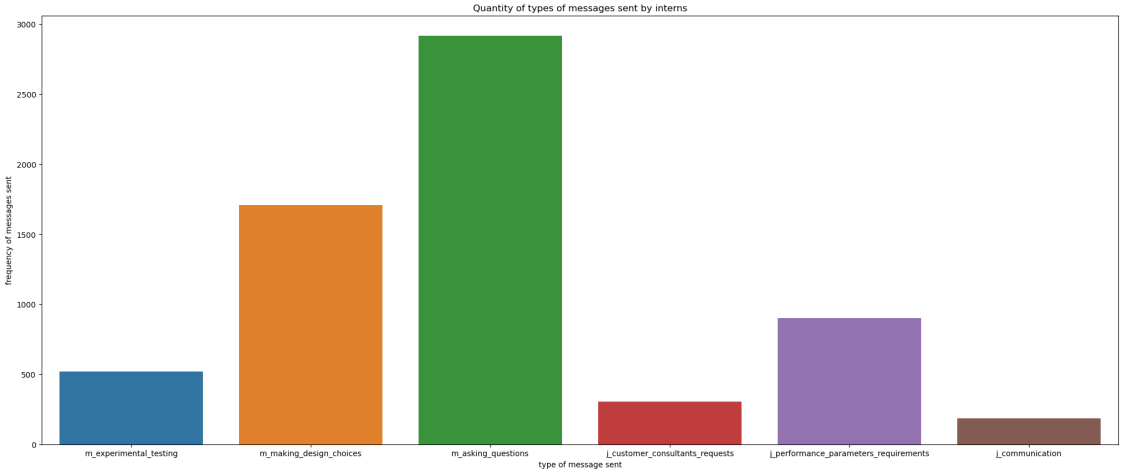
[ ]:

userIDs	implementation	group_id	m_experimental_testing	m_making_design_choices	m_asking_questions	j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	wordCount	OutcomeScore
2	a	2	2	4	17	0	4	0	704	4
3	a	2	1	4	3	0	1	0	157	4
4	a	2	2	2	3	1	5	0	349	4
5	a	2	0	0	6	0	2	0	342	2
6	a	2	0	2	7	1	3	0	399	2

1.0.5 Exploratory Data Analysis & Visualisation

The data was wrangled by using the identifier columns, such as `userID`, `implementation` and `group_id` to separate interns into different instance objects with a unique set of quantitative attributes. For example, after the wrangling process, an intern would be identifed by the aforementioned keys and contained the frequency distribution of the types of messages they sent, the total number of words they typed and most importantly, their outcome score. To get a more complete and general picture of the most common and least common types of messages sent by interns over the whole program's period of operation, a bar plot has been generated.

[ ]:
```
message_sum_graph=pd.DataFrame(teams.sum()).
  ↪drop(['OutcomeScore','wordCount'],axis=0).transpose()
f, ax = plt.subplots(figsize=(25, 10))
sns.barplot(data=message_sum_graph)
plt.title('Quantity of types of messages sent by interns')
plt.ylabel('frequency of messages sent')
plt.xlabel('type of message sent')
plt.show()
```

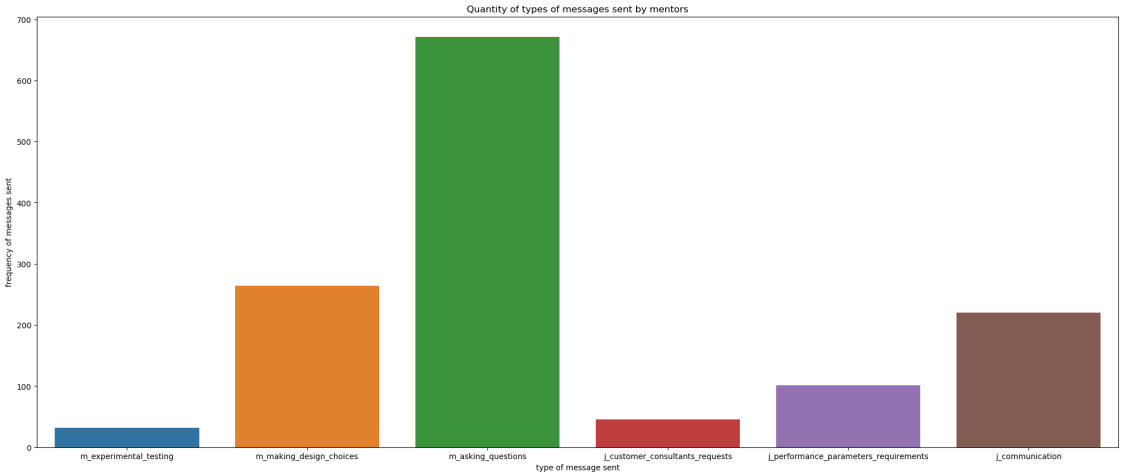


According to the graph above, the most common type of message sent by interns was the `m_asking_questions` type of message, which may imply that interns mainly used the chatting app to communicate with mentors about the aspects of the project that they were uncertain of. Interestingly, the rarest types of messages sent were `j_communication` and `j_customer_consultant_requests`, which could indicate that interns either were not very coordinated when it came to expressing ideas in a team, or used other means of communication, and did not consider clients' needs when given the opportunity to receive advice from consultants or had limited contact hours with consultants. Both events could explain the presence of a significant volume of interns who performed below average, as shown in the next figure.

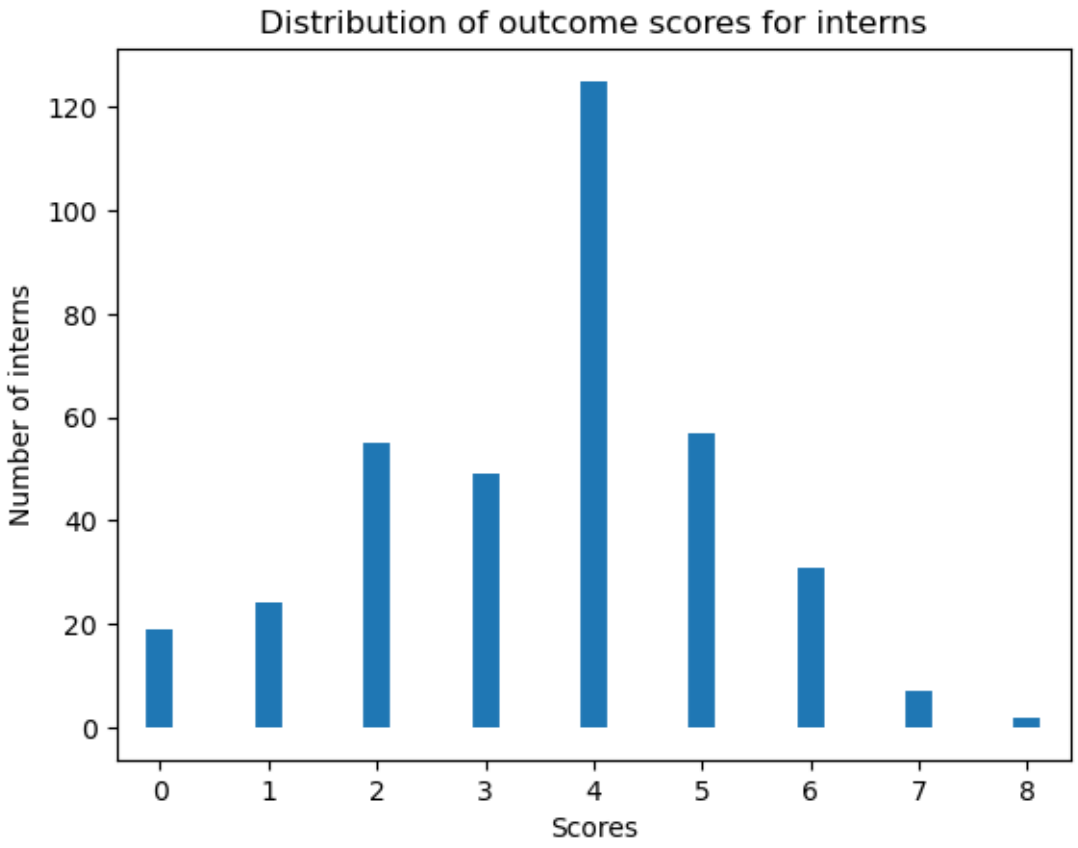
The process is repeated for the mentors to see if there is an insightful relationship between the distributions of messages. As observed, these proportional distributions are very similar, perhaps except for the frequency of communicative messages, which appears to be moderately higher because

mentors tend to convey a higher volume of miscellaneous messages to interns to ensure the program runs smoothly. An example would be introducing themselves to interns.

[ ]:
```
message_sum_graph=pd.DataFrame(men_teams.drop('wordCount',axis=1).sum()).
↪transpose()
f, ax = plt.subplots(figsize=(25, 10))
sns.barplot(data=message_sum_graph)
plt.title('Quantity of types of messages sent by mentors')
plt.ylabel('frequency of messages sent')
plt.xlabel('type of message sent')
plt.show()
```

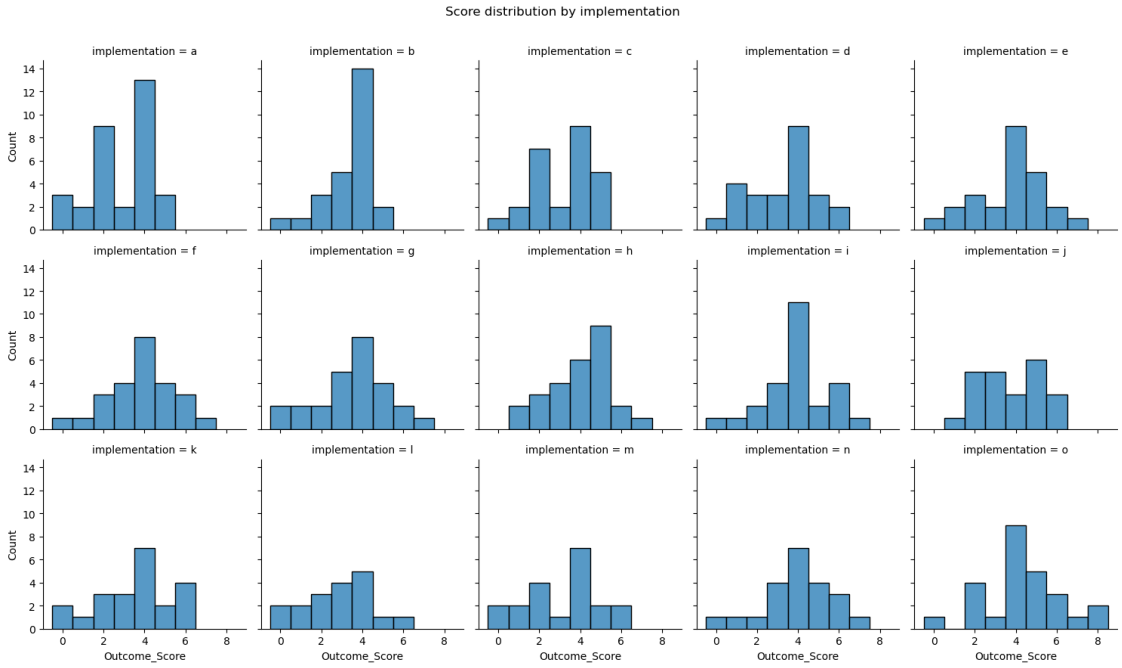


[ ]:
```
dist_scores=teams['OutcomeScore']
data=np.array(dist_scores)
values, counts = np.unique(data, return_counts=True)
plt.vlines(values, 0, counts, color='C0', lw=10)
plt.title('Distribution of outcome scores for interns')
plt.xlabel('Scores')
plt.ylabel('Number of interns')
plt.show()
```



[ ]:
```
sort_teams=chatData1.groupby(['userIDs','implementation',]).sum()
scores=chatData1.drop_duplicates(subset=['userIDs'])
sort_teams['Outcome_Score']=scores['OutcomeScore'].values
sort_teams=sort_teams.drop(teams.columns[7],axis=1)
sort_teams=sort_teams.reset_index()
```

[ ]:
```
g = sns.FacetGrid(sort_teams, col="implementation",col_wrap=5)
g.map(sns.histplot,'Outcome_Score',discrete=True)
g.fig.suptitle('Score distribution by implementation')
g.fig.subplots_adjust(top=0.9)
plt.show()
```

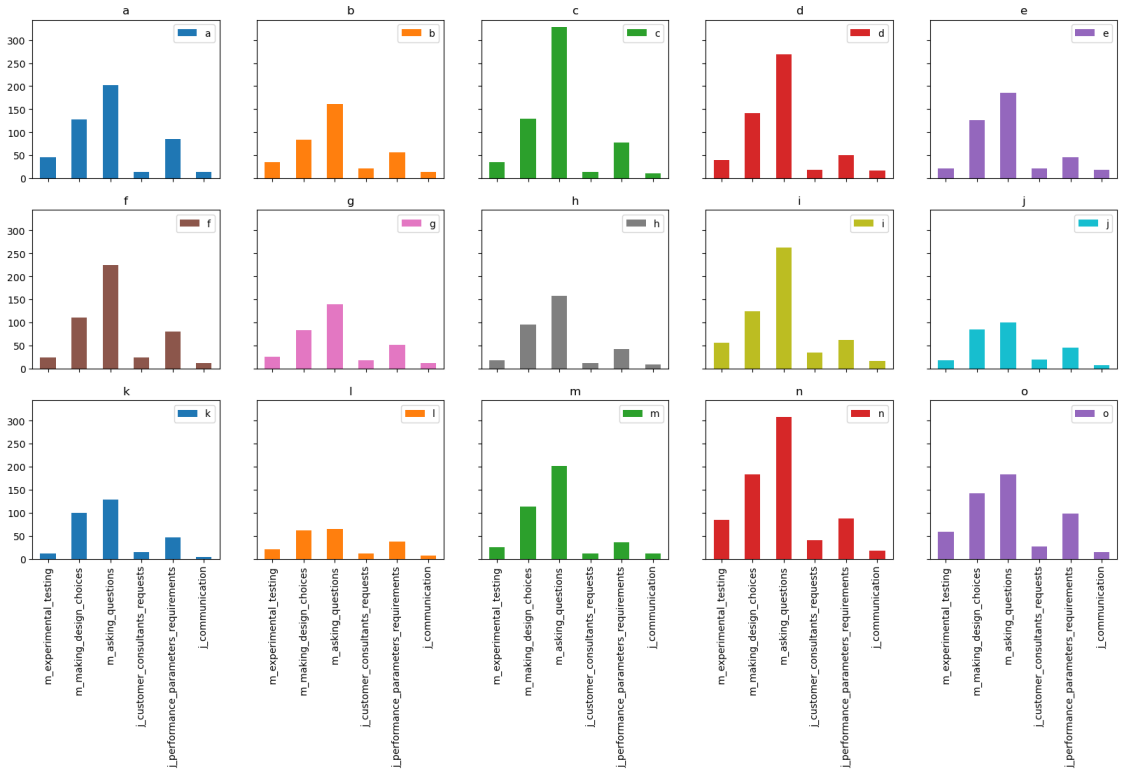


[ ]:
```
implementation_variables = teams.reset_index()
implementation_variables = implementation_variables.drop(['userIDs',␣
 ↪'group_id'], axis=1)

implementation_variables = implementation_variables.groupby(['implementation']).
 ↪sum()

implementation_variables = implementation_variables.reset_index()

# implementation_variables = implementation_variables.drop(['OutcomeScore',␣
 ↪'wordCount'], axis=1)

implementation_variables = implementation_variables.
 ↪drop(['wordCount','OutcomeScore'], axis=1)
```

[ ]:
```
implementation_graph=implementation_variables.set_index('implementation').T
implementation_graph.plot(kind='bar', subplots=True, layout=(3, 5),␣
 ↪figsize=(20, 10), sharey=True, title=['a', 'b', 'c', 'd', 'e', 'f', 'g',␣
 ↪'h', 'i', 'j', 'k', 'l', 'm', 'n',
       'o'])
```

[ ]:
```
array([[<Axes: title={'center': 'a'}>, <Axes: title={'center': 'b'}>,
        <Axes: title={'center': 'c'}>, <Axes: title={'center': 'd'}>,
        <Axes: title={'center': 'e'}>],
       [<Axes: title={'center': 'f'}>, <Axes: title={'center': 'g'}>,
```

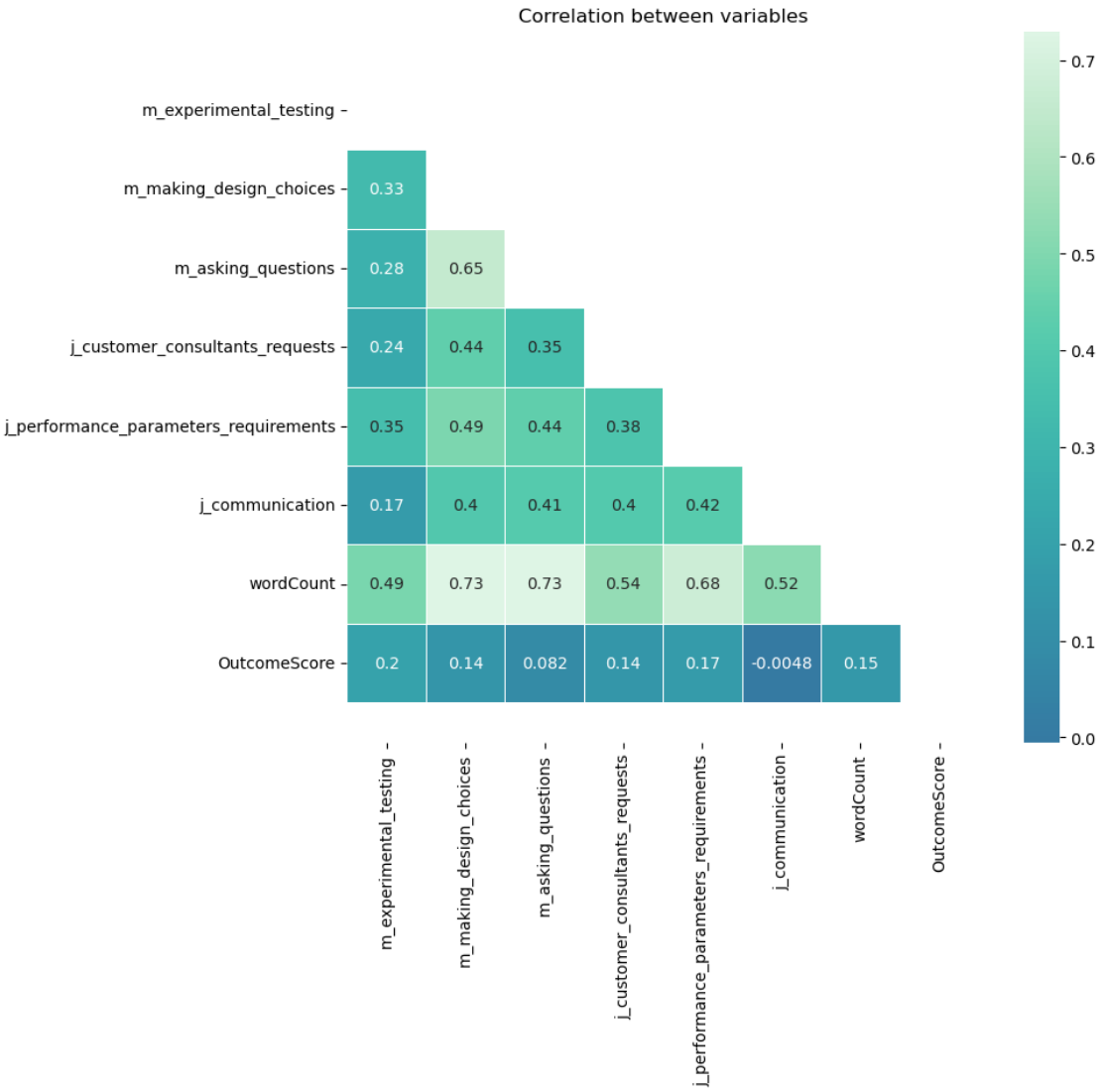
```
       <Axes: title={'center': 'h'}>, <Axes: title={'center': 'i'}>,
       <Axes: title={'center': 'j'}>],
      [<Axes: title={'center': 'k'}>, <Axes: title={'center': 'l'}>,
       <Axes: title={'center': 'm'}>, <Axes: title={'center': 'n'}>,
       <Axes: title={'center': 'o'}>]], dtype=object)
```



[ ]:
```
def plt_correlation_matrix(corrs):
    '''Uses the seaborn heatmap to plot the correlation matrix of a pandas␣
 ↪dataframe'''
    # as this is a symmetric table, set up a mask so that we only plot values
    # below the main diagonal
    mask = np.triu(np.ones_like(corrs, dtype=np.bool))
    f, ax = plt.subplots(figsize=(10, 8)) # initialise the plots and axes
    # plot the correlations as a seaborn heatmap, with a colourbar
    sns.heatmap(corrs, mask=mask, center=0, annot=True, square=True,␣
 ↪linewidths=.5, cmap='mako')
    # do some fiddling so that the top and bottom are not obscured
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
```

[ ]:
```
plt_correlation_matrix(teams.corr())
plt.title('Correlation between variables')
```

```
plt.show()
```



The most significant factors influencing `OutcomeScore` appear to be `m_experimental_testing` and `j_performance_parameters_requirements` whilst `j_communication` and `m_asking_questions` appear to have the least effect on score, relatively speaking, albeit the correlation for all factors is low. Therefore, more concrete evidence is required and can be found using feature importance functions of different types of models. A concerning result in this correlation matrix pertains to the fact that there may be some colinearity issues particularly because the correlation between the other variables is high in some cases, making it difficult to distinguish the individual effects of each variable on outcome score.

[ ]:
```
teams=teams.drop(['wordCount'],axis=1)
X=teams.drop(['OutcomeScore'],axis=1)
```

```
y=teams['OutcomeScore']
nX=(X-X.mean())/X.std()
```

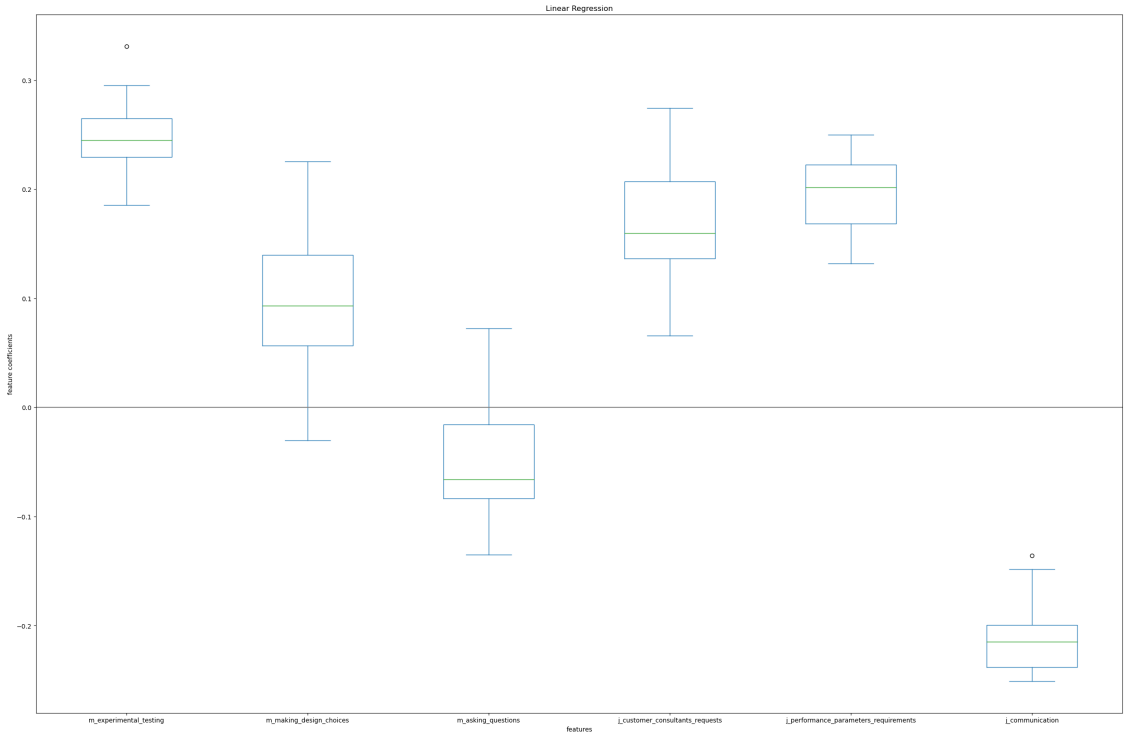
1.1 Categorical Analysis

1.1.1 Linear Regression

The first approach was to normalise all features and map all potential relationships between each feature and outcome score to compare significance of features in affecting scores. The main issues with this approach will be outlined below.

[ ]:
```
X_train, X_test, y_train, y_test =train_test_split(nX,y,test_size=0.2,␣
 ↪random_state=RS)
linear = LinearRegression()
linear.fit(X_train,y_train)
y_pred = linear.predict(X_test)
```

[ ]:
```
feature_names = X.columns.tolist()
feature_scores = cross_validate(
    linear, nX, y,
    cv=RepeatedKFold(n_splits=5, n_repeats=5, random_state=np.random.
 ↪RandomState(RS)),
    return_estimator=True,
)
coefs = pd.DataFrame([est.coef_ for est in feature_scores['estimator']],␣
 ↪columns=feature_names)
ax = coefs.plot(kind='box',figsize=(30,20))
plt.title('Linear Regression')
plt.xlabel('features')
plt.ylabel('feature coefficients')
plt.axhline(y=0, color='.5')
plt.subplots_adjust(left=.1)
plt.show()
```



The graph above demonstrates that the coefficients have moderate variability for the all variables. The variable, `j_communication` appears to have the worst effect on `OutcomeScore` but this finding may not be as statistically significant as the other variables' effects on `OutcomeScore` because the correlation between `j_communication` and `OutcomeScore` is relatively low compared to other variables, as seen in the correlation matrix. `m_experimental_testing`, the variable with the highest correlation to `OutcomeScore` and seems to have positive effect on outcome score.

[ ]:
```
pd.DataFrame({
    "R^2": {
        "Results": r2_score(y_test,y_pred)
    },
    "RMSE": {
        "Results": mean_squared_error(y_test,y_pred, squared=False),
    },
    "MAE": {
        "Results": mean_absolute_error(y_test,y_pred),
    }
})
```

[ ]:

	R^2	RMSE	MAE
Results	-0.112264	1.623714	1.371916

According to the accuracy metrics, the R^2 in particular, linear regression is an inappropriate choice to model feature importance because R^2 scores close to 0 indicate extremely poor fitting results to the data. The reasons for this may be that predicting scores may not be a regression

problem or the data is unlikely to be linear. Therefore, classification algorithms may need to be used instead or another type of regression model may need to be explored.

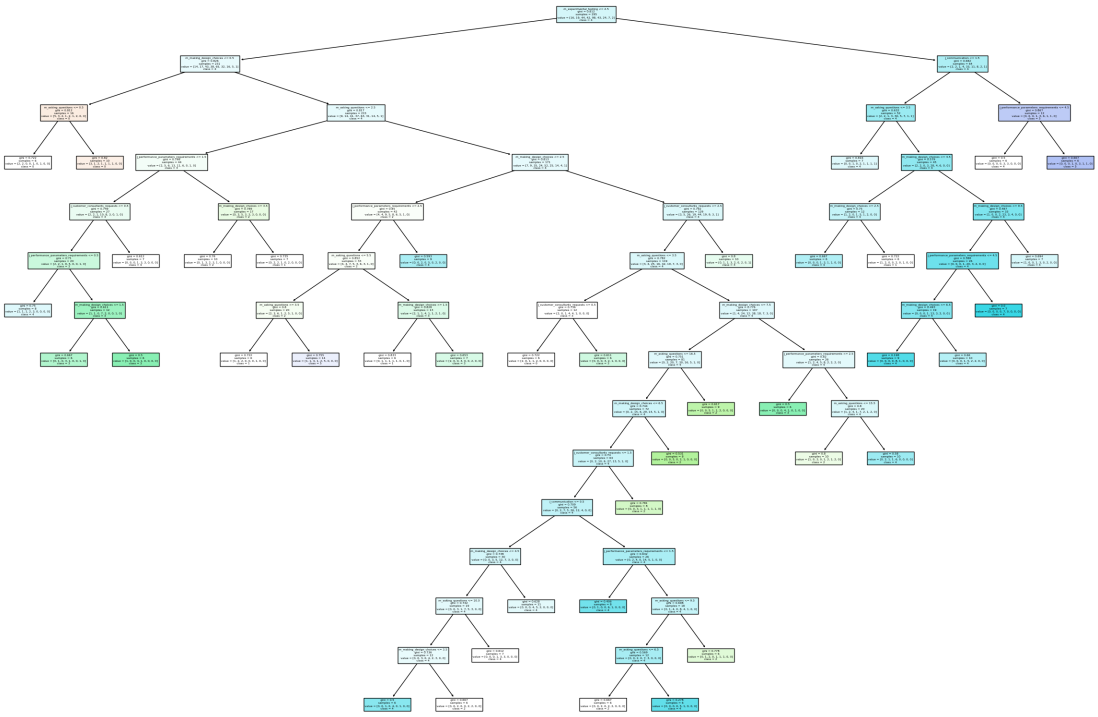
1.1.2 Decision Trees (Classifier)

Decision trees was the next simple approach to be taken. It involves splitting data at nodes to classify them at different levels of 'purity'. The gini score or index is used to quantify the randomness present in the data after each split.Therefore, a higher gini index indicates less purity. Since decision trees operate on a relative scale, no normalisation of features needs to occur. This model was chosen because of the need to choose a classifier that could express feature importance. The model was also ideal because it operated in a relative scope meaning no normalisation was required.

[ ]:
```
classes=sorted(set(y))
for i in range(len(classes)):
    classes[i]=str(classes[i])
X=teams.drop(['OutcomeScore'],axis=1)
y=teams['OutcomeScore']
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.2,␣
 ↪random_state=RS)

parameters={'min_samples_leaf':range(1,10)
           }
dtc=DecisionTreeClassifier(random_state=RS)
clf = GridSearchCV(estimator=dtc, param_grid=parameters,cv=3,scoring='accuracy')
clf.fit(X_train,y_train)
print(f"GridSearchCV parameters: {clf.best_estimator_}")
```

```
GridSearchCV parameters: DecisionTreeClassifier(min_samples_leaf=6,
random_state=42)
```

[ ]:
```
leaf=6

dtc=DecisionTreeClassifier(min_samples_leaf=leaf,random_state=RS)
dtc.fit(X_train,y_train)
Y_pred = dtc.predict(X_test)
fig, ax = plt.subplots(figsize=(30,20))
plot_tree(dtc,
 filled = True,
 rounded = False,
 class_names = classes,
 feature_names = X.columns
 );
plt.show()
```



[ ]:
```
features_list=list(X.columns)
feature_importances=dtc.feature_importances_
for importance,feat in zip(feature_importances,features_list):
    print(f"{feat} feature importance: {importance}")
```

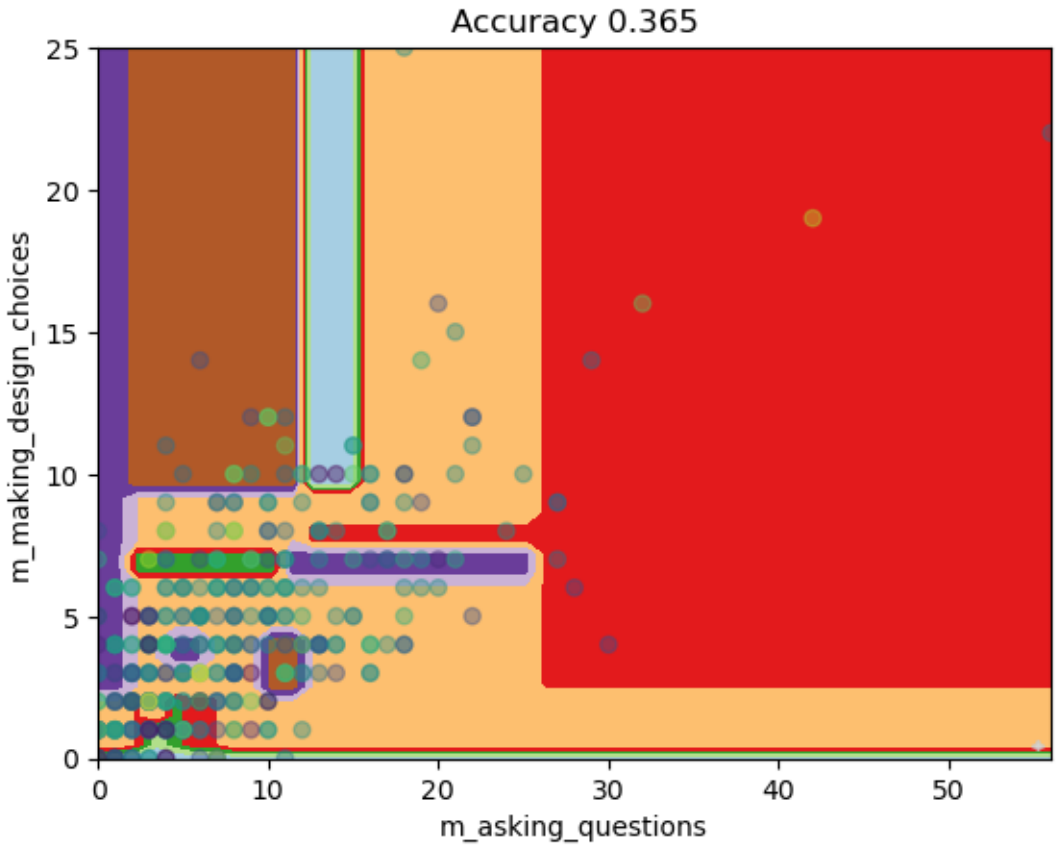
```
m_experimental_testing feature importance: 0.09753695798284226
m_making_design_choices feature importance: 0.2879207120821278
m_asking_questions feature importance: 0.26459964454972795
j_customer_consultants_requests feature importance: 0.09900328721606022
j_performance_parameters_requirements feature importance: 0.18199960201347118
j_communication feature importance: 0.06893979615577062
```

The gini scores are still extremely high in the decision tree plot, indicating strong signs of misclassification. To affirm that this is the case, a decision boundary plot with the accuracy shown can be generated. Additionally, there is no clear factor that explains the variance in outcome score, citing an inappropriate result for determining the most prominent variables influencing the target variable.

[ ]:
```
feature1='m_asking_questions'
feature2='m_making_design_choices'
X = X[[feature1,feature2]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2␣
 ↪,random_state=RS)
dtc=DecisionTreeClassifier(min_samples_leaf=leaf,criterion='gini',random_state=RS)
```

```
dtc.fit(X_train,y_train)
Y_pred=dtc.predict(X_test)
plt.title('Accuracy {0}'.format(np.round(accuracy_score(y_test, Y_pred),3)))
plt.xlabel(feature1)
plt.ylabel(feature2);
plt.legend(loc="lower right", borderpad=0, handletextpad=0)
xx, yy = np.meshgrid(np.linspace(X.m_asking_questions.min(),X.
 ↪m_asking_questions.max()),
                     np.linspace(X.m_making_design_choices.min(),X.
 ↪m_making_design_choices.max()))
plt_decision_boundaries(dtc,xx,yy)
plt.scatter(X[feature1], X[feature2], c=y, alpha=0.4);
#plot legend here
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



As observed in the decision boundary,there is significant evidence of overfitting. However, there are multiple regions classifying outcome scores to students, which is somewhat promising. A method to combat the effect of overfitting, a random forest can be used and optimised over a similar scope

of parameters.

1.1.3 Random Forests

To combat the issue of having high variance, previously posed by decision trees, the next model used will be random forest classifiers which will lower the variance as much as possible. The random forest classifier algorithm operates analogously to a series of repeated decision tree classifiers under different randomised conditions and retains the aggregate result of the repeated operations as the output. This combats the issue of bias because of the repeating mechanism running the decision tree classifier on a greater area of the data set. GridSearchCV will be the optimisation mechanism used.

[ ]:
```
X=teams.drop(['OutcomeScore'],axis=1)
y=teams['OutcomeScore']
X_train, X_test, y_train, y_test =train_test_split(X,y,train_size=0.
 ↪8,random_state=RS)
grid_space={'n_estimators':[10,100,200],
            'min_samples_leaf':range(1,10)
           }#cross validation process
rf=RandomForestClassifier(random_state=RS)
grid = GridSearchCV(estimator=rf,param_grid=grid_space,cv=3,scoring='accuracy')
model_grid = grid.fit(X_train,y_train)
print('Best hyperparameters are: '+str(model_grid.best_params_))
```

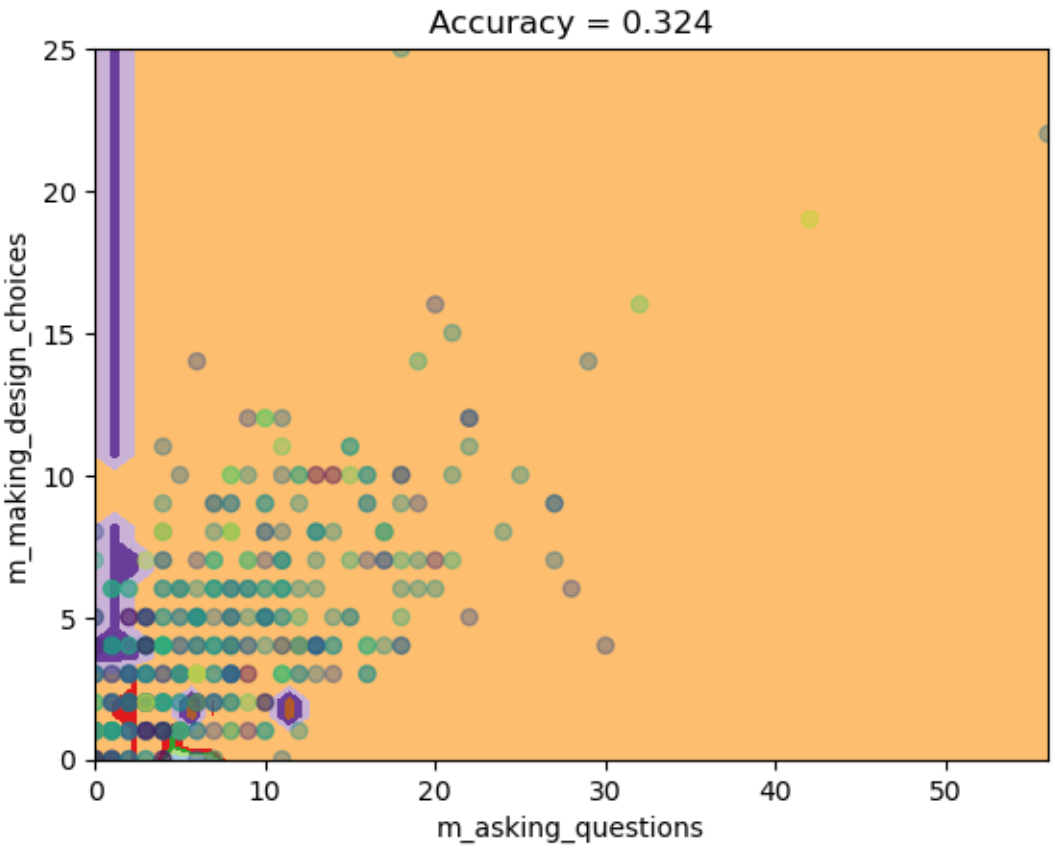
```
Best hyperparameters are: {'min_samples_leaf': 8, 'n_estimators': 100}
```

[ ]:
```
samples_leaf=8
estimators=100


df_dtc = RandomForestClassifier(n_estimators=estimators,
                                min_samples_leaf=samples_leaf,
                                random_state=RS)
df_dtc = df_dtc.fit(X_train, y_train)
Y_pred = df_dtc.predict(X_test)
for name, score in zip(X.columns, df_dtc.feature_importances_):
    print(name,np.round(score,3))
```

```
m_experimental_testing 0.195
m_making_design_choices 0.239
m_asking_questions 0.218
j_customer_consultants_requests 0.1
j_performance_parameters_requirements 0.185
j_communication 0.063
```

None of the factors are significant in determining outcome score which may be an issue when producing results since random forests utilise a series of decision trees to reduce bias and have more definitively asserted that no factors affect outcome score.

[ ]:
```
X=teams.drop(['OutcomeScore'],axis=1)
y=teams['OutcomeScore']
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.
 ↪2,random_state=RS)
feature1='m_asking_questions'
feature2='m_making_design_choices'
X = X[[feature1,feature2]]
y=np.array(y).flatten()
X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪random_state=RS,test_size=0.2)

xx, yy = np.meshgrid(np.linspace(X.m_asking_questions.min(),X.
 ↪m_asking_questions.max()),
                     np.linspace(X.m_making_design_choices.min(),X.
 ↪m_making_design_choices.max()))
df_dtc = RandomForestClassifier(n_estimators=estimators,
                                min_samples_leaf=samples_leaf,
                                random_state=RS)
df_dtc = df_dtc.fit(X_train, y_train)
Y_pred = df_dtc.predict(X_test)
plt_decision_boundaries(df_dtc,xx,yy)

plt.scatter(X.iloc[:,0],X.iloc[:,1],c=y, alpha=0.4);
plt.title('Accuracy = {0}'.format(np.round(accuracy_score(y_test, Y_pred),3)))
plt.xlabel(feature1)
plt.ylabel(feature2)
#plot legend here
plt.show()
```



As shown in the decision boundary, it is indeed evident that the random forests classifier significantly reduced overfitting. However, this was at the cost of reducing the accuracy of classification. This may be because the random forests algorithm was unable to sufficiently tackle the issue with high bias in the data set. A concrete example of this bias was that an overwhelming majority of interns achieved a single outcome score over the rest.

1.1.4 SVM classifier

One of the models used in the project was Support vector machines (SVMs) which are a type of supervised learning algorithm that can be used for classification and regression tasks. They work by finding a hyperplane that separates different classes of data, and they use kernel functions to transform the data into a higher-dimensional space where it can be more easily separated. This model was used due to its effectiveness in dealing with multidimensional data sets and its ability to work with non-linearly separable data.

[ ]:
```
X=teams.drop(['OutcomeScore'],axis=1)
y=teams['OutcomeScore']
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.2,␣
  ↪random_state=RS)
grid_space={'kernel':['rbf','sigmoid','linear'],
```

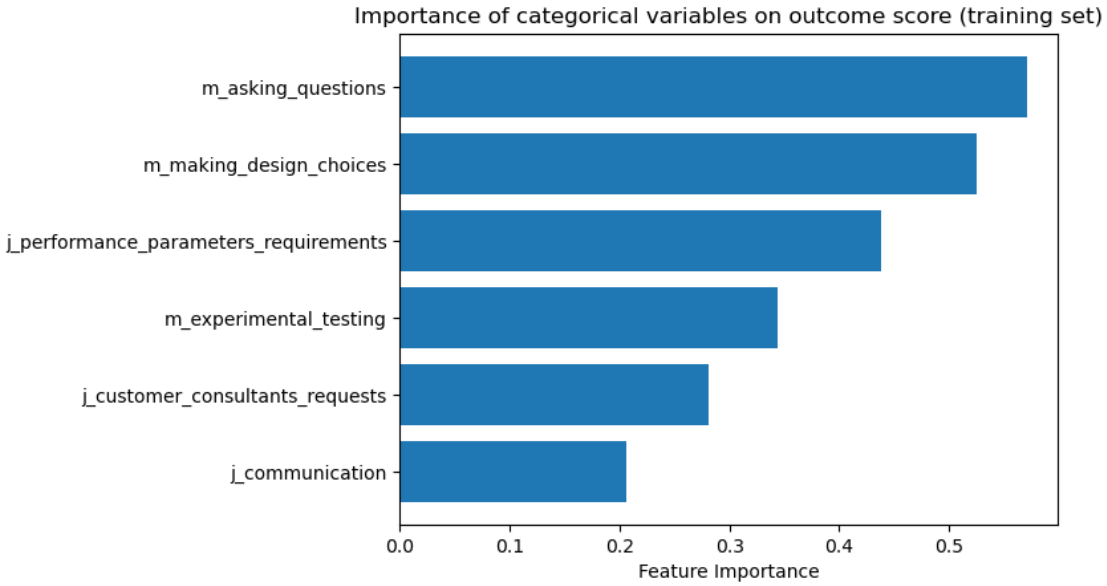
```
            'C':[0.1,1,10,100],
            'gamma': [0.01,0.1,1]
           }#cross validation process
svc=SVC(random_state=RS)
grid = GridSearchCV(estimator=svc,param_grid=grid_space,cv=3,scoring='accuracy')
model_grid = grid.fit(X_train,y_train)
print('Best hyperparameters are: '+str(model_grid.best_params_))
```

```
Best hyperparameters are: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
```

[ ]:
```
C=10
gamma=1
svc = SVC(kernel='rbf',C=C,gamma=1)
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
```

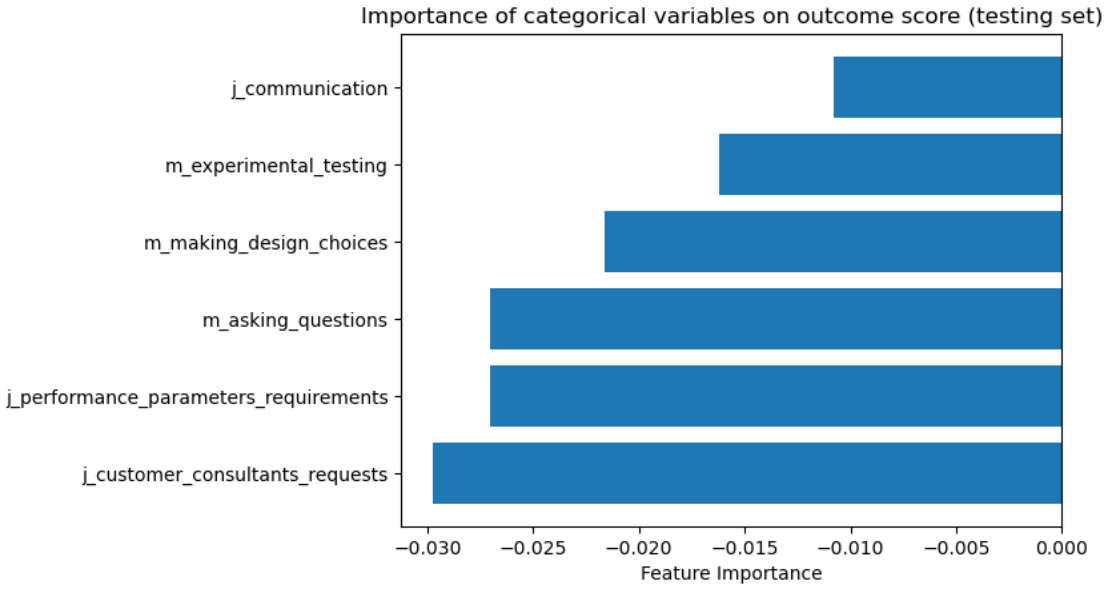
[ ]:
```
%matplotlib inline
perm_importance = permutation_importance(svc, X_train, y_train,random_state=RS)

feature_names = list(X.columns)
features = np.array(feature_names)

sorted_idx = perm_importance.importances_mean.argsort()
plt.barh(features[sorted_idx], perm_importance.importances_mean[sorted_idx])
plt.xlabel("Feature Importance")
plt.title("Importance of categorical variables on outcome score (training set)")
```

[ ]:
```
Text(0.5, 1.0, 'Importance of categorical variables on outcome score (training
set)')
```



[ ]:
```
%matplotlib inline
perm_importance = permutation_importance(svc, X_test, y_test,random_state=RS)

feature_names = list(X.columns)
features = np.array(feature_names)

sorted_idx = perm_importance.importances_mean.argsort()
plt.barh(features[sorted_idx], perm_importance.importances_mean[sorted_idx])
plt.xlabel("Feature Importance")
plt.title("Importance of categorical variables on outcome score (testing set)")
```

[ ]:
```
Text(0.5, 1.0, 'Importance of categorical variables on outcome score (testing
set)')
```

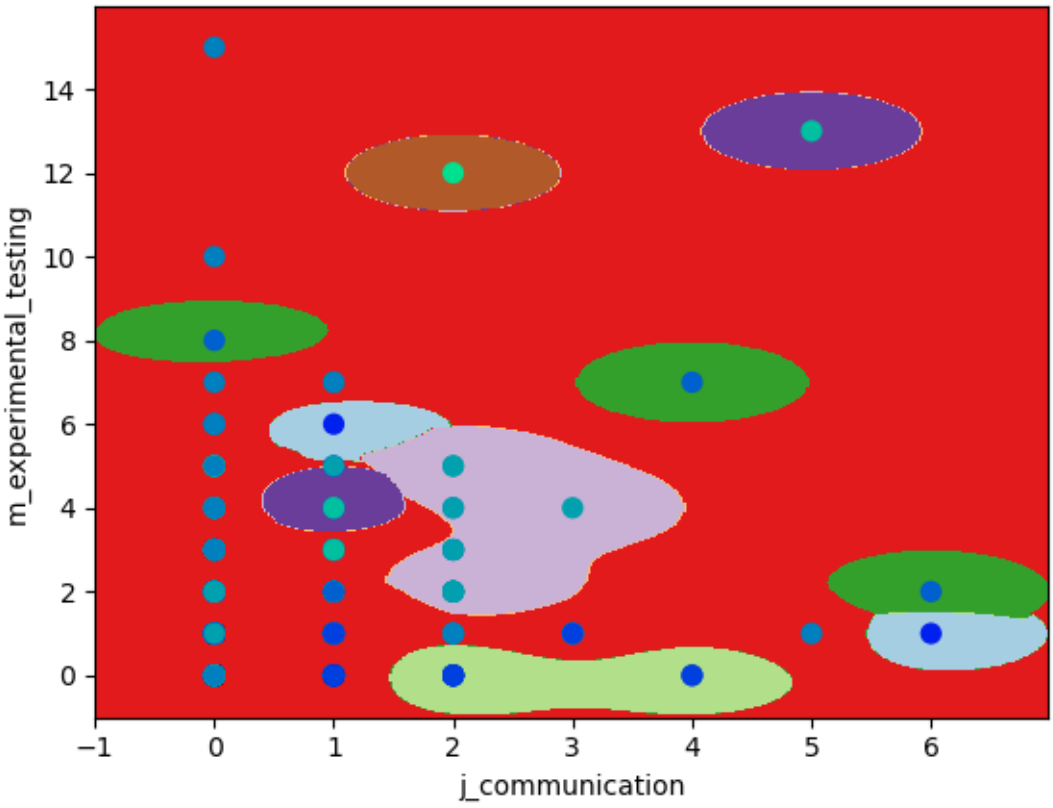


[ ]:
```
def plt_decision_boundaries(skm,X):
    """
    Takes a sklearn model (skm) with two features specified by the (N,2) array␣
↪X and plots the decision boundaries.
    """
    h = .02  # step size in the mesh
    x_min, x_max = X[:,0].min() - 1, X[:,0].max() + 1 # find the minimum and␣
↪maximum of the first feature
    y_min, y_max = X[:,1].min() - 1, X[:,1].max() + 1 # find the minimum and␣
↪maximum of the second feature
```

```
    # create a rectangular grid which goes from the minimum to maximum values␣
↪in step-size of h
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))
    # ravel is a numpy method which converts a two-dimensional array of size␣
↪(n,m) to a vector of length nm
    # column_stack is a numpy function which takes two column arrays of length␣
↪N
    # and creates a two-dimensional array of size (N,2)
    # now pass the (N,2) array to the model and predict values based on these␣
↪features, zz will have size (N,1)
    zz = skm.predict(np.column_stack([xx.ravel(), yy.ravel()]))
    zz = zz.reshape(xx.shape) # reshape zz so it has the size of the original␣
↪array xx, i.e., (n,m)
    plt.contourf(xx, yy, zz, cmap=plt.cm.Paired) # plot the decision boundaries␣
↪as filled contours
```

[ ]:
```
feature_1='j_communication'
feature_2='m_experimental_testing'
X1=teams.loc[:,[feature_1,feature_2]]
y1=teams['OutcomeScore']
X_train, X_test, y_train, y_test =train_test_split(X1,y1,test_size=0.
↪2,random_state=RS)
X1=X1.to_numpy()
```

[ ]:
```
C=10
gamma=1
svc = SVC(kernel='rbf',C=C,gamma=gamma)
svc.fit(X_train, y_train)
Y_pred = svc.predict(X_test)
plt_decision_boundaries(svc, X1) # plot the decision boundaries
plt.scatter(X1[:,0], X1[:,1], c=y, s=50, cmap='winter'); # scatter plot of X␣
↪labelled by y
plt.xlabel(feature_1)
plt.ylabel(feature_2)
plt.show()
```



1.2 Textual Analysis

Data Wrangling & Processing

[ ]:
```
# Read CSV file
chatData = pd.read_csv('VI_data.csv', encoding='latin1')

# We have no interest in predicting Mentor OutcomeScore as these are always 4.
# If kept in, will create a bias, therefore DROP.

chatData = chatData[chatData['RoleName'] == 'Player']
chatData
# Drop every column except the chat content, the outcome score and word count
messages = chatData[['userIDs','content', 'OutcomeScore', 'wordCount']]
# reshape dataset
messages = messages.reset_index(drop=True)


# # Create a column for the number of messages each student sent:
message_counts = messages.groupby("userIDs")["content"].count().reset_index().
  ↪rename(columns={'content':'messages_sent'})
```

```
messages = pd.merge(messages, message_counts, on='userIDs')
messages.head(5)
```

[ ]:

```
   userIDs                                            content  OutcomeScore  \
0        2                                Hello I am Brandon!             4
1        2  Is it imperative to know the answers to the qu…             4
2        2             Where do we go to start the discussion             4
3        2            Where do we go to start the discussion?             4
4        2  I somehow managed to miss the assignment which…             4

   wordCount  messages_sent
0          4             65
1         13             65
2          8             65
3          8             65
4         30             65
```

[ ]:

```
# Function to preprocess each chat message in the dataframe

stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.porter.PorterStemmer()

def preprocess_message(chat):
    tokeniser = RegexpTokenizer(r'\w+')
    tokens = tokeniser.tokenize(chat)

    # Lowercase and lemmatise
    lemmatiser = WordNetLemmatizer()
    lemmas = [lemmatiser.lemmatize(token.lower(), pos='v') for token in tokens]

    # Remove stopwords (words deemed insignificant)
    keywords= [lemma for lemma in lemmas if lemma not in stopwords]
    return " ".join(keywords)
```

[ ]:

```
# Create a new column for the processed chat messages
messages['clean_text'] = messages['content'].apply(preprocess_message)
messages.head()
```

[ ]:

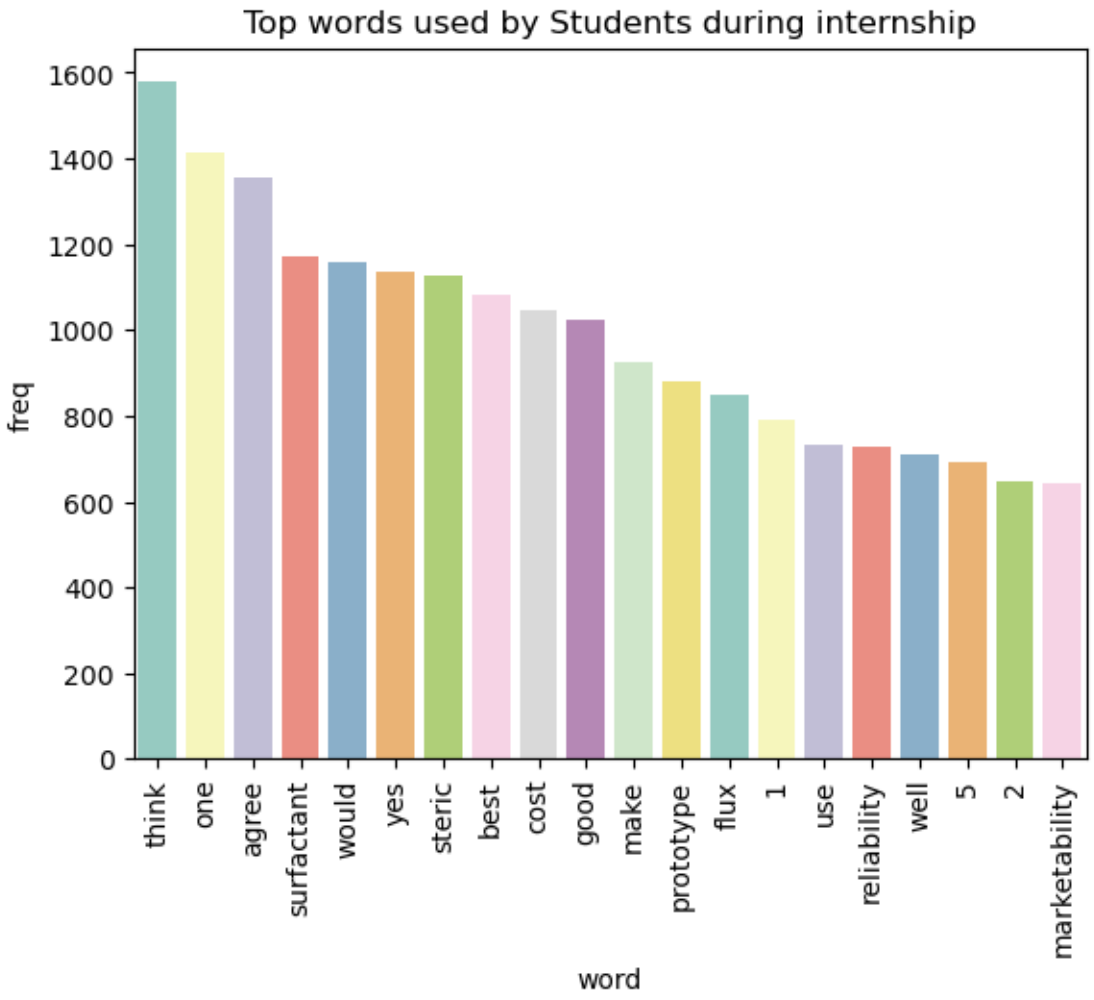
```
   userIDs                                            content  OutcomeScore  \
0        2                                Hello I am Brandon!             4
1        2  Is it imperative to know the answers to the qu…             4
2        2             Where do we go to start the discussion             4
3        2            Where do we go to start the discussion?             4
4        2  I somehow managed to miss the assignment which…             4

   wordCount  messages_sent                                      clean_text
0          4             65                                   hello brandon
```

```
1         13             65          imperative know answer question interview
2          8             65                                go start discussion
3          8             65                                go start discussion
4         30             65  somehow manage miss assignment obviously base …
```

Data Analysis

[ ]:
```
## Find top words by frequency in dataset

num_messages = messages.shape[0]

words = []

for index, row in messages.iterrows():
    # split the string in the 'text' column by whitespace and append the␣
 ↪resulting list to the word_list
    words.extend(row['clean_text'].split())

no_words_set = len(words)

all_words = pd.Series(words)
word_freq = pd.DataFrame(all_words.value_counts())

word_freq = word_freq.rename(columns={0:"term appears"})
word_freq['tf'] = word_freq['term appears']/word_freq.shape[0]

top_20 = word_freq.head(20)

sns.barplot(x=top_20.index, y=top_20['term appears'], palette='Set3')
plt.xticks(rotation = 90)
plt.xlabel('word')
plt.ylabel('freq')
plt.title('Top words used by Students during internship')
```

[ ]:
```
Text(0.5, 1.0, 'Top words used by Students during internship')
```



[ ]:
```
word_freq = word_freq.reset_index()
word_freq = word_freq.rename(columns={'index':'word'})
word_freq.head()
```

[ ]:

	word	term appears	tf
0	think	1578	0.370858
1	one	1415	0.332550
2	agree	1357	0.318919
3	surfactant	1170	0.274971
4	would	1158	0.272150

Part-Of-Speech Tagging

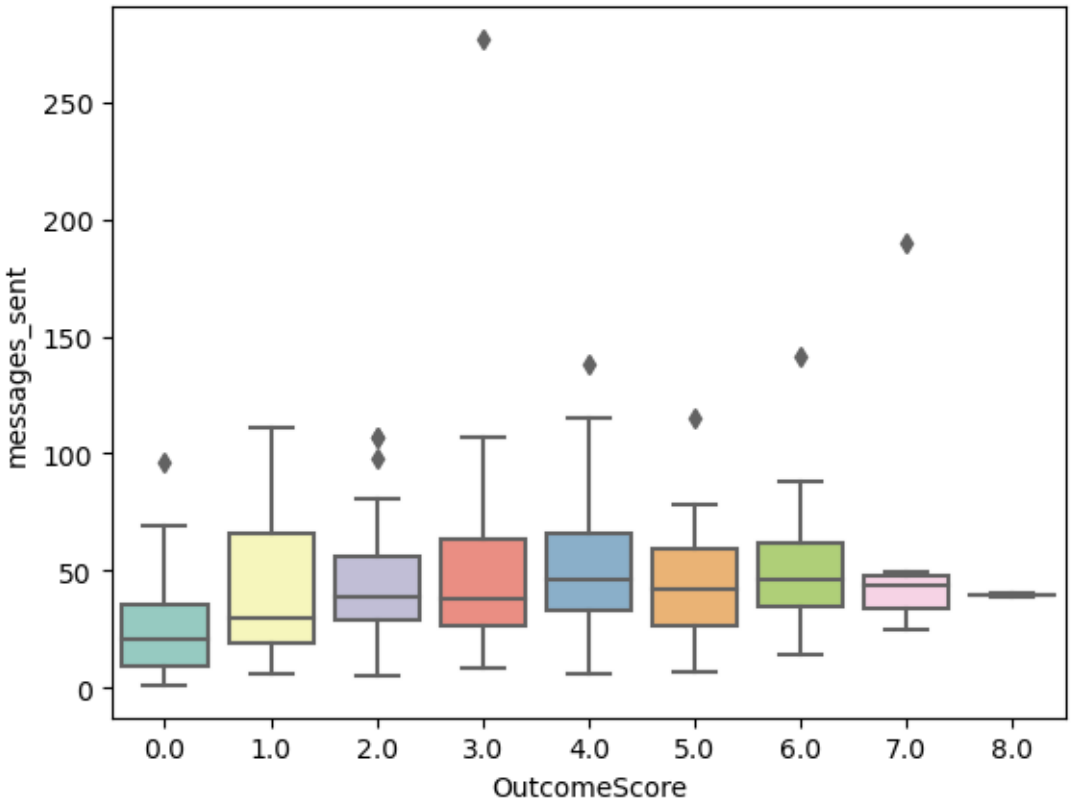
[ ]:
```
word_freq['pos_tags']  = word_freq['word'].apply(lambda x: nltk.pos_tag(nltk.
↪word_tokenize(x))[0][1])
```

```
nouns = word_freq[word_freq['pos_tags'] == 'NN']
nouns.head(10)
# nouns = [word_freq.index[i] for i in range(len(pos_tags)) if pos_tags[i][1]␣
 ↪== 'NN']

word_freq.head(20)

top10_NN = nouns.head(10)['word']

# We'll be using these top 10 most featured nouns to see if they are able to␣
 ↪increase
# the accuracy of our model.

top10_NN
```

[ ]:
```
0             think
2             agree
3        surfactant
6            steric
8              cost
11        prototype
12             flux
14              use
15      reliability
19    marketability
Name: word, dtype: object
```

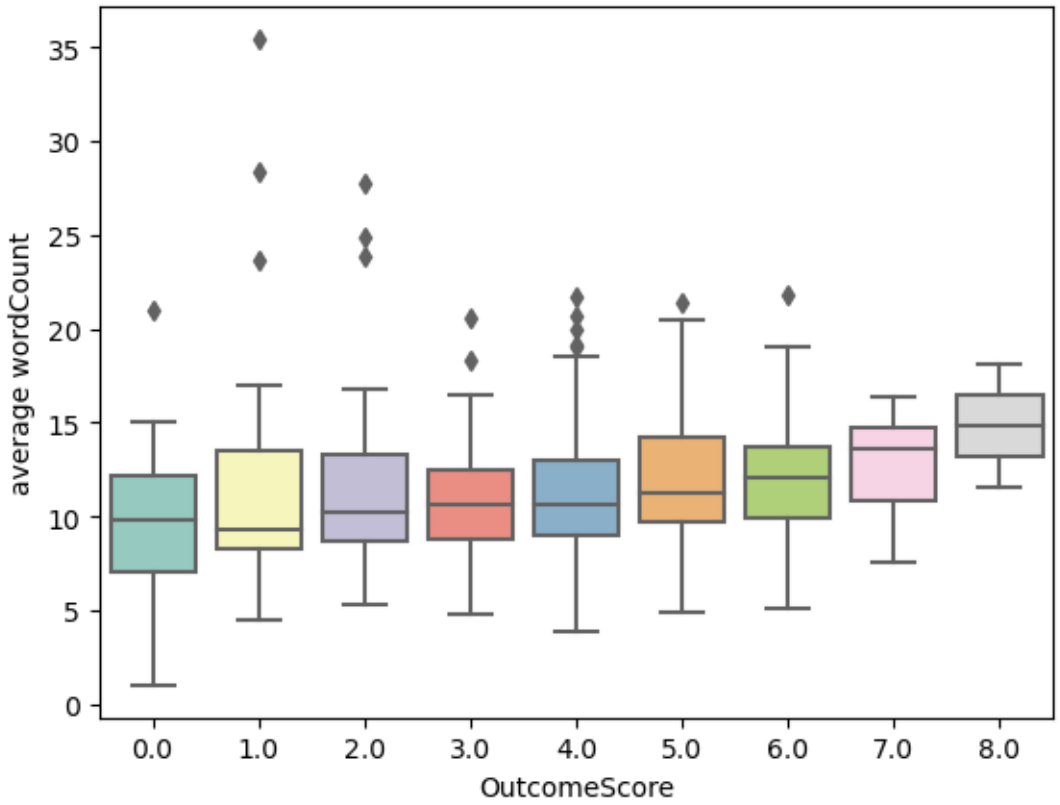
[ ]:
```
# Let's see if we can find a correlation  between the number of messages sent␣
 ↪and the outcome score

messagesSent = messages.groupby(['userIDs'])['messages_sent', 'OutcomeScore'].
 ↪mean().reset_index()
messagesSent

sns.boxplot(x='OutcomeScore', y='messages_sent', data=messagesSent,␣
 ↪palette='Set3')

corr_coef = messagesSent['OutcomeScore'].corr(messagesSent['messages_sent'])
print(corr_coef)
```

```
0.13763014641711357
```



[ ]:
```
# What about word count?

word_count = messages.groupby(['userIDs'])['wordCount', 'OutcomeScore'].mean().
  ↪reset_index()

sns.boxplot(x='OutcomeScore', y='wordCount', data=word_count, palette='Set3')
plt.ylabel('average wordCount')

corr_coef_WC = word_count['OutcomeScore'].corr(word_count['wordCount'])
print(corr_coef_WC)
```

```
0.09591840881501122
```

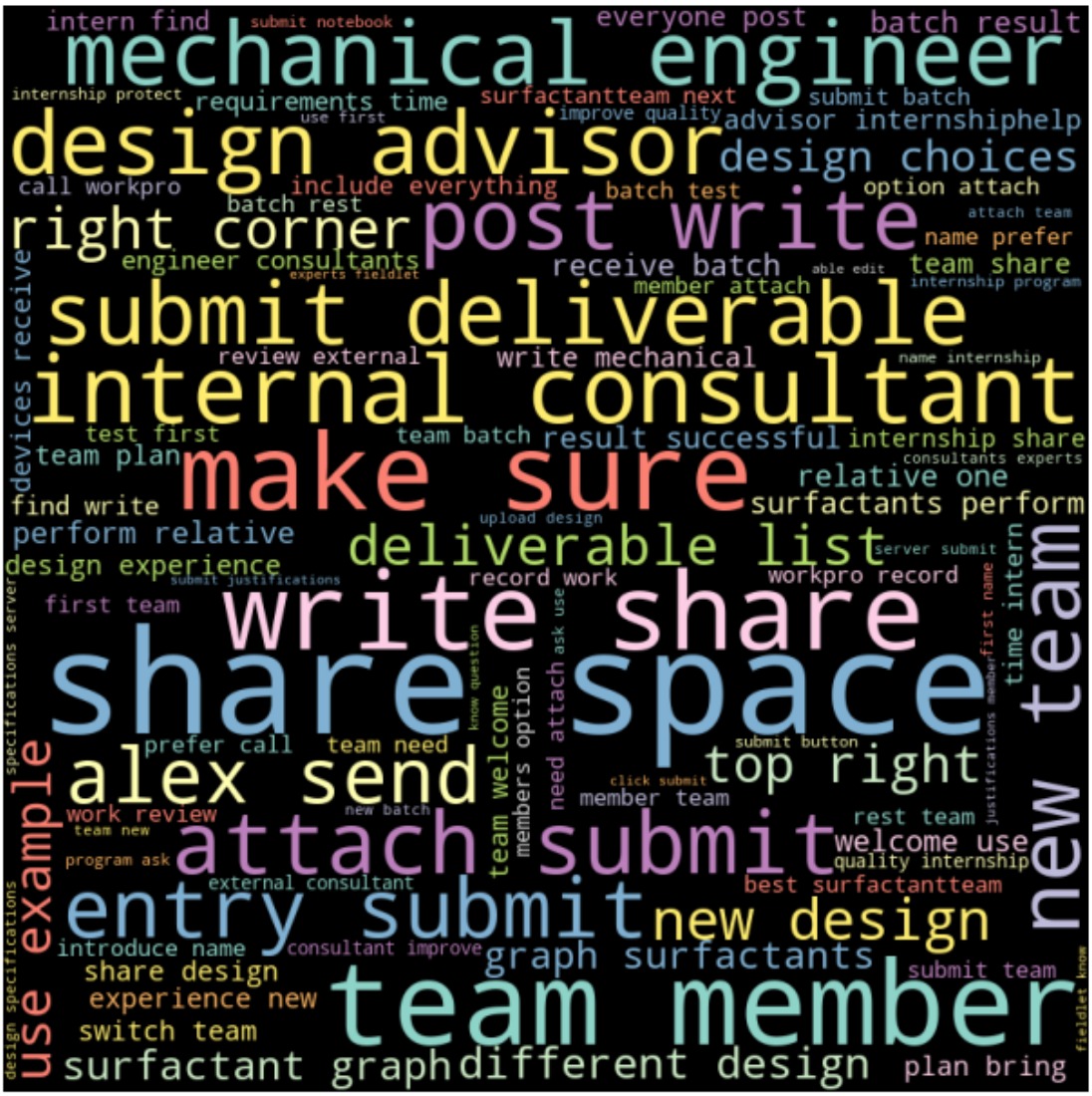


[ ]:
```
messages = pd.read_csv('VI_data.csv', encoding='latin1')
students = messages[messages['RoleName'] == 'Player']
mentors =  messages[messages['RoleName'] == 'Mentor']

def preprocess_message(chat):
    tokeniser = RegexpTokenizer(r'\w+')
    tokens = tokeniser.tokenize(chat)

    # Lowercase and lemmatise
    lemmatiser = WordNetLemmatizer()
    lemmas = [lemmatiser.lemmatize(token.lower(), pos='v') for token in tokens]

    # Remove stopwords (words deemed insignificant)
    keywords= [lemma for lemma in lemmas if lemma not in stopwords]
    return " ".join(keywords)

students['clean_text'] = students['content'].apply(preprocess_message)
mentors['clean_text'] = mentors['content'].apply(preprocess_message)
messages['clean_text'] = messages['content'].apply(preprocess_message)
```

[ ]:
```
student_words = ' '
mentor_words = ' '
all_words = ' '
stopwords = set(STOPWORDS)

for text in students['clean_text']:
    text = str(text).lower()

    tokens=text.split()
    student_words = student_words + ' '.join(tokens)

for text in mentors['clean_text']:
    text = str(text).lower()

    tokens=text.split()
    mentor_words = mentor_words + ' '.join(tokens)

for text in messages['clean_text']:
    text = str(text).lower()

    tokens=text.split()
    all_words = all_words + ' '.join(tokens)
```

[ ]:
```
wordcloud = WordCloud(
    width=800,
    height=800,
    background_color="black",
    stopwords=stopwords,
    min_font_size=10,
    colormap="Set3",
    random_state = 1,
).generate(mentor_words)

# plot the WordCloud image
plt.figure(figsize=(6, 6), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.title('ALL MENTOR MESSAGES')

plt.show()
```

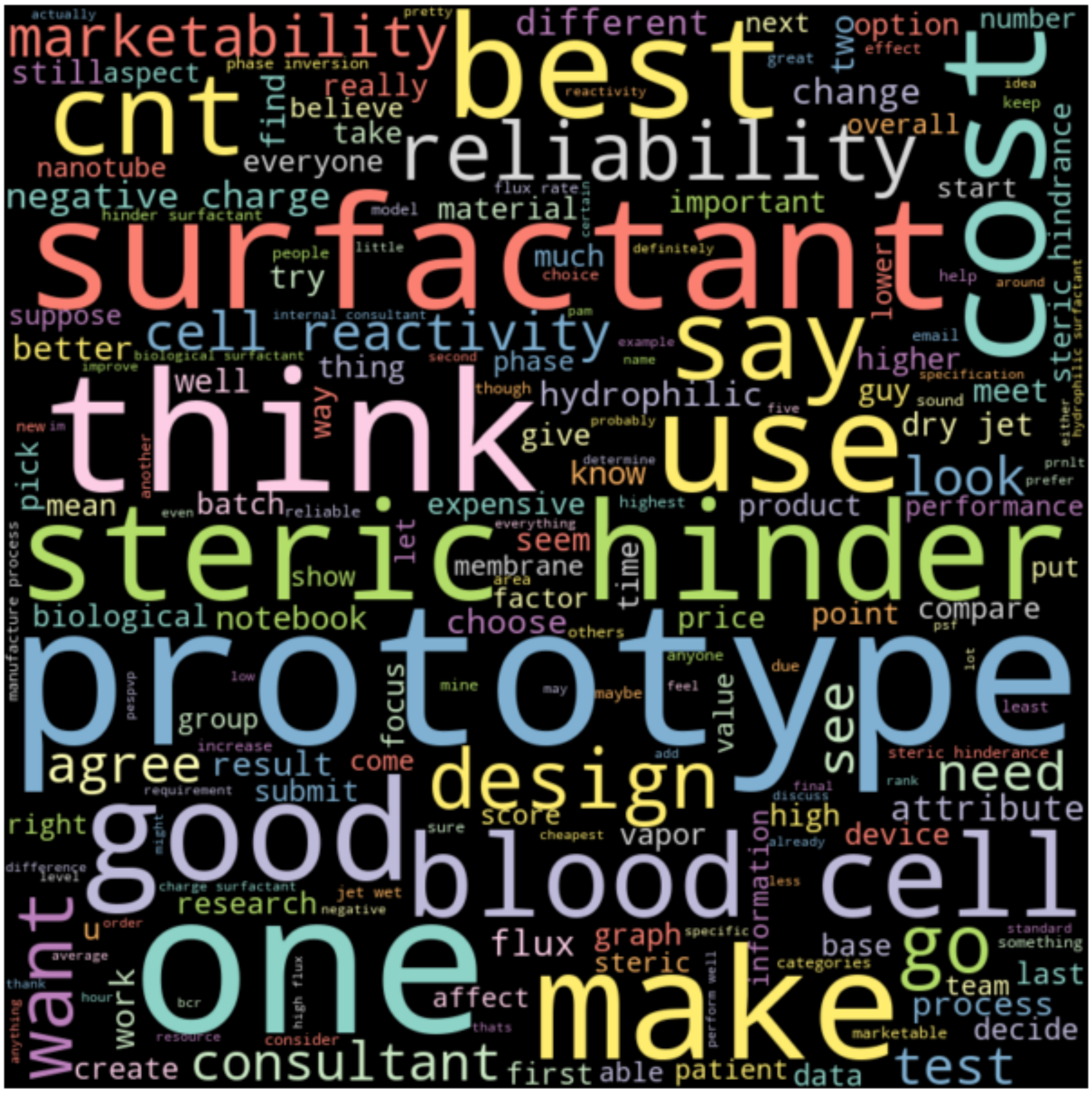
ALL MENTOR MESSAGES



[ ]:
```
wordcloud = WordCloud(
    width=800,
    height=800,
    background_color="black",
    stopwords=stopwords,
    min_font_size=10,
    colormap="Set3",
    random_state = 1
).generate(student_words)

# plot the WordCloud image
plt.figure(figsize=(6, 6), facecolor=None)
```

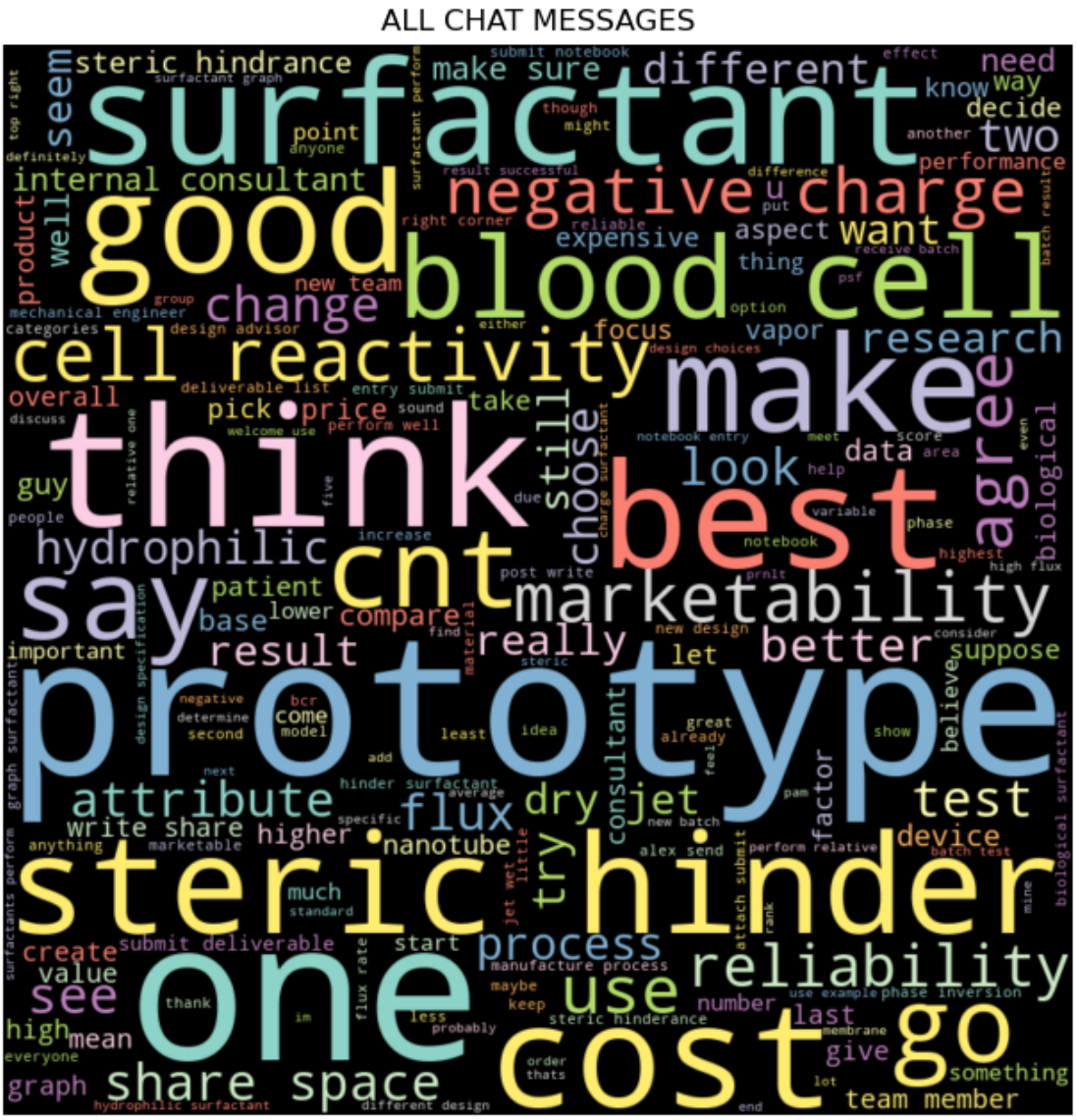
```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad=0)
plt.title('ALL STUDENT MESSAGES')


plt.show()
```

ALL STUDENT MESSAGES




```
[ ]: wordcloud = WordCloud(
         width=800,
         height=800,
```

```
    background_color="black",
    stopwords=stopwords,
    min_font_size=10,
    colormap="Set3",
    random_state = 1
).generate(all_words)


# plot the WordCloud image
plt.figure(figsize=(6, 6), facecolor=None)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad=0)
plt.title('ALL CHAT MESSAGES')

plt.show()
```



1.3 Modelling

Types of Models to attempt:

- Linear Regression
- Logistic Regression
- Random Forest Classification (we are predicting a catergorical variable (9 discrete values) therefore shouldn't use regression)
- Support Vector Machines

What features can we use to create these models?

- TF-IDF (Term Frequency-Inverse Document Frequency)

- Sentiment Analysis
- Word Count
- Presence of top nouns (by frequency)
- Number of messages sent by students

[ ]:
```
# First let's try a simple regression analysis with our 2 variables (WordCount␣
↪and messages_sent)

reg_mes=sm.ols(formula = 'OutcomeScore ~ messages_sent+wordCount', data=␣
↪messages).fit()
print(reg_mes.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           OutcomeScore   R-squared:                       0.003
Model:                            OLS   Adj. R-squared:                  0.003
Method:                 Least Squares   F-statistic:                     26.32
Date:                Sun, 28 May 2023   Prob (F-statistic):           3.88e-12
Time:                        20:02:36   Log-Likelihood:                -31433.
No. Observations:               16902   AIC:                         6.287e+04
Df Residuals:                   16899   BIC:                         6.290e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept         3.5556      0.025    144.128      0.000       3.507
3.604
messages_sent     0.0021      0.000      7.039      0.000       0.002
0.003
wordCount         0.0018      0.001      2.027      0.043    6.04e-05
0.004
==============================================================================
Omnibus:                      111.530   Durbin-Watson:                   0.040
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              113.331
Skew:                          -0.197   Prob(JB):                     2.46e-25
Kurtosis:                       2.928   Cond. No.                         156.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

From the above results we can see that we get a very low R-squared score. This indicates that using messages_sent and wordCount and linear regression was not a good method for predicting

OutcomeScore.

Let's try doing some Natural Language Processing to see what outcome we can get.

Vectorisation There are 4 different Vectorisation methods that we can use to convert our text data into numbers rather than words. This provides us with something useful for making predictions. - TF-IDF Vectorisation - HashingVectoriser - Doc2Vec - CountVectoriser

[ ]:
```
randomState = 42

X = messages['clean_text']
y = messages['OutcomeScore']

#TF-IDF Vectorizer
vectorizer_tf = TfidfVectorizer()
X_vect_tf = vectorizer_tf.fit_transform(X)

#HashingVectoriser
vectorizer_h = HashingVectorizer()
X_vect_h = vectorizer_h.fit_transform(X)


# Doc2Vec
X_cont = messages['content']
documents = [TaggedDocument(words=word_tokenize(text.lower()), tags=[str(i)])␣
 ↪for i, text in enumerate(X_cont)]

model = Doc2Vec(documents, vector_size=100, window=5, min_count=1, workers=4,␣
 ↪epochs=20)

doc_vectors = [model.infer_vector(doc.words) for doc in documents]

#CountVectorizer
vectorizer_c = CountVectorizer()
X_vect_c = vectorizer_c.fit_transform(X)
```

[ ]:
```
# Lets try a linear regression, we are predicting categorical data therefore␣
 ↪this model will not work!

X_train, X_test, y_train, y_test = train_test_split(X_vect_tf,y, test_size=0.2,␣
 ↪random_state=randomState)

lin_model = LinearRegression()

lin_model.fit(X_train, y_train)

y_pred = lin_model.predict(X_test)
# accuracy = accuracy_score(y_test, y_pred)
```

```
# print(f"accuracy: {accuracy}")
mse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', mse)

r2 = r2_score(y_test, y_pred)
print('R^2: ', r2)
```

```
RMSE: 1.7564643978749295
R^2:  -0.29293874380455875
```

As we can see for Linear Regression we get a rmse of 1.756, let's see how it compares to Logistic Regression which is a preferable model for the data we are trying to predict.

[ ]:
```
# Let's create a simple Logistic Regression model using TF-IDF

X_train, X_test, y_train, y_test = train_test_split(X_vect_tf,y, test_size=0.2,␣
 ↪random_state=randomState)

tf_model = LogisticRegression(max_iter=1000)

tf_model.fit(X_train, y_train)

y_pred = tf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"accuracy: {accuracy}")
rmse =mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', rmse)
r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
accuracy: 0.35610766045548653
RMSE: 1.6250952694299048
R^2: -0.1068
```

[ ]:
```
# Hashing Vectoriser

X_train, X_test, y_train, y_test = train_test_split(X_vect_h,y, test_size=0.2,␣
 ↪random_state=randomState)

h_model = LogisticRegression(max_iter=1000)

h_model.fit(X_train, y_train)

y_pred_h = h_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_h)
print(f"accuracy: {accuracy}")
rmse =mean_squared_error(y_test, y_pred_h, squared=False)
print('RMSE:', rmse)
```

```
r2 = r2_score(y_test, y_pred_h)
print("R^2: %.4f" % r2)
```

```
accuracy: 0.3528541851523218
RMSE: 1.6292759363902498
R^2: -0.1125
```

[ ]:
```
# doc2vec

X_train, X_test, y_train, y_test = train_test_split(doc_vectors, y, test_size=0.
 ↪2, random_state=randomState)

doc_model = LogisticRegression(max_iter=1000)

doc_model.fit(X_train, y_train)

y_pred = doc_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"accuracy: {accuracy}")
rmse =mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', rmse)
r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
accuracy: 0.35403726708074534
RMSE: 1.5912545670974487
R^2: -0.0612
```

[ ]:
```
# Count Vectoriser

X_train, X_test, y_train, y_test = train_test_split(X_vect_c,y, test_size=0.2,␣
 ↪random_state=randomState)

c_model = LogisticRegression(max_iter=1000)

c_model.fit(X_train, y_train)

y_pred_c = c_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_c)
print(f"accuracy: {accuracy}")
rmse =mean_squared_error(y_test, y_pred_c, squared=False)
print('RMSE:', rmse)
r2 = r2_score(y_test, y_pred_c)
print("R^2: %.4f" % r2)
```

```
accuracy: 0.3300798580301686
RMSE: 1.745743121887939
R^2: -0.2772
```

We get an improved rmse score for Logistic Regression when compared to Linear Regression. Indicates that it is a better model to use. However we get a negative R^2 score, which indicates that the model is not fitting the data well at all. Thereforee we should try another model to see if we get a slightly better result. Our doc2vec model also produces the best results.

[ ]:
```
# We should also try using a random forest regression approach to test accuracy.

X_train, X_test, y_train, y_test = train_test_split(X_vect_tf, y, test_size=0.2, random_state=42)

# Train a logistic regression model on the training data
rf_model = RandomForestClassifier(n_estimators=150, max_depth=10)
rf_model.fit(X_train, y_train)

# Evaluate the model on the test data
y_predRF = rf_model.predict(X_test)

accuracy = accuracy_score(y_test, y_predRF)
print(f"accuracy: {accuracy}")

rmse = mean_squared_error(y_test, y_predRF, squared=False)
print("RMSE: %.4f" % rmse)

r2 = r2_score(y_test, y_predRF)
print("R^2: %.4f" % r2)
```

```
accuracy: 0.3581780538302277
RMSE: 1.5753
R^2: -0.0400
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(doc_vectors, y, test_size=0.2, random_state=42)

# Train a logistic regression model on the training data
rf_model = RandomForestClassifier(n_estimators=50, max_depth=10)
rf_model.fit(X_train, y_train)

# Evaluate the model on the test data
y_predRF = rf_model.predict(X_test)

accuracy = accuracy_score(y_test, y_predRF)
print(f"accuracy: {accuracy}")

rmse = mean_squared_error(y_test, y_predRF, squared=False)
print("RMSE: %.4f" % rmse)

r2 = r2_score(y_test, y_predRF)
```

```
print("R^2: %.4f" % r2)
```

```
accuracy: 0.35729074238391006
RMSE: 1.5774
R^2: -0.0428
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(doc_vectors, y, test_size=0.
↪2, random_state=42)

# Train a logistic regression model on the training data
rf_model = DecisionTreeClassifier(random_state=42, max_depth=10)
rf_model.fit(X_train, y_train)

# Evaluate the model on the test data
y_predRF = rf_model.predict(X_test)

accuracy = accuracy_score(y_test, y_predRF)
print(f"accuracy: {accuracy}")

rmse = mean_squared_error(y_test, y_predRF, squared=False)
print("RMSE: %.4f" % rmse)

r2 = r2_score(y_test, y_predRF)
print("R^2: %.4f" % r2)
```

```
accuracy: 0.34131913635019223
RMSE: 1.6456
R^2: -0.1349
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_vect_h,y, test_size=0.2,␣
↪random_state=randomState)

rf_model = RandomForestClassifier(n_estimators=50, max_depth=10,␣
↪random_state=42)
rf_model.fit(X_train, y_train)

y_pred_h = h_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_h)
print(f"accuracy: {accuracy}")
rmse =mean_squared_error(y_test, y_pred_h, squared=False)
print('RMSE:', rmse)
r2 = r2_score(y_test, y_pred_h)
print("R^2: %.4f" % r2)
```

```
accuracy: 0.3528541851523218
RMSE: 1.6292759363902498
R^2: -0.1125
```

Using a random forest regression we get an improved RMSE on the one we achieved in the logistic

regression. While we don't get a positive R^2 score, it is much closer to zero than in our logistic regression.

Using the different vectorisation methods, we can see that the doc2vec model has the best accuracy while the tf-idf vectoriser has the best rmse.

We should use one of these vectorisors in any further calculations.

Sentiment Analysis Sentiment analysis is a technique in NLP that involves analysing text in order to determine the sentiment or emotional tone of a text. In this project we use sentiment analysis to classify student's chat messages as positive, negative or neutral, we then use this to determine whether it has an effect on OutcomeScore.

Sentiment analysis assigns a numerical score to the chat messages

We will use packages spaCy and textblob to find these sentiment scores. .blob.polarity gives us our sentiment/polarity score. It is a score between [-1, 1] where < 0 indicates negative sentiment, > 0 indicates positive sentiment while anything close to 0 indicates neutral sentiment.

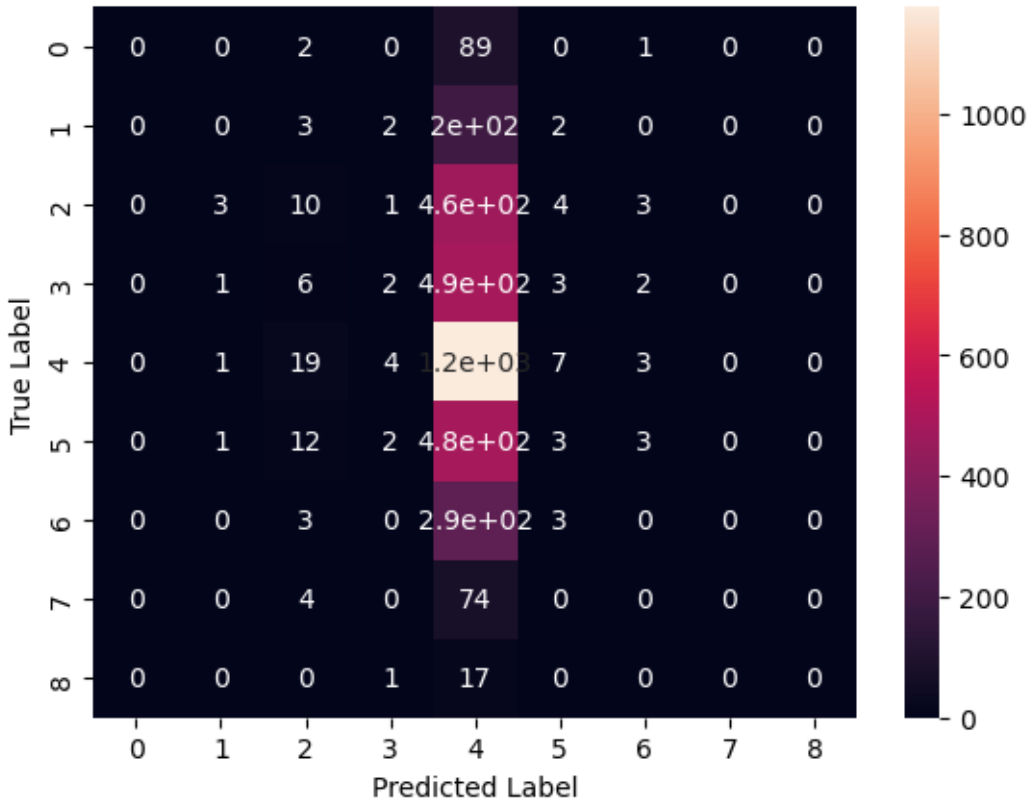
[ ]:
```
nlp = spacy.load('en_core_web_sm')
nlp.add_pipe('spacytextblob')

# Create function to determine sentiment score of chat message

def get_sentiment_scores(text):
    doc = nlp(text)
    sentiment = doc._.blob.polarity
    return round(sentiment,2)

messages['sentimentScore'] = messages['clean_text'].apply(get_sentiment_scores)
messages.head()
```

[ ]:
```
   userIDs                                          content  OutcomeScore  \
0        2                                Hello I am Brandon!             4
1        2  Is it imperative to know the answers to the qu…             4
2        2            Where do we go to start the discussion             4
3        2           Where do we go to start the discussion?             4
4        2  I somehow managed to miss the assignment which…             4

   wordCount  messages_sent  \
0          4             65
1         13             65
2          8             65
3          8             65
4         30             65

                                      clean_text  sentimentScore
0                                  hello brandon            0.00
1          imperative know answer question interview            0.00
```

```
2                                  go start discussion            0.00
3                                  go start discussion            0.00
4  somehow manage miss assignment obviously base …           -0.43
```

[ ]:
```
X_sentiment = messages[['sentimentScore']]

X_train, X_test, y_train, y_test = train_test_split(X_sentiment, y, test_size=0.
 ↪2, random_state=42)

clf = RandomForestClassifier(n_estimators=150, max_samples=.5).fit(X_train,␣
 ↪y_train)

# Make predictions on the test set
y_pred_rf = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred_rf)
print('Accuracy:', acc)


# Evaluate the model using mean squared error
rmse = mean_squared_error(y_test, y_pred_rf, squared=False)
print('RMSE:', rmse)

r2 = r2_score(y_test, y_pred_rf)
print("R^2: %.4f" % r2)


# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot=True)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

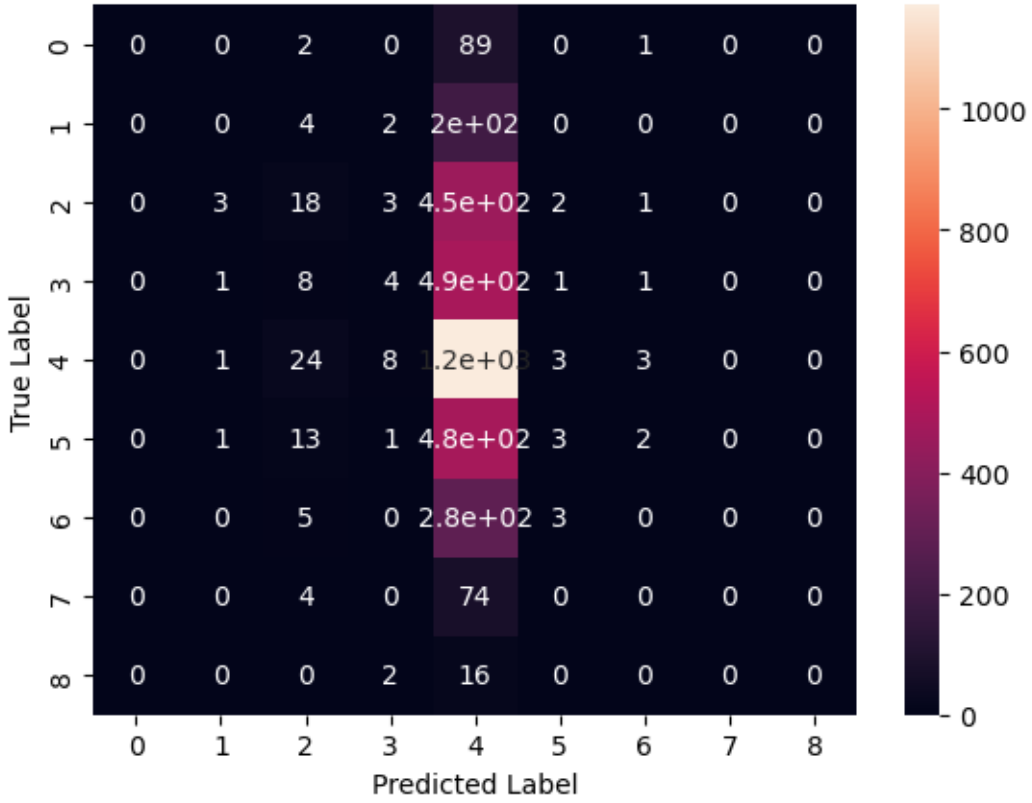
```
Accuracy: 0.3525584146702159
RMSE: 1.605779269700264
R^2: -0.0806
```



[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_sentiment, y, test_size=0.
↪2, random_state=42)

clf = DecisionTreeClassifier(random_state=42, max_depth=10).fit(X_train,␣
↪y_train)

# Make predictions on the test set
y_pred_rf = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred_rf)
print('Accuracy:', acc)


# Evaluate the model using mean squared error
rmse = mean_squared_error(y_test, y_pred_rf, squared=False)
print('RMSE:', rmse)

r2 = r2_score(y_test, y_pred_rf)
print("R^2: %.4f" % r2)
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot=True)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Accuracy: 0.35403726708074534
RMSE: 1.600983134804863
R^2: -0.0742
```



[ ]:
```
# Combine vectorisation with sentiment score
X = pd.concat([pd.DataFrame(X_vect_tf.toarray()), messages['sentimentScore']],␣
 ↪axis=1)
X.columns = X.columns.astype(str)


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Train a linear regression model
clf = RandomForestClassifier(n_estimators=50, max_samples=.5).fit(X_train,␣
 ↪y_train)

# Evaluate the model on the testing set
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print('Accuracy:', acc)

rmse = mean_squared_error(y_test, y_pred, squared=False)
print("RMSE: %.4f" % rmse)

r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy: 0.3427979887607217
RMSE: 1.6843
R^2: -0.1889
```

From the above models we can see that using vectorisation is not a good metric to use to predict OutcomeScore. When paired with sentimentScore, tf-idf vectorisation causes the accuracy to go down and the RMSE to go up which is not what we want.

Keyword Frequency In the analysis section of this notebook, we extracted the 10 most used nouns by students during their internship. We want to see if using these words leads to a higher outcomeScore.

[ ]:
```
keywords = np.array(top10_NN)

# Let's add the presence of these nouns to our feature matrix
for keyword in keywords:
    messages[f'{keyword}_frequency'] = messages['content'].apply(lambda x: x.
 ↪count(keyword))

messages.head(5)
```

[ ]:

	userIDs	content	OutcomeScore	\
0	2	Hello I am Brandon!	4	
1	2	Is it imperative to know the answers to the qu…	4	
2	2	Where do we go to start the discussion	4	
3	2	Where do we go to start the discussion?	4	
4	2	I somehow managed to miss the assignment which…	4	

	wordCount	messages_sent	\
0	4	65	
1	13	65	
2	8	65	
3	8	65	

```
4         30             65

                                        clean_text  sentimentScore  \
0                                     hello brandon            0.00
1          imperative know answer question interview            0.00
2                               go start discussion            0.00
3                               go start discussion            0.00
4  somehow manage miss assignment obviously base …           -0.43

   think_frequency  agree_frequency  surfactant_frequency  steric_frequency  \
0                0                0                     0                 0
1                0                0                     0                 0
2                0                0                     0                 0
3                0                0                     0                 0
4                0                0                     0                 0

   cost_frequency  prototype_frequency  flux_frequency  use_frequency  \
0               0                    0               0              0
1               0                    0               0              0
2               0                    0               0              0
3               0                    0               0              0
4               0                    0               0              0

   reliability_frequency  marketability_frequency
0                      0                       0
1                      0                       0
2                      0                       0
3                      0                       0
4                      0                       0
```

[ ]:
```
# Let's create the Keyword Frequency Matrix that we can use to create our model :)

X_keywords = messages.iloc[:, 7:17]
X_keywords
y = messages['OutcomeScore']

X_train, X_test, y_train, y_test = train_test_split(X_keywords, y, test_size=0.2, random_state=42)

rf_model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42, )
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred_rf)
print('Accuracy:', acc)

rmse = mean_squared_error(y_test, y_pred_rf, squared=False)
print("RMSE: %.4f" % rmse)

r2 = r2_score(y_test, y_pred_rf)
print("R^2: %.4f" % r2)
```

```
Accuracy: 0.3427979887607217
RMSE: 1.5877
R^2: -0.0564
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_keywords, y, test_size=0.
↪2, random_state=42)


clf = DecisionTreeClassifier(max_depth=10, random_state=42).fit(X_train,␣
↪y_train)

# Make predictions on the test set
y_pred_rf = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred_rf)
print('Accuracy:', acc)

rmse = mean_squared_error(y_test, y_pred_rf, squared=False)
print("RMSE: %.4f" % rmse)

r2 = r2_score(y_test, y_pred_rf)
print("R^2: %.4f" % r2)
```

```
Accuracy: 0.3427979887607217
RMSE: 1.6036
R^2: -0.0776
```

[ ]:
```
row = {'no_estimators': 150, 'accuracy': accuracy}
for name, score in zip(X_keywords.columns, rf_model.feature_importances_):
    print(name,np.round(score,3))
```

```
think_frequency 0.102
agree_frequency 0.062
surfactant_frequency 0.148
steric_frequency 0.097
cost_frequency 0.12
prototype_frequency 0.11
flux_frequency 0.104
use_frequency 0.112
reliability_frequency 0.078
```

```
marketability_frequency 0.069
```

From the above analysis we can see that use of the words 'surfactant' and 'use' had the biggest impact, followed by 'cost'. 'agree' had the smallest impact.

Topic Modelling Let's see if we can extract some topics from the chat messages? Will the presence of certain topics lead to a higher accuracy?

[ ]:
```
messages['tokens'] = messages['clean_text'].apply(word_tokenize)

# Create a document-term matrix
dictionary = corpora.Dictionary(messages['tokens'])
doc_term_matrix = [dictionary.doc2bow(doc) for doc in messages['tokens']]

# Train the LDA model
num_topics = 10
lda_model = models.LdaModel(doc_term_matrix, num_topics=num_topics,␣
 ↪id2word=dictionary)

# Interpret the topics
for idx, topic in lda_model.print_topics(-1):
    print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: 0.025*"one" + 0.024*"okay" + 0.023*"see" + 0.023*"would" + 0.022*"say" +
0.021*"well" + 0.021*"meet" + 0.020*"consultants" + 0.018*"tyler" + 0.014*"go"
Topic: 1
Words: 0.046*"batch" + 0.033*"ok" + 0.031*"one" + 0.030*"like" + 0.029*"make" +
0.029*"create" + 0.024*"pick" + 0.018*"everyone" + 0.015*"thats" +
0.014*"prototypes"
Topic: 2
Words: 0.171*"yes" + 0.023*"device" + 0.023*"would" + 0.023*"agree" +
0.020*"correct" + 0.019*"research" + 0.017*"make" + 0.016*"group" +
0.015*"specifications" + 0.014*"many"
Topic: 3
Words: 0.035*"flux" + 0.028*"definitely" + 0.025*"guy" + 0.021*"reliability" +
0.020*"add" + 0.020*"cnt" + 0.019*"take" + 0.018*"much" + 0.018*"cost" +
0.015*"great"
Topic: 4
Words: 0.046*"vapor" + 0.038*"steric" + 0.034*"dry" + 0.032*"phase" + 0.032*"1"
+ 0.031*"jet" + 0.025*"20" + 0.025*"cnt" + 0.023*"4" + 0.021*"5"
Topic: 5
Words: 0.048*"notebook" + 0.031*"email" + 0.027*"need" + 0.027*"team" +
0.023*"submit" + 0.021*"share" + 0.020*"anyone" + 0.017*"one" + 0.017*"next" +
0.016*"else"
Topic: 6
Words: 0.054*"agree" + 0.047*"surfactant" + 0.036*"best" + 0.034*"steric" +
0.033*"cost" + 0.029*"hinder" + 0.022*"also" + 0.018*"reliability" +
```

```
0.018*"well" + 0.018*"think"
Topic: 7
Words: 0.102*"good" + 0.078*"thank" + 0.051*"yeah" + 0.038*"sound" +
0.025*"blood" + 0.023*"cell" + 0.023*"okay" + 0.022*"reactivity" + 0.019*"work"
+ 0.016*"idea"
Topic: 8
Words: 0.062*"prototype" + 0.032*"think" + 0.028*"get" + 0.027*"5" +
0.025*"prototypes" + 0.020*"one" + 0.020*"right" + 0.020*"4" + 0.017*"sure" +
0.017*"us"
Topic: 9
Words: 0.035*"process" + 0.032*"change" + 0.026*"use" + 0.024*"different" +
0.023*"1" + 0.020*"prototype" + 0.019*"result" + 0.018*"flux" +
0.017*"attribute" + 0.017*"one"
```

[ ]:
```
topic_probs = pd.DataFrame(columns=['topic_{}'.format(i) for i in␣
 ↪range(lda_model.num_topics)])

# Loop over each message and obtain its topic distribution probabilities
for index, row in messages.iterrows():
    message = row['clean_text']
    bow_vector = lda_model.id2word.doc2bow(message.split())
    topic_distribution = lda_model.get_document_topics(bow_vector)

    # Fill in the corresponding topic probabilities for the current message
    probs_dict = dict(topic_distribution)
    topic_probs.loc[index] = [probs_dict.get(i, 0.0) for i in range(lda_model.
 ↪num_topics)]

# Concatenate the topic probabilities DataFrame with the original DataFrame
df_with_topics = pd.concat([messages, topic_probs], axis=1)
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(topic_probs, y, train_size=.
 ↪8, random_state=42)
# instantatiate the RFC with 100 ensemble members
clf = RandomForestClassifier(n_estimators=50, max_samples=.5, max_depth=10,␣
 ↪random_state=42).fit(X_train, y_train)

y_pred = clf.predict(X_test) # calculate the predicted values
accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))

rmse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', mse)

r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy 0.359
RMSE: 1.7564643978749295
R^2: -0.0416
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(topic_probs, y, train_size=.
 ↪8, random_state=42)
# instantatiate the RFC with 100 ensemble members
clf = DecisionTreeClassifier(max_depth=10, random_state=42).fit(X_train,␣
 ↪y_train)

y_pred = clf.predict(X_test) # calculate the predicted values
accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))

rmse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', mse)

r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy 0.344
RMSE: 1.7564643978749295
R^2: -0.1112
```

[ ]:
```
row = {'no_estimators': 150, 'accuracy': accuracy}
for name, score in zip(topic_probs.columns, clf.feature_importances_):
    print(name,np.round(score,3))
```

```
topic_0 0.126
topic_1 0.065
topic_2 0.099
topic_3 0.095
topic_4 0.14
topic_5 0.084
topic_6 0.076
topic_7 0.102
topic_8 0.082
topic_9 0.131
```

With this model we get an accuracy score of 0.359, this is the highest accuracy that we've been able to achieve so far, however the RMSE is considerably higher than when we examined keyword frequency and sentiment analysis.

This is the best model that we have been able to achieve so far :)))

Let's try Combining all our features and seeing which features are the best at predicting outcome score

Final Model

[ ]:
```
features = df_with_topics.drop(columns={'OutcomeScore', 'content',␣
 ↪'clean_text', 'userIDs', 'tokens'})
features.head()
```

[ ]:

	wordCount	messages_sent	sentimentScore	think_frequency	agree_frequency	\
0	4	65	0.00	0	0	
1	13	65	0.00	0	0	
2	8	65	0.00	0	0	
3	8	65	0.00	0	0	
4	30	65	-0.43	0	0	

	surfactant_frequency	steric_frequency	cost_frequency	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

	prototype_frequency	flux_frequency	…	topic_0	topic_1	topic_2	\
0	0	0	…	0.035336	0.272511	0.444793	
1	0	0	…	0.018548	0.018546	0.018547	
2	0	0	…	0.774986	0.025005	0.025000	
3	0	0	…	0.774986	0.025005	0.025000	
4	0	0	…	0.786791	0.000000	0.000000	

	topic_3	topic_4	topic_5	topic_6	topic_7	topic_8	topic_9
0	0.035344	0.035336	0.035336	0.035336	0.035336	0.035336	0.035336
1	0.018547	0.018546	0.018551	0.018546	0.833074	0.018547	0.018547
2	0.025002	0.025000	0.025003	0.025001	0.025001	0.025001	0.025001
3	0.025002	0.025000	0.025003	0.025001	0.025001	0.025001	0.025001
4	0.000000	0.080226	0.082808	0.000000	0.000000	0.000000	0.000000

[5 rows x 23 columns]

[ ]:
```
y = messages['OutcomeScore']

X_train, X_test, y_train, y_test = train_test_split(features, y, train_size=.8,␣
 ↪random_state=42)
# instantatiate the RFC with 100 ensemble members
clf = RandomForestClassifier(n_estimators=50, random_state=42, max_depth=10,␣
 ↪max_samples=.2).fit(X_train, y_train)

y_pred = clf.predict(X_test) # calculate the predicted values

accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))
```

```
mse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', mse)

r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy 0.388
RMSE: 1.5384262731152567
R^2: 0.0081
```

[ ]:
```
y = messages['OutcomeScore']

X_train, X_test, y_train, y_test = train_test_split(features, y, train_size=.8,␣
 ↪random_state=42)
# instantatiate the RFC with 100 ensemble members
clf = DecisionTreeClassifier(random_state=42, max_depth=10).fit(X_train,␣
 ↪y_train)

y_pred = clf.predict(X_test) # calculate the predicted values

accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))

mse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', mse)

r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy 0.495
RMSE: 1.58426901476692
R^2: -0.0519
```

With this model we are achieve our highest accuracy score and our lowest RMSE score. We also achieve a positive R^2 for the first time which indicates that this model gets us closer to predicting OutcomeScore Accurately.

[ ]:
```
for name, score in zip(features.columns, clf.feature_importances_):
    print(name,np.round(score,3))
```

```
wordCount 0.023
messages_sent 0.819
sentimentScore 0.016
think_frequency 0.001
agree_frequency 0.0
surfactant_frequency 0.001
steric_frequency 0.001
```

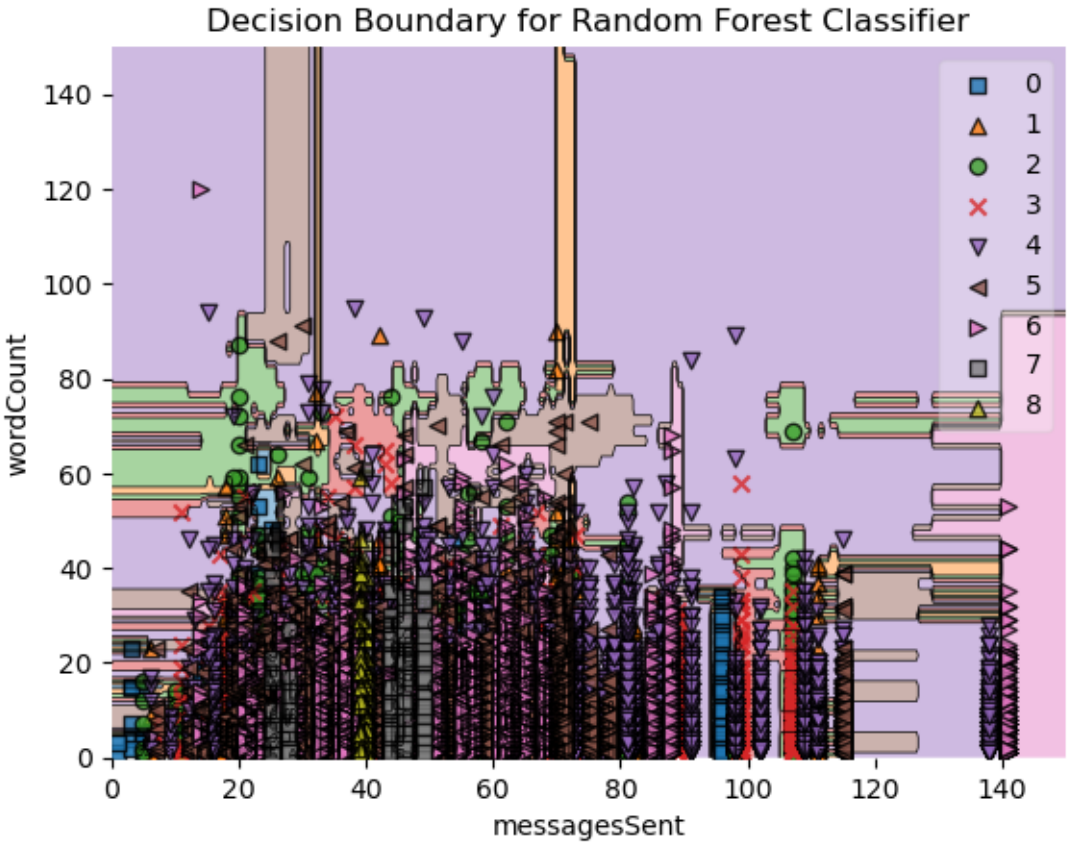
```
cost_frequency 0.002
prototype_frequency 0.004
flux_frequency 0.0
use_frequency 0.001
reliability_frequency 0.001
marketability_frequency 0.002
topic_0 0.01
topic_1 0.019
topic_2 0.008
topic_3 0.009
topic_4 0.015
topic_5 0.012
topic_6 0.018
topic_7 0.011
topic_8 0.013
topic_9 0.014
```

[ ]:
```
# let's take the best features that we found above


X = features[['messages_sent', 'wordCount']]

y = messages['OutcomeScore']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.8,␣
 ↪random_state=42)
# instantatiate the RFC with 100 ensemble members
clf = RandomForestClassifier(n_estimators=50, max_samples=.5, random_state=42).
 ↪fit(X_train, y_train)

y_pred = clf.predict(X_test) # calculate the predicted values
accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))

rmse = mean_squared_error(y_test, y_pred, squared=False)
print('Mean squared error:', rmse)
```

```
Accuracy 0.503
Mean squared error: 1.689608443778028
```

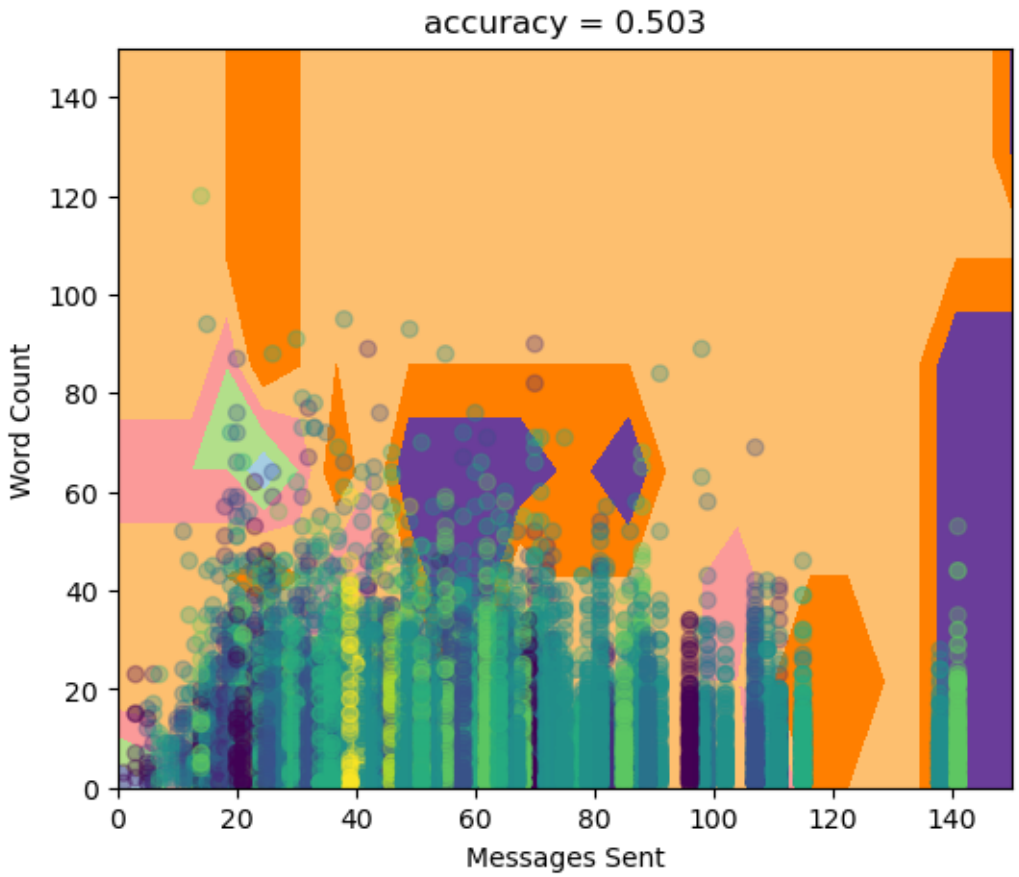
[ ]:
```
from mlxtend.plotting import plot_decision_regions

# Plot decision boundary
plot_decision_regions(X.values, y.values, clf=clf, legend=1)
plt.xlabel('messagesSent')
plt.ylabel('wordCount')
plt.title('Decision Boundary for Random Forest Classifier')
```

```
plt.xlim(0, 150)
plt.ylim(0, 150)
plt.show()
```



[ ]:
```
from logitplots import plt_decision_boundaries

fig, ax = plt.subplots(figsize=(6,5))
xx, yy = np.meshgrid(np.linspace(0, 300), np.linspace(0, 1050))

plt_decision_boundaries(clf,xx,yy)

plt.scatter(x=X['messages_sent'], y=X['wordCount'], c=y, alpha=0.3)
plt.title('accuracy = {0}'.format(accuracy))
plt.xlabel('Messages Sent')
plt.ylabel('Word Count');

plt.xlim(0, 150)
plt.ylim(0, 150)
```

```
plt.show()
```



[ ]:
```
X = messages[['messages_sent']]

y = messages['OutcomeScore']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.8,␣
 ↪random_state=42)
# instantatiate the RFC with 100 ensemble members
clf = RandomForestClassifier(n_estimators=50, max_samples=.5, random_state=42).
 ↪fit(X_train, y_train)

y_pred = clf.predict(X_test) # calculate the predicted values
accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))

rmse = mean_squared_error(y_test, y_pred, squared=False)
print('Mean squared error:', rmse)
```

```
r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy 0.513
Mean squared error: 1.644723937121495
R^2: -0.1337
```

Combining the above with the Dummy Variables provided in Original data set

[ ]:
```
chatData = pd.read_csv('VI_data.csv', encoding='latin1')

# We have no interest in predicting Mentor OutcomeScore as these are always 4.
# If kept in, will create a bias, therefore DROP.

chatData = chatData[chatData['RoleName'] == 'Player']

# Drop every column except the chat content, the outcome score and word count
messages2 = chatData.drop(columns={'Unnamed: 0', 'implementation', 'Line_ID',␣
 ↪'ChatGroup', 'group_id', 'roomName', 'RoleName'})


# # Create a column for the number of messages each student sent:
message_counts = messages2.groupby("userIDs")["content"].count().reset_index().
 ↪rename(columns={'content':'messages_sent'})

messages2 = pd.merge(messages2, message_counts, on='userIDs')
messages2.head(10)

messages2['clean_text'] = messages2['content'].apply(preprocess_message)
messages2['sentimentScore'] = messages2['clean_text'].
 ↪apply(get_sentiment_scores)

for keyword in keywords:
    messages2[f'{keyword}_frequency'] = messages2['content'].apply(lambda x: x.
 ↪count(keyword))

messages2['tokens'] = messages2['clean_text'].apply(word_tokenize)

# Create a document-term matrix
dictionary = corpora.Dictionary(messages2['tokens'])
doc_term_matrix = [dictionary.doc2bow(doc) for doc in messages2['tokens']]

# Train the LDA model
num_topics = 10
lda_model = models.LdaModel(doc_term_matrix, num_topics=num_topics,␣
 ↪id2word=dictionary)
```

```
topic_probs = pd.DataFrame(columns=['topic_{}'.format(i) for i in␣
 ↪range(lda_model.num_topics)])

# Loop over each message and obtain its topic distribution probabilities
for index, row in messages2.iterrows():
    message = row['clean_text']
    bow_vector = lda_model.id2word.doc2bow(message.split())
    topic_distribution = lda_model.get_document_topics(bow_vector)

    # Fill in the corresponding topic probabilities for the current message
    probs_dict = dict(topic_distribution)
    topic_probs.loc[index] = [probs_dict.get(i, 0.0) for i in range(lda_model.
 ↪num_topics)]

# Concatenate the topic probabilities DataFrame with the original DataFrame
df_with_topics = pd.concat([messages2, topic_probs], axis=1)
```

[ ]:
```
all_features = df_with_topics.drop(columns={'OutcomeScore', 'content',␣
 ↪'clean_text', 'userIDs', 'tokens'})
all_features
```

[ ]:
```
       m_experimental_testing  m_making_design_choices  m_asking_questions  \
0                           0                        0                   0
1                           0                        0                   1
2                           0                        0                   0
3                           0                        0                   1
4                           0                        0                   0
…                         …                        …                 …
16897                       0                        0                   0
16898                       0                        0                   0
16899                       0                        0                   0
16900                       0                        0                   0
16901                       0                        0                   0

       j_customer_consultants_requests  j_performance_parameters_requirements  \
0                                    0                                      0
1                                    0                                      0
2                                    0                                      0
3                                    0                                      0
4                                    0                                      0
…                                  …                                    …
16897                                0                                      0
16898                                0                                      0
16899                                0                                      0
16900                                0                                      0
16901                                0                                      0
```

```
       j_communication  wordCount  messages_sent  sentimentScore  \
0                    0          4             65            0.00
1                    0         13             65            0.00
2                    0          8             65            0.00
3                    0          8             65            0.00
4                    0         30             65           -0.43
…                  …        …            …              …
16897                0          2             72            0.00
16898                0         26             72            0.25
16899                0         16             72            0.00
16900                0         15             72            0.00
16901                0          1             72            0.40

       think_frequency  …   topic_0   topic_1   topic_2   topic_3   topic_4  \
0                    0  …  0.034407  0.034408  0.034407  0.034407  0.034412
1                    0  …  0.018542  0.018542  0.833116  0.018542  0.018542
2                    0  …  0.025002  0.025002  0.428137  0.025007  0.025003
3                    0  …  0.025002  0.025002  0.427982  0.025007  0.025003
4                    0  …  0.065662  0.000000  0.151246  0.109211  0.000000
…                  …  …       …        …       …        …       …
16897                0  …  0.050001  0.549986  0.050001  0.050005  0.050001
16898                0  …  0.000000  0.000000  0.000000  0.260968  0.000000
16899                0  …  0.014286  0.014288  0.322714  0.014292  0.014287
16900                0  …  0.012501  0.344803  0.012500  0.012501  0.373739
16901                0  …  0.100000  0.099999  0.100000  0.100001  0.100000

        topic_5   topic_6   topic_7   topic_8   topic_9
0      0.034407  0.346016  0.034407  0.378722  0.034407
1      0.018544  0.018542  0.018543  0.018542  0.018543
2      0.025011  0.025003  0.025004  0.371826  0.025004
3      0.025011  0.025003  0.025004  0.371982  0.025004
4      0.130874  0.395324  0.000000  0.000000  0.119071
…         …       …        …        …        …
16897  0.050001  0.050001  0.050001  0.050003  0.050001
16898  0.000000  0.000000  0.000000  0.000000  0.677655
16899  0.014286  0.014286  0.562981  0.014286  0.014294
16900  0.012500  0.012501  0.193947  0.012503  0.012504
16901  0.100000  0.100001  0.100000  0.100000  0.099999

[16902 rows x 29 columns]
```

[ ]:
```
y = messages2['OutcomeScore']

X_train, X_test, y_train, y_test = train_test_split(all_features, y,␣
 ↪train_size=.8, random_state=42)
# instantatiate the RFC with 100 ensemble members
```

```
clf = RandomForestClassifier(n_estimators=50, random_state=42, max_depth=10,␣
 ↪max_samples=.5).fit(X_train, y_train)

y_pred = clf.predict(X_test) # calculate the predicted values

accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))

mse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', mse)

r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy 0.391
RMSE: 1.5365025181135177
R^2: 0.0106
```

[ ]:
```
y = messages2['OutcomeScore']

X_train, X_test, y_train, y_test = train_test_split(all_features, y,␣
 ↪train_size=.8, random_state=42)
# instantatiate the RFC with 100 ensemble members
clf = DecisionTreeClassifier(random_state=42, max_depth=10).fit(X_train,␣
 ↪y_train)

y_pred = clf.predict(X_test) # calculate the predicted values

accuracy = np.round(accuracy_score(y_test, y_pred),3)
# print the accuracy of the RFC
print('Accuracy {0}'.format(accuracy))

mse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', mse)

r2 = r2_score(y_test, y_pred)
print("R^2: %.4f" % r2)
```

```
Accuracy 0.515
RMSE: 1.5412114526541674
R^2: 0.0045
```

[ ]:
```
for name, score in zip(all_features.columns, clf.feature_importances_):
    print(name,np.round(score,3))
```

```
m_experimental_testing 0.004
m_making_design_choices 0.002
m_asking_questions 0.006
```

```
j_customer_consultants_requests 0.0
j_performance_parameters_requirements 0.001
j_communication 0.001
wordCount 0.025
messages_sent 0.82
sentimentScore 0.017
think_frequency 0.001
agree_frequency 0.001
surfactant_frequency 0.0
steric_frequency 0.0
cost_frequency 0.002
prototype_frequency 0.004
flux_frequency 0.0
use_frequency 0.0
reliability_frequency 0.001
marketability_frequency 0.0
topic_0 0.008
topic_1 0.007
topic_2 0.01
topic_3 0.018
topic_4 0.012
topic_5 0.011
topic_6 0.023
topic_7 0.008
topic_8 0.01
topic_9 0.006
```

[ ]: