ADS2001 Final Project Report
Virtual Internship

Chow Chun Kei (33520771)

Deng Yuxuan (34254900)

Huang Xiaohui (34518053)

Koh Xuan Qing (33521719)

Lim Hui Shawn (34017259)

Sunaina Rayaprol (33944091)

Wesley Kong Zhe Hung (33521743)

Table of Contents

Executive Summary

Problem statement

The goal of this project is to investigate how communication patterns within student teams during a virtual engineering internship influenced their performance on their final design report. Specifically, we seek to model the relationship between various discourse features, such as design choices, justifications, and more that were captured in the chat logs, and the quality of the team's final design report. The dataset originates from *Nephrotex*, which is a virtual internship simulation where biomedical engineering students collaborate online to design a medical device for patients with kidney failure. Each team engages in structured design tasks and interacts via a chat tool, with some guidance from the assigned mentors. By aggregating individual-message-level data into team-level statistics and applying various data modeling techniques, this project aims to determine which types of discourse are associated with higher report scores. Another objective is to explore the influence of mentors on team performance to draw actionable insights into how specific communication affects the outcome score for students.

Methodology

This study aims to predict team-level performance in the Nephrotex virtual internship by analyzing chat transcripts between players and mentors. The dataset contains chat logs from 75 teams. Initial data preprocessing involved removing duplicate columns and imputation of missing values. Since the original data was unaggregated and only consisted of individual chat instances by every player, the chat messages were then aggregated at the team level to facilitate group-wise analysis, leaving us with only 75 unique data points.

Exploratory data analysis (EDA) was conducted to investigate the relationships between features and team outcomes. Correlation matrices and boxplots helped identify key features and revealed class imbalance issues within the dataset. To address this, stratified sampling was used while splitting data into training and testing datasets.

Feature engineering was performed in two stages: basic and extensive. Basic features included simple metrics like word count categories and sentiment scores (positive, negative, and neutral). For extensive feature engineering, advanced natural language processing techniques were applied. Latent Dirichlet Allocation (LDA) was used to identify underlying discussion topics within team chats. Additional features such as sentiment consistency (measured by the standard deviation of sentiment polarity), lexical diversity (vocabulary richness), and topic coherence (semantic clarity) were computed. Other relevant features included the ratio of mentor to player messages and the frequency of terms related to experimental testing and design choices. Each of these NLP-derived features was aggregated at the team level to align with the target metric.

A robust features selection framework was also proposed to address the issue of increasing dimensionality of features subset after extensive features engineering. This involves the three essential stages for features selection, namely filter, wrapper and embedded stage. A total of 15 features were deliberately selected and used to train the machine learning models.

For model development, several classifiers were trained and tested, including Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree, Random Forest, and AdaBoostClassifier. Model performance was evaluated using accuracy, precision, recall and F1-score metrics. Initial models exhibited overfitting and poor generalization. However, after applying extensive feature engineering and meticulous features selection, model performance improved significantly, especially for ensemble methods like AdaBoostClassifier, reducing overfitting and increasing predictive accuracy.

Findings

By combining insights derived from EDA and machine learning (ML) modeling, the analysis revealed key patterns in team communication behavior and their relationship to the team's report performance.

The initial EDA focused on understanding chat dynamics, message characteristics, and team composition. From EDA, in general it was observed that the mentors were ineffective at improving team performance, suggesting the dominant role of peer collaboration in shaping outcomes. High-performing teams tended to have more medium-length and elaborate messages. Sentiment analysis further revealed that teams with consistently positive sentiment tended to perform better. Topic modeling (LDA) uncovered semantically distinct clusters of messages, including themes such as "design choices," "prototype discussion," and "cost and reliability," with some topics being more prevalent among higher-performing teams.

Building on EDA, several ML models were developed to classify teams into outcome categories. Initial models that used basic features, like word counts, sentiment scores, and average message length, suffered from overfitting. Tree-based models like Decision Tree and Random Forest achieved perfect training accuracy but failed to generalize, with test accuracies falling below 0.35. KNN performed poorly due to class imbalance, and even AdaBoost, while more stable, had limited success, indicating a need for richer feature representation.

Subsequently, advanced feature engineering and selection significantly improved model performance. Features such as topic distributions (LDA topics), sentiment consistency (standard deviation), lexical diversity, and topic coherence emerged as top predictors in the Random Forest model. AdaBoostClassifier achieved a marked improvement with reduced overfitting, showing a test accuracy gain of ~55%. The most influential predictors were all related to player chat content, confirming the hypothesis that active, thoughtful player communication strongly correlates with team success. In contrast, mentor contributions had only marginal influence on prediction performance, suggesting their role is supportive rather than determinative.

Overall, the findings underscore that collaborative, reflective and content-rich discourse among team members is a pivotal factor driving success in virtual internships, surpassing mere chat volume or mentor presence.

1. Introduction

1.1. Background

Nephrotex is a virtual internship program developed at the University of Wisconsin-Madison. It is a virtual simulation of an internship at a fictitious biomedical engineering firm (Virtual Internships, n.d.). This program was offered to first year students as a half-semester module to study in-depth a single engineering topic (AMD NextGen Engineer, 2012). In Nephrotex, students are required to design a nanotechnology-based membrane used in kidney dialysis (Virtual Internships, n.d.). At the end of the internship, each student is to produce a final report on their work. The objective of this internship was to encourage students to persist in their career paths in engineering by having a positive view and better understanding of an engineer's daily tasks (AMD NextGen Engineer, 2012).

Each implementation of Nephrotex consists of five preset teams to which students are allocated into. Then, each team contains 3 to 7 players (the students), with 1 or 2 mentors to help lead reflective discussions.

The chat data across several implementations of Nephrotex were collected. By knowing which discourse features affect student final report performance, future implementations of the virtual internship can promote more effective discussions. This will lead to a more fruitful experience for every Nephrotex player. Therefore, by utilizing various data science techniques, this project aims to analyze the collected data at team-level to uncover the factors influencing students' final report marks.

1.2. Data Overview

The dataset provided for this project is a single CSV file: *virtualInternshipData_ADS2001.csv.* This dataset contains 17 variables and 19180 observations. Most of the variables are categorical. These data are the chat history of 75 teams collected over the first half of the virtual internships from 15 different implementations.

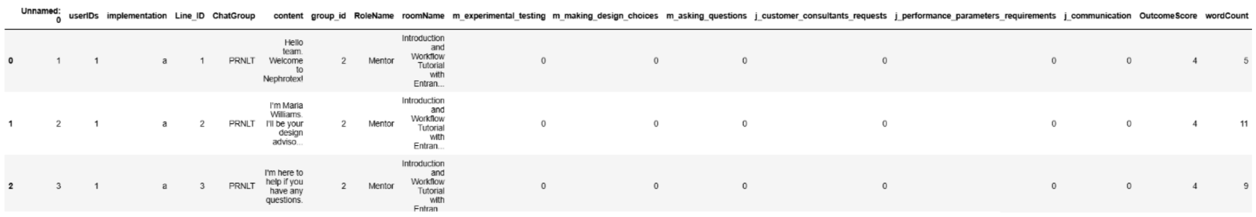
The variables in the dataset are: *userIDs* (a unique numerical ID for each chatter), *implementation* (unique alphabetical ID for each implementation; ranges from *a* to *o*), *Line_ID* (a unique numerical ID for each chat message), *ChatGroup* (the team of the chatter; *PRNLT, PMMA, PSF, PAM,* or *PESPVP*), *content* (the content of each chat message), *group_id* (a numerical ID corresponding to each *ChatGroup*), *RoleName* (role of the chatter; either *Mentor* or *Player*), *roomName* (the activity the chatter was participating in: *Introduction and Workflow Tutorial with Entrance Interview*, *Background research on dialysis*, *Graphing Surfactant Data*, *Reflection team discussion of surfactants*, *Summarize internal consultant requirements*, *Choose consultants to analyze, Individuals design 5 prototypes*, *Team designs batch using 1 material*, *Individual analysis of first batch*, or *Reflection team discussion of first batch results*), wordCount (the number of words in each *content*). Then, the discourse variables that have values of 1 if present or 0 if absent based on each *content* entry: *m_experimental_testing* (talk about using experimental techniques to understand technical aspects of a design), *m_making_design_choices* (talk about choosing specifications for a design), *m_asking_questions* (asking questions), *j_customer_consultants_requests* (talk about justifying design choices to meet consultant requests), *j_performance_parameters_requirements* (talk about justifying design choices based on performance parameters), *j_communication* (talk about justifying design choices by facilitating communication among engineers). Finally, *OutcomeScore* which is the final report mark of the chatter, ranging from integers 0 (low) to 8 (high).

1.3. Expectation and approach

We expect to find a positive influence of mentor involvement on the team-level outcome score, as mentor guidance is likely to enhance team performance in a virtual internship setting. Additionally, higher overall communication volume, regardless of topic, is anticipated to correlate with better outcomes, reflecting stronger engagement, collaboration, and problem-solving among team members. A significant challenge in this project was the unstructured nature of the raw chat data, which existed at the individual message level and lacked direct labels. To enable meaningful analysis, the data was transformed into structured, team-level aggregates. Key natural language processing (NLP) techniques, such as Latent Dirichlet Allocation (LDA), were used to extract topic distributions, while Singular Value Decomposition (SVD) reduced the dimensionality of the text representations and captured latent semantic patterns. These features were then aggregated to characterize team communication dynamics. A robust pipeline of features engineering and selection is also expected to be implemented to select the most significant features to be trained in the machine learning model to enhance predictivity power. Another modelling challenge involved managing the high dimensionality and sparsity of the text-derived features, which was addressed through dimensionality reduction techniques. Furthermore, the relatively small sample size (n = 75 teams) limited model complexity and increased the risk of overfitting. To mitigate this, we selected interpretable and robust models, including random forest and Ada Boost Classifier, and employed careful cross-validation. Overall, our approach focused on extracting structured signals from unstructured chat logs and assessing their relationship with team success.

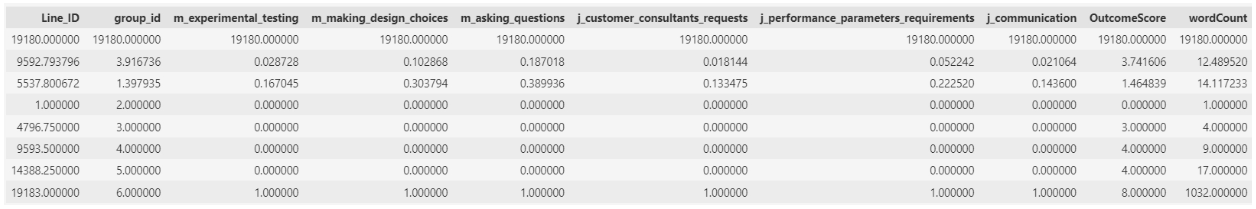
2. Data Quality

The first few observations of our dataset are shown in Figure 1.1. Fortunately, our dataset had no formatting errors—no inconsistencies in variable names nor any inconsistent variable data types. The descriptive statistics for this dataset (Fig. 1.2) also does not contain any conflicting values, and the range of values are all reasonable.



	Unnamed: 0	userIDs	implementation	Line_ID	ChatGroup	content	group_id	RoleName	roomName	m_experimental_testing	m_making_design_choices	m_asking_questions	j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	OutcomeScore	wordCount
0	1	1	a	1	PRNLT	Hello team. Welcome to Nephrotex!	2	Mentor	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	4	5
1	2	1	a	2	PRNLT	I'm Maria Williams. I'll be your design adviso...	2	Mentor	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	4	11
2	3	1	a	3	PRNLT	I'm here to help if you have any questions.	2	Mentor	Introduction and Workflow Tutorial with Entran	0	0	0	0	0	0	4	9

Figure 1.1: *Head of the original dataset*

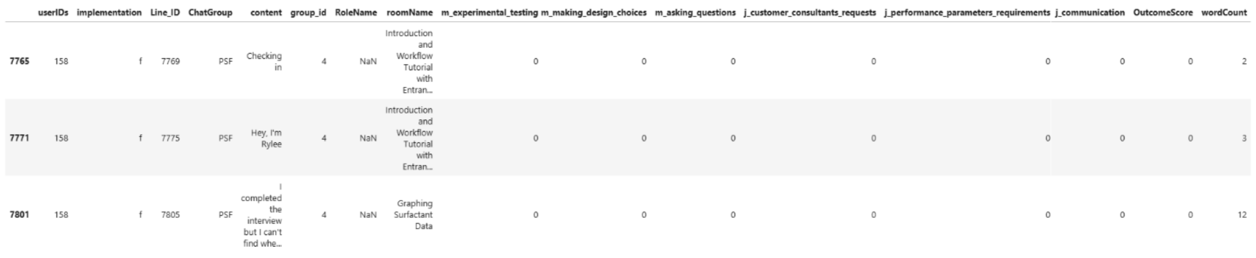


Line_ID	group_id	m_experimental_testing	m_making_design_choices	m_asking_questions	j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	OutcomeScore	wordCount
19180.000000	19180.000000	19180.000000	19180.000000	19180.000000	19180.000000	19180.000000	19180.000000	19180.000000	19180.000000
9592.793796	3.916736	0.028728	0.102868	0.187018	0.018144	0.052242	0.021064	3.741606	12.489520
5537.800672	1.397935	0.167045	0.303794	0.389936	0.133475	0.222520	0.143600	1.464839	14.117233
1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
4796.750000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.000000	4.000000
9593.500000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.000000	9.000000
14388.250000	5.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.000000	17.000000
19183.000000	6.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	8.000000	1032.000000

Figure 1.2: *Descriptive statistics of original dataset, for numerical features*

Issues

Our dataset contained some duplicate data and missing values. First, there was an index column in the csv file, thus once imported into a dataframe, we end up with two identical index columns (one generated from creating a pandas dataframe, one from the original dataset). This can be seen in Figure 1.1 where the index column and first column are identical. Furthermore, of the entire dataset, there were 3 observations that contained missing values (Fig. 1.3). This presence of missing values will hinder our processing if left alone.



	userIDs	implementation	Line_ID	ChatGroup	content	group_id	RoleName	roomName	m_experimental_testing	m_making_design_choices	m_asking_questions	j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	OutcomeScore	wordCount
7765	158	f	7769	PSF	Checking in	4	NaN	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	0	2
7771	158	f	7775	PSF	Hey, I'm Rylee	4	NaN	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	0	3
7801	158	f	7805	PSF	I completed the interview but I can't find whe...	4	NaN	Graphing Surfactant Data	0	0	0	0	0	0	0	12

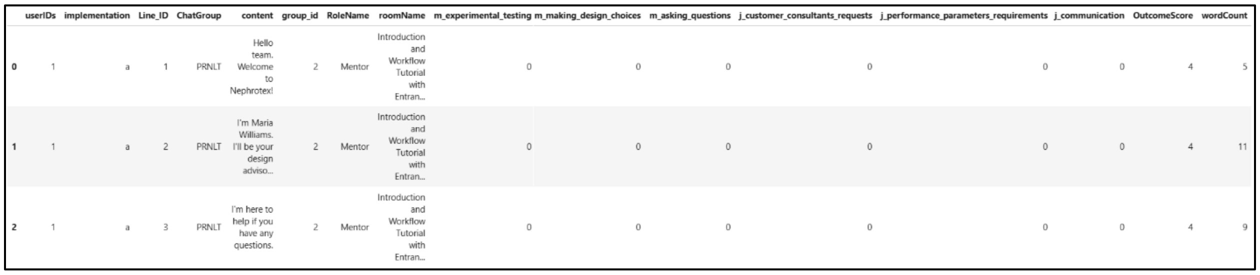
Figure 1.3: *The observations with missing values in the dataset (in column RoleName)*

2.1. Data cleaning

Good quality data ensures reliability and accuracy. Therefore, before further processing, the aforementioned data quality issues were cleaned up accordingly.

Removing duplicate data

We dropped the additional index column that came with the dataset to remove unnecessary, duplicate data. The first few observations of our dataset with the duplicate index column removed is shown in Figure 1.4.

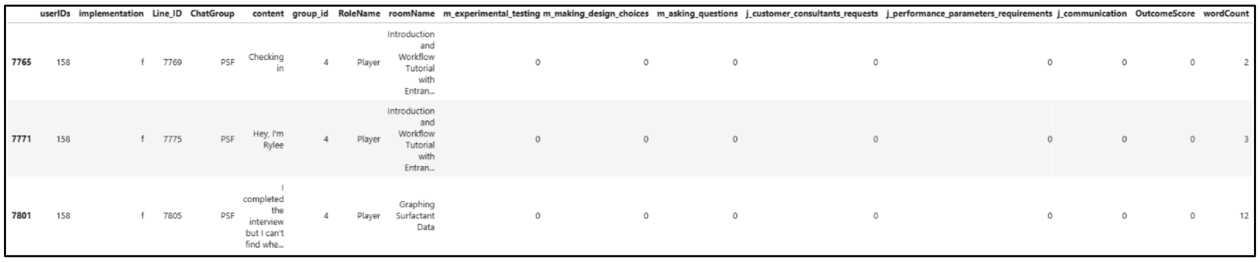


	userIDs	implementation	Line_ID	ChatGroup	content	group_id	RoleName	roomName	m_experimental_testing	m_making_design_choices	m_asking_questions	j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	OutcomeScore	wordCount
0	1	a	1	PRNLT	Hello team. Welcome to Nephrotex!	2	Mentor	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	4	5
1	1	a	2	PRNLT	I'm Maria Williams. I'll be your design adviso...	2	Mentor	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	4	11
2	1	a	3	PRNLT	I'm here to help if you have any questions.	2	Mentor	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	4	9

Figure 1.4: *Head of the original dataset with duplicate index column removed*

Imputation

Upon further inspection, the observations with missing values are only those involving the *RoleName* for *userIDs* of value *158*. Investigating the *content* i.e. the messages of this user, their message on *Line_ID 7805* ("I completed the interview but I can't find where the notebook is...") indicates this user is a player, not a mentor. This is because a mentor would not participate in player activities like interviews, nor be confused regarding details of the activities. Therefore, these missing values were manually imputed to *Player* (Fig. 1.5).



	userIDs	implementation	Line_ID	ChatGroup	content	group_id	RoleName	roomName	m_experimental_testing	m_making_design_choices	m_asking_questions	j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	OutcomeScore	wordCount
7765	158	f	7769	PSF	Checking in	4	Player	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	0	2
7771	158	f	7775	PSF	Hey, I'm Rylee	4	Player	Introduction and Workflow Tutorial with Entran...	0	0	0	0	0	0	0	3
7801	158	f	7805	PSF	I completed the interview but I can't find whe...	4	Player	Graphing Surfactant Data	0	0	0	0	0	0	0	12

Figure 1.5: *The missing values in RoleName previously now imputed to "Player"*

3. Model Development

The aim of this project is to determine the key discourse features influencing students' report marks at team level. Thus, the model development in this project is focused on feature selection.

Three stages were involved in the model development for this project. First, the modelling was executed after some preliminary exploratory data analysis (EDA) and basic feature engineering. However, the model performances were mediocre (detailed in the *Results* section of the report). Thus, further extensive feature engineering was implemented. Finally, feature selection using filter, wrapper and embedded methods was conducted again on the high-dimensional data to select the most significant features to be trained again into the models

3.1. Before Extensive Feature Engineering

3.1.1. Basic Feature Engineering

Word Count

From the feature 'wordCount', we derived the feature 'message_length_category' which categorizes the different levels of word count (Figure A1). Messages with words lesser than 20 are categorized as 'short', words more than 50 are categorized as 'long' and everything in between as 'medium'. This categorization was chosen to provide a simple yet effective way to differentiate between brief, moderate, and extensive messages, enabling more nuanced analysis of message length patterns.

Natural Language Processing (NLP)

We were introduced to NLP by our lecturer as extra knowledge that could prove substantial in this project. According to Khurana et al. (2022), NLP is a field of Artificial Intelligence and Linguistics, aimed to facilitate the comprehension of statements or words written in human languages by computers. This comprehension is then used by computers analyze textual data and uncover further insights into communication patterns. Since the dataset of this project consists of chat logs between the players and mentors, we decided that it was appropriate to utilize NLP to create more features for our dataset.

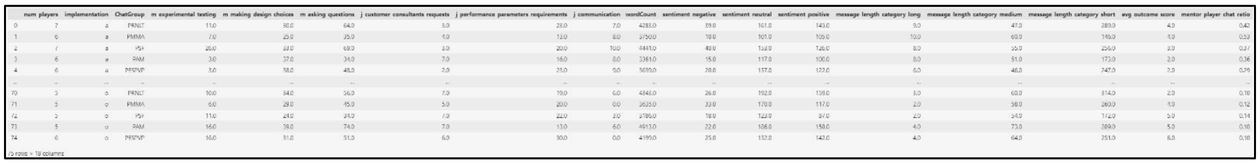
One NLP method we used was sentiment analysis, which identifies and categorizes emotions or opinions expressed in text (Mao et al., 2024). This feature seemed appropriate to add as it can show how the emotions of people in a group can affect their ability to communicate and by extension the outcome score. By applying sentiment analysis to the chat logs, we were able to assign sentiment scores like positive, negative, or neutral to each message exchanged between players and mentors. These sentiment scores were then transformed using one-hot encoding to generate 3 new sentiment features *'sentiment_positive', sentiment_negative',* and *'sentiment_neutral'* (Figure A2).

3.1.2. Conversion of single-message level data to team level statistics

This project involved analysis of the data at team-level. To do this, the data was first converted into team-level statistics by the code in Appendix B1. This code aggregates each feature at team-level for each team and adds each observation to a newly created team-level dataframe.

The resulting features are: *num_players* (number of players in the team), *implementation, ChatGroup, m_experimental_testing, m_making_design_choices, m_asking_questions, j_customer_consultants_requests, j_performance_parameters_requirements, j_communication, wordCount, sentiment_negative, sentiment_neutral, sentiment_positive, message_length_category_long, message_length_category_medium, message_length_category_short,* and additionally two new features *mentor_player_chat_ratio* (the mentor to player chat ratio in the team, created to account for the mentor's influence on the team) and *avg_outcome_score* (the mean of the students' individual final report marks for players of the team; ranges from integers 2.0 to 6.0). The mean was used as the metric for the team-level outcome scores because it standardizes the comparisons between teams, as the mean is not influenced by different team sizes. This results in a total of 18 features which are mainly a mix of discrete numerical counts and continuous features.

Through this process, the 19180 single chat message observations were used to create a team-level dataframe of 75 observations, where each observation represents the statistics for each of the 75 teams. The resulting dataframe, as in Figure 2.1, is used hereon for further analysis with the team-level final report marks, *avg_outcome_score* as our target variable.



	num players	implementation	ChatGroup	m experimental testing	m making design choices	m asking questions	j customer consultants requests	j performance parameters requirements	j communication	wordCount	sentiment negative	sentiment neutral	sentiment positive	message length category long	message length category medium	message length category short	avg outcome score	mentor player chat ratio
0	7	a	PRNLT	11.0	30.0	64.0	5.0	23.0	7.0	4283.0	39.0	161.0	143.0	9.0	47.0	289.0	4.0	0.42
1	6	a	PMMA	7.0	25.0	35.0	4.0	13.0	8.0	3750.0	10.0	101.0	105.0	10.0	60.0	146.0	4.0	0.53
2	7	a	PSF	26.0	33.0	69.0	3.0	20.0	10.0	4441.0	40.0	153.0	126.0	8.0	55.0	256.0	3.0	0.37
3	6	a	PAM	3.0	37.0	34.0	7.0	16.0	8.0	3361.0	15.0	117.0	100.0	8.0	51.0	173.0	2.0	0.36
4	6	a	PESPVP	3.0	38.0	48.0	2.0	23.0	9.0	3639.0	20.0	157.0	122.0	6.0	46.0	247.0	2.0	0.29
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
70	5	o	PRNLT	10.0	34.0	56.0	7.0	19.0	6.0	4343.0	26.0	192.0	159.0	3.0	60.0	314.0	2.0	0.10
71	5	o	PMMA	6.0	29.0	45.0	5.0	20.0	0.0	3635.0	33.0	170.0	117.0	2.0	58.0	260.0	4.0	0.12
72	5	o	PSF	11.0	24.0	34.0	7.0	22.0	3.0	3186.0	18.0	123.0	87.0	2.0	54.0	172.0	5.0	0.14
73	5	o	PAM	16.0	39.0	74.0	7.0	13.0	6.0	4913.0	22.0	186.0	158.0	4.0	73.0	289.0	5.0	0.10
74	6	o	PESPVP	16.0	31.0	51.0	6.0	30.0	0.0	4199.0	25.0	152.0	142.0	4.0	64.0	251.0	6.0	0.10

75 rows × 18 columns

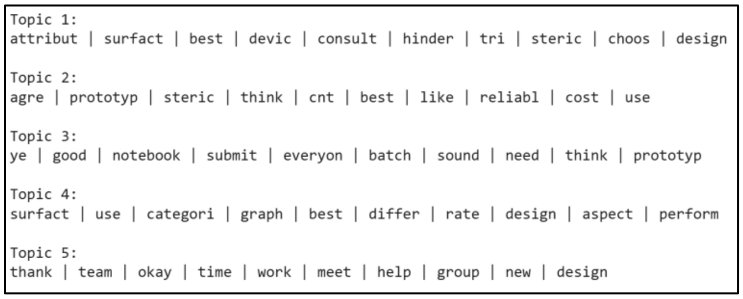
Figure 2.1: *Preview of the resulting team-level statistics dataframe*

3.1.3. Initial EDA

EDA is an essential process to becoming familiar with the data. This process enables data scientists to uncover existing trends in the data, thus informing subsequent procedures. In this project, we conducted analysis on the textual data (*content* column of the original dataset) and brief correlation analysis to visualize how discourse features influence the team outcome scores.

Analysis of textual data

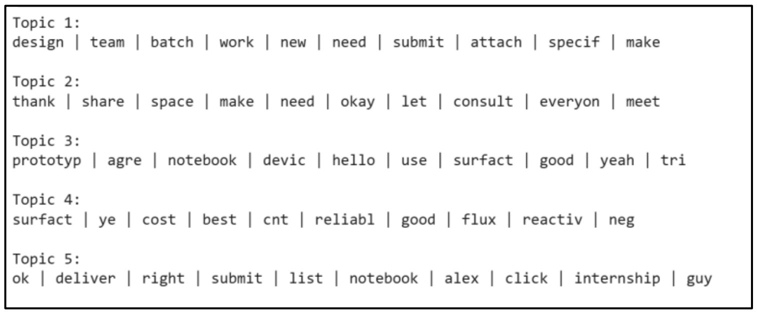
Since a significant portion of our dataset consists of chat logs between the players and mentors, we applied LDA topic modelling to find outstanding chat topics from the chat data. Our goal was to uncover any patterns in communication that might correlate with team performance, in other words, their outcome score. Therefore, we ran the model for the top 5 and bottom 5 performing teams to check for any differences.



```
Topic 1:
attribut | surfact | best | devic | consult | hinder | tri | steric | choos | design

Topic 2:
agre | prototyp | steric | think | cnt | best | like | reliabl | cost | use

Topic 3:
ye | good | notebook | submit | everyon | batch | sound | need | think | prototyp

Topic 4:
surfact | use | categori | graph | best | differ | rate | design | aspect | perform

Topic 5:
thank | team | okay | time | work | meet | help | group | new | design
```

Figure 2.2: *The topics obtained from running the LDA model for the top 5 performing teams from the dataset.*

Fig. 2.2 shows the output we got from applying LDA to the top performing teams. Topic 1 shows us that there is frequent use of evaluative language such as ‘attribute’, ‘consult’, etc. This tells us that the teams who performed well were doing more technical analysis, which is backed by the main words used in Topic 2. In Topic 3 and 4, we can observe decision-making language such as ‘need’ and ‘use’, which could imply that the teams had conversations based on collaborating to make decisions. Finally, Topic 5 mainly contains words like ‘thank’, ’help’ and more which tells us that collaboration was present in these teams. Overall, it is evident that the well performing teams excelled in communication and collaborative decision making.



```
Topic 1:
design | team | batch | work | new | need | submit | attach | specif | make

Topic 2:
thank | share | space | make | need | okay | let | consult | everyon | meet

Topic 3:
prototyp | agre | notebook | devic | hello | use | surfact | good | yeah | tri

Topic 4:
surfact | ye | cost | best | cnt | reliabl | good | flux | reactiv | neg

Topic 5:
ok | deliver | right | submit | list | notebook | alex | click | internship | guy
```

Figure 2.3: *The topics obtained from running the LDA model for the bottom 5 performing teams from the dataset.*

Conversely, Fig. 2.3 is largely task-level and execution focused, with frequent use of simple, functional terms like ‘submit’, ok’ and ‘guy’ as seen in Topic 1 and 5. This suggests that team discussions were primarily centered around completing tasks rather than engaging in deeper analysis or reflective dialogue. The tone is generally casual, and the phrasing leans toward passivity, with little evidence of critical thinking or intentional design decisions. This is mainly seen in Topics 2 and 3 where we see words like ‘okay’ and ‘yeah’. Topic 4 is the only topic where we can observe some technical terms such as ‘surfact’ and ‘reliable’. These patterns in conversations may indicate a lack of strategic planning or problem-solving focus, which could have contributed to their lower performance.

Correlation analysis

We generated the correlations scores between all features of our data, filtering out the correlations with absolute score below 0.2 (Appendix C1). An absolute score below 0.2 indicates a very weak relationship and is thus negligible (Papageorgiou, 2022). Among the correlations with our target*,* only the discourse feature *m_experimental_testing* is weakly, positively correlated to our target *avg_outcome_score* with a correlation value of 0.35 (weak correlations are absolute values between 0.20 to 0.40, according to Papageorgiou (2022)). This contradicts our expectations; we assumed that all discourse features would be positively correlated with the target. Hence, we further investigate these relationships by visualization using boxplots which allow us to observe the spread of data across a feature by each class.

i. Investigating relationship between original discourse features and target

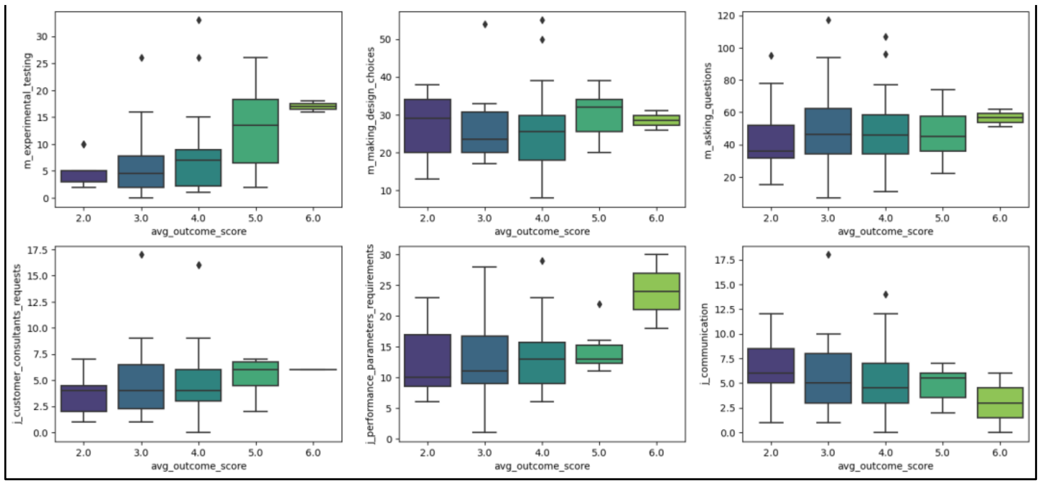


Figure 2.4: *Boxplots of original discourse features against target (avg_outcome_score)*

Indeed, of the original discourse features provided, only *m_experimental_testing* has a clearer distinction between classes for the distribution of observations (Fig. 2.4). The remaining features have most of the data within similar ranges regardless of the target class. So, the original discourse features alone are insufficient to determine the score for a given team. This may be due to the original discourse features not representing the chat messages topics accurately. Thus, additional feature engineering from the original textual data (*content* column) may be needed to produce an accurate model.

ii. Investigating the mentor's influence on the outcome scores

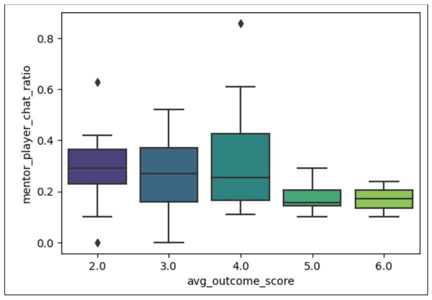


Figure 2.5: *Boxplot of mentor_player_chat_ratio against avg_outcome_score*

Furthermore, we expected to find a positive relationship between the mentor's effect on the group (as represented by *mentor_player_chat_ratio*) and our target, *avg_outcome_score*. Contrary to our expectations, from Figure 2.5, it seems like this relationship is slightly negative and very weak. This also aligns with its absolute correlation score below 0.20 from the heatmap in Appendix C1. This indicates that the virtual internships have ineffective mentorship; perhaps the mentors lack adequate training, which ended up confusing and thus discouraging the students from performing better (Tuma et al., 2021).

3.1.4. Model Building

Baseline Model

To develop appropriate predictive models for estimating group outcome scores, the dataset is aggregated to contain various group-level features based on chat-based interaction. We removed non-predictive features (*avg_outcome_score, implementation, Chatgroup*) before model training. Since *avg_outcome_score* is the target variable, including it as an input feature would result in data leakage, where the model has access to information it is supposed to predict. This would artificially inflate performance and prevent the model from generalizing to unseen data. *Implementation and Chatgroup* were removed prior to model training because they are identifier variable, not meaningful predictors, *Implementation* represents a unique ID for each design submission, which carries no inherent information about team communication or performance, and *Chatgroup* represents non-unique team ID, and does not relate to communication patterns we aim to model.

We first applied several baseline classification models, including Logistic Regression, K-nearest Neighbors (KNN), Decision Tree, and Random Forest. These are commonly used supervised machine learning algorithms suitable for classification tasks. Among them, Logistic Regression provides a simple linear decision boundary and serves as a suitable starting point for comparison. KNN, being a distance-based model, is sensitive to the scale of features, features with large values may dominate the distance calculation, therefore we applied feature normalization to avoid biased results (Bishnoi et al., 2022). On the other hand, Decision Tree and Random Forest are tree-based models that make splits based on feature threshold rather than distances, meaning they are unaffected by the scale of input of features and do not require normalization. Testing these baseline models allowed us to evaluate how well standard approaches perform using our current features and highlighted the need for more advanced methods and better feature engineering in the next steps. However, model performance generally does not meet our expectations (details at Section 4: Results), thus further steps have to be implemented to boost the performance of models.

3.2. Extensive Feature Engineering

3.2.1. Features Engineering using Term Frequency-Inverse Document Frequency (TF-IDF) and Polynomial Interaction Features

Due to poor model performance, we hypothesized that there were still not enough important features being trained in the model. Hence, we decided to engineer the features extensively to break down every word of chat content. In this context, TF-IDF approach was used to convert text data into numerical matrix with a certain weightage based on the importance of words. As asserted by Qaiser and Ali (2018), the TF formula is used to check word frequency in text data while the IDF formula is applied to measure how rare or important the word is. The main intention of doing so was to extract every single meaningful and unique individual message. In this way, linguistic signals and communication patterns can be extracted that reflect students' behavior. For instance, frequent words like 'prototype' can be extracted and further investigated if it correlates with higher outcome scores. On the other hand, polynomial interaction features were used to capture high-order interactions among text data. To this extent, we used a degree of 2 to generate pairwise interaction terms. For example, pairwise features were generated from single words in chat content like 'new prototype' and 'team improve'. This is crucial to capture nonlinear dependencies and cross-features interactions that are not significant in single value features. Consequently, the explicit feature relationships will be captured when we use tree-based or ensemble learning models to predict the outcome score later. In short, TF-IDF generated an addition of 1000 features while polynomial interaction features generated 231 extra features. The codes are shown in Appendix D.

3.2.2. Further Linguistic Feature Engineering for Team Communication Analysis

Word Count

To uncover further insights into how the length of messages affect the outcome score, three new features were derived from the feature '*wordCount*'. Two features, '*team_char_count*' and '*team_word_count*', represent the total number of characters and words typed by a team, respectively (Figure A3). The third feature, *team_avg_word_length*', captures the average words sent in a team, providing a measure of linguistic complexity or brevity in team communication (Figure A3). We proposed that the length of chat content is influential to the student performance, because it is a direct reflection on how students communicate.

NLP

To gain deeper insights into team communication and its relationship with the outcome score, several advanced linguistic features were engineered. We created the feature '*sentiment_consistency*' that measures the standard deviation of sentiment scores in a team (Figure A4). This is because a consistent sentiment is crucial to ensure performance stability across the whole virtual internship program. We hypothesized that if the team has a consistent positive sentiment value in chat content throughout their collaboration, their final report score will be higher and vice versa.

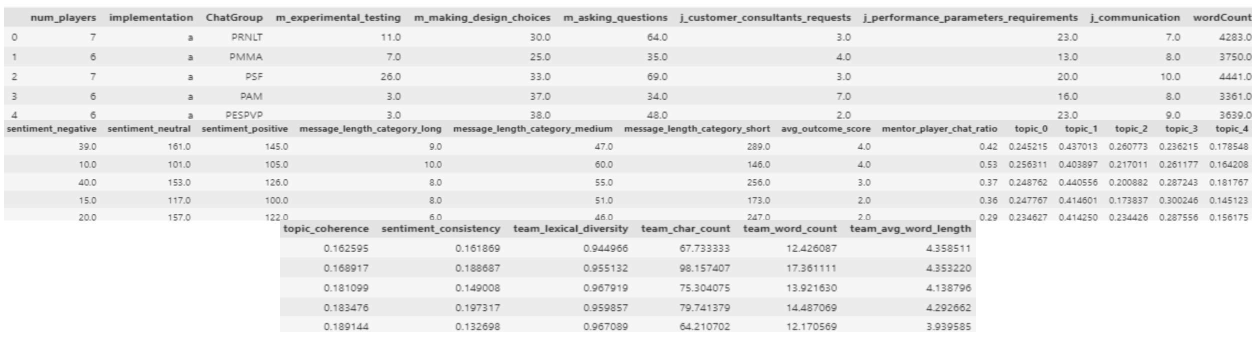
Additionally, we created a feature '*lexical_diversity*' which measures the richness of vocabulary in the chat by calculating the ratio of unique words to the total number of words (Figure A5). We decided to include this feature because we believed that the use of diverse vocabulary could influence the outcome score. A value close to 1 indicates high diversity, meaning a wide range of vocabulary is used, while a value near 0 suggests low diversity and frequent repetition of the same words.

Moreover, the new feature '*topic_coherence*' was made by combining two measures: average entropy and average cosine similarity, as shown in Figure A6. The entropy captures the topic spread for each team member's messages, where higher values indicate less focus, while cosine similarity measures the similarity of discussion topics between team members. The main intention of introducing topic coherence is because we speculated that a high similarity topic discussion and diverse vocabulary used may be a strong indication of effective communication and collaboration hence increasing the team outcome score.

Finally, we employed Latent Dirichlet Allocation (LDA) which is a topic modeling technique used to uncover hidden themes in text by representing each message as a mixture of topics (Jelodar et al., 2019). According to Figure A7, the cleaned chat content is first converted into a Bag-of-Words matrix using CountVectorizer, and then LDA is applied to extract 5 distinct topics. Each message is assigned a probability distribution across these topics, stored in new columns *(topic_0, topic_1, topic_2, topic_3,* t*opic_4*), which are then added back to the original dataset to provide insight into the thematic structure of the conversations. This step is extremely vital because by summarizing chat contents into certain topics enable the model to capture the nature and focus of team dialogues on how they relate to overall success.

3.2.3. Conversion to team statistics & evaluating correlations of new features

The data was recomputed into team-level statistics to include the extensive engineered features by the modified conversion code in Figure B2. Due to the over 1000 features generated from chat messages using TF-IDF and polynomial interactions, we did not take account of these features before filtering and wrapper stages of the features selection. Therefore, the additional meaningful features are *topic_0, topic_1, topic_2, topic_3, topic_4, topic_coherence, sentiment_consistency, team_lexical_diversity, team_char_count, team_word_count, team_avg_word_length*. This results in an overall total of 29 features which are mainly a mix of discrete numerical counts and continuous features. The resulting dataframe, as in Figure 2.6, is used hereon for further analysis with the team-level final report marks, *avg_outcome_score* as our target variable.



	num_players	implementation	ChatGroup	m_experimental_testing	m_making_design_choices	m_asking_questions	j_customer_consultants_requests	j_performance_parameters_requirements	j_communication	wordCount
0	7	a	PRNLT	11.0	30.0	64.0	3.0	23.0	7.0	4283.0
1	6	a	PMMA	7.0	25.0	35.0	4.0	13.0	8.0	3750.0
2	7	a	PSF	26.0	33.0	69.0	3.0	20.0	10.0	4441.0
3	6	a	PAM	3.0	37.0	34.0	7.0	16.0	8.0	3361.0
4	6	a	PESPVP	3.0	38.0	48.0	2.0	23.0	9.0	3639.0

sentiment_negative	sentiment_neutral	sentiment_positive	message_length_category_long	message_length_category_medium	message_length_category_short	avg_outcome_score	mentor_player_chat_ratio	topic_0	topic_1	topic_2	topic_3	topic_4
39.0	161.0	145.0	9.0	47.0	289.0	4.0	0.42	0.245215	0.437013	0.260773	0.236215	0.178548
10.0	101.0	105.0	10.0	60.0	146.0	4.0	0.53	0.256311	0.403897	0.217011	0.261177	0.164208
40.0	153.0	126.0	8.0	55.0	256.0	3.0	0.37	0.248762	0.440556	0.200882	0.287243	0.181767
15.0	117.0	100.0	8.0	51.0	173.0	2.0	0.36	0.247767	0.414601	0.173837	0.300246	0.145123
20.0	157.0	122.0	6.0	46.0	247.0	2.0	0.29	0.234627	0.414250	0.234426	0.287556	0.156175

topic_coherence	sentiment_consistency	team_lexical_diversity	team_char_count	team_word_count	team_avg_word_length
0.162595	0.161869	0.944966	67.733333	12.426087	4.358511
0.168917	0.188687	0.955132	98.157407	17.361111	4.353220
0.181099	0.149008	0.967919	75.304075	13.921630	4.138796
0.183476	0.197317	0.959857	79.741379	14.487069	4.292662
0.189144	0.132698	0.967089	64.210702	12.170569	3.939585

Figure 2.6: *Head of the resulting team-level statistics dataframe*

To ensure the inclusion of these additional features may contribute to subsequent modelling, a correlation heatmap including the new 11 features was generated (similarly filtered to show correlations above absolute value 0.2; refer to Appendix C2). In particular, the new features *topic_2* and *topic_3* show notable correlation to the target, with scores -0.32 and 0.30 respectively. This implies the extensive feature engineering was successful in creating new features that have relevance to the target. Hence, we proceed with modelling on this extended dataframe.

3.3. Features Selection

After all extensive features engineering, we have a total of 1260 features with only 75 rows of samples. This will cause machine learning models to struggle with the curse of dimensionality, spurious correlations and overfitting. Thus, a robust and meaningful features selection framework is required to overcome these challenges. This framework includes the three essential stages for features selection, namely filter, wrapper and embedded methods.

3.3.1. Filter Stage

Filter methods are preprocessing approaches that treat each feature individually by observing only the inherent data characteristic, which is usually an intrinsic statistical measure (Hambali et al., 2020). First, we implemented multicollinearity filter, where two or more highly correlated features (a conservative threshold of 0.95) were filtered out. This method is used to remove redundant information and noise, while avoiding any biased feature importance measures. Hence, features such as 'consultant improve' and 'attach submit' which brings very similar meaning were removed. A total of 254 features were dropped and we used the remaining features to train the baseline model and evaluate its performance. Here, we are using the Logistic Regression model. The training accuracy is 0.89 while the testing accuracy is 0.421, indicating a slight improvement. Next, we used variance thresholding to remove features that do not vary much across the data set (standard deviation below 0.1). The intention of doing so is because such features contain very limited information and are almost constant across all samples, indicating that they do not help in model performance. A total of 83 features were removed, such as 'alex' and 'just posted' which obviously brings no further information but just to increase data noises. The training and testing accuracy were then improved to 0.93 and 0.472 respectively, indicating that the variance filtering is helping to increase model performance.

In addition, we used the classic ReliefF method for the final filter. As stated by Colombelli et al. (2022), the ReliefF algorithm scores each feature based on the difference of nearest neighbors instance pair and rank them. We decided to use ReliefF because it captures feature interactions and local instance-based differences, which are vital in behavioral data like chat content. In this way, each chat feature can be differed to solve multiclass problems, which is our case since we have outcome scores varied from 0 to 8. The effectiveness of this multivariate filter is proved where the testing score is increased to 0.526, with a total of 400 features selected.

3.3.2. Wrapper Stage

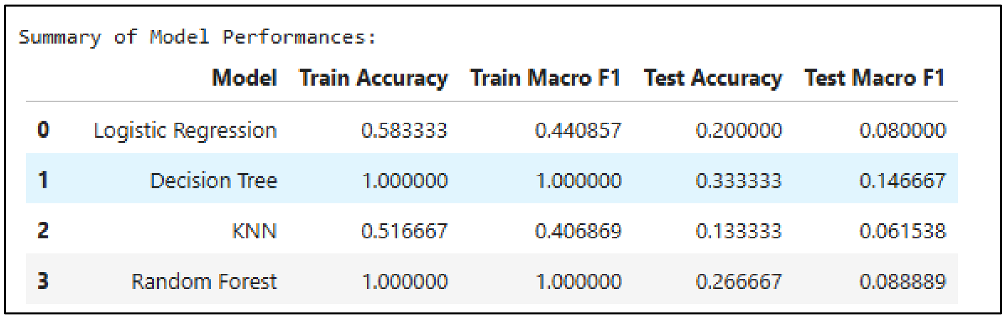
However, one of the drawbacks of filter approach is the disregard of filter and classification algorithm, where potential interactions are not captured since it only access features independently. Therefore, wrapper methods that select features iteratively around a learning algorithm are used. In this context, we used the SelectFromModel (SFM) to train a base model (Random Forest) on the feature space filtered by ReliefF where the SFM will select features with importance higher than the mean importance. Then, we combined SFM with RecursiveFeaturesElimination (RFE) to deliberately eliminate the least important features one by one until only 30 features are left. Implementing wrapper methods is important to assess how feature subsets perform with the model in the loop and ultimately select the most important features only. The model performance increased to a training score of 0.834 and testing score of 0.633, where the difference between training and testing score decreased significantly, indicating that overfitting issue is much improved.

3.3.3. Embedded Stage

Moreover, to further investigate the features importance of selected features, we decided to use the Random Forest's built-in feature importance. This is an embedded method because features selection is performed in the model training process. The importance scores are calculated based on the amount of each features reduces the Gini index across all trees. Finally, a total of 15 features are selected, and being trained again in the AdaBoostClassifier model.

4. Results

4.1. Findings of baseline models with basic feature engineering



```
Summary of Model Performances:
```

	Model	Train Accuracy	Train Macro F1	Test Accuracy	Test Macro F1
0	Logistic Regression	0.583333	0.440857	0.200000	0.080000
1	Decision Tree	1.000000	1.000000	0.333333	0.146667
2	KNN	0.516667	0.406869	0.133333	0.061538
3	Random Forest	1.000000	1.000000	0.266667	0.088889

Figure 3.1: *Summary of model performance for baseline models*

Two metrics were used to evaluate model performance: accuracy and macro F1 Score. Accuracy measures the proportion of correct predictions among all samples, but it may overlook poor performance on minority classes. In contrast, Macro F1 Score gives equal weight to each class, making it more appropriate for evaluating models on imbalanced datasets (Opitz, 2024).

From the results in Figure 3.1, we observed that all models performed poorly on the testing set. Decision Tree and Random Forest achieved perfect accuracy (1.0) on the training data but experienced significant drops in test accuracy, approximately 0.33 and 0.27, respectively. This indicates overfitting, likely due to the small size of the training dataset and insufficient features being trained, causing the models to memorize training data but failed to predict unseen data correctly (Hastie et al., 2001). KNN showed the weakest performance, with a testing accuracy of just around 0.133 and a macro F1 score of 0.062. This poor performance can be attributed to data imbalance, where certain outcome classes are underrepresented, leading to skewed distribution, limiting the performance of the models (Ding et al., 2022).

To improve model performance, we applied AdaBoostClassifier, an ensemble learning method that combines multiple weak classifiers to improve performance. Unlike Random Forest, which builds full decision trees, AdaBoostClassifier typically uses very shallow trees called decision stumps, trees with only one split node and two leaves (Iba & Langley,1992). AdaboostClassifier works by refining previously misclassified samples in each training iteration, gradually improving the model accuracy.

However, AdaBoostClassifier still suffered from overfitting. While it achieved perfect training accuracy (1.0), its test accuracy remained low at 0.267, similar to Random Forest. This indicates the model struggled to generalize to unseen data. One likely reason is insufficient NLP processing in feature engineering, indicating model performance would be expected to improve with extensive feature engineering in next stages.

4.2. Findings of models with extensive feature engineering

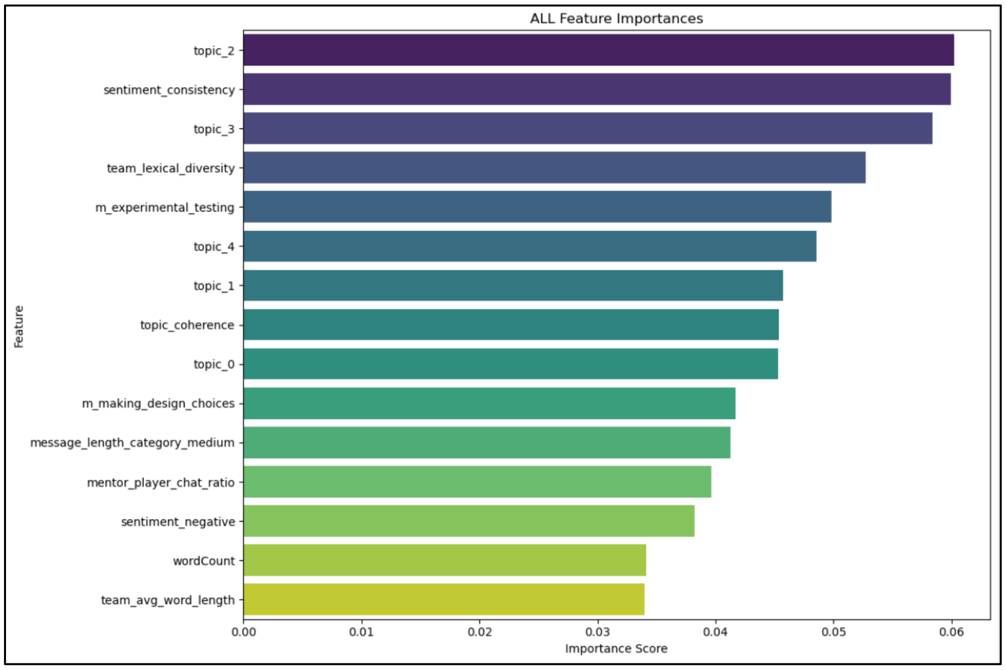
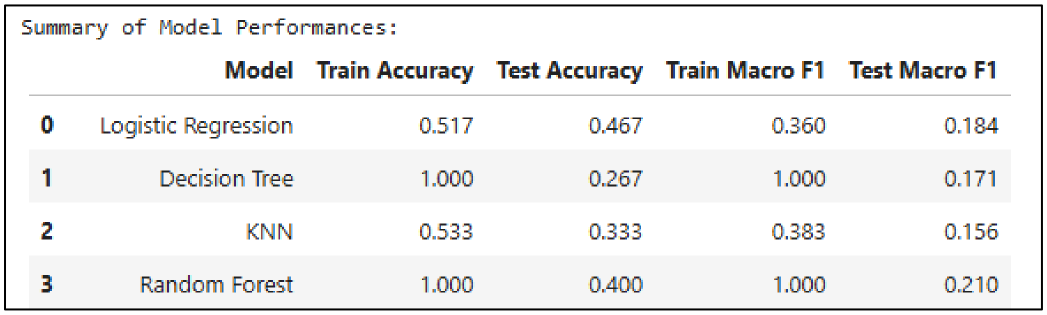


Figure 3.2: *Top 15 features computed from embedded features importance in Random Forest*

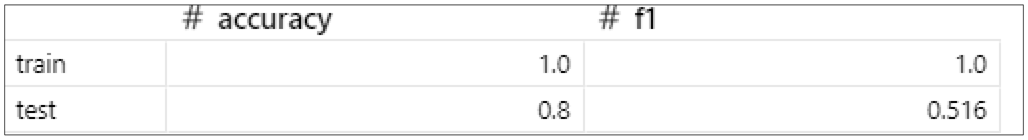
As shown in Figure 3.2, the top 15 most important features are generally the ones that are engineered using the LDA model, namely topic 0 to 4. Sentiment consistency computed using the standard deviation, lexical diversity which measures richness of vocabulary and topic coherence to evaluate semantic coherence is also considered as the most important features. Additionally, original features like *m_experimental_testing, message_length_category_medium, m_making_design_choices* and features from basic features engineering such as *sentiment_negative, team_avg_word_length* and *mentor_player_chat_ratio* are also considered as significant.



Summary of Model Performances:

	Model	Train Accuracy	Test Accuracy	Train Macro F1	Test Macro F1
0	Logistic Regression	0.517	0.467	0.360	0.184
1	Decision Tree	1.000	0.267	1.000	0.171
2	KNN	0.533	0.333	0.383	0.156
3	Random Forest	1.000	0.400	1.000	0.210

Figure 3.3: *Summary of model performance for baseline models after features engineering*



	# accuracy	# f1
train	1.0	1.0
test	0.8	0.516

Figure 3.4: *Summary of model performance for AdaBoostClassifier*

Based on both Figure 3.3 and 3.4, we can observe that the model performance of both baseline models and AdaBoostClassifier model improved significantly. This is because the testing accuracy has increased remarkably, which reduces the difference between training and testing scores. We can then say that the overfitting issue of the model is resolved to some extent, especially for the AdaBoostClassifier model, where the accuracy difference between training and testing score is reduced to only 0.2. Therefore, the best model is the AdaBoostClassifier with a weak learner of RandomForestClassifier, having hyperparameter being tuned to max_depth of 200, n_estimator of 200 and min_samples_split of 10 for RandomForestClassifier, while the n_estimator is 200 and learning_rate of 0.2 is tuned for the AdaBoostClassifier.

As we have tweaked the model performance to optimize testing accuracy by features engineering, we can confidently say that the newly engineered features play a vital role in affecting the team performance. In this context, chat content of players greatly influences the team average outcome score. For instance, sentiment consistency where a consistent positive sentiment between player chat correlates positively to their team score and vice versa. Furthermore, *topic_2* and *topic_3* are also significant features that relate to student performance. This is said so as chat that contains topic like prototype, reliable, best, cost, sound, and submit generally results is higher average outcome score. These topics reflect a high commitment and engagement of players in a team to thrive in the virtual internship. Moreover, the team's lexical diversity and topic coherence also contributes to the team's performance. If team player chatted with a lot of diverse vocabulary and topics, their outcome score will be higher because this indicates that they brainstormed and think critically to design their final report. Apart from that, *m_experimental_testing* is also an important feature where students who talk more about experimenting advanced techniques in their prototype will have a higher outcome score. Not to mention that the effect of mentor also helps in student performance but in a limited way as its features importance is just slightly above than the shared features importance. This is because the report greatly hinges on how students collaborate and work together as a team, where mentoring is just a supplementary support rather than a driver of success. In short, the communication pattern in chat between players is crucial and deterministic on their reports' marks, where discourse features engineered from natural language processing remarkably affecting model performance in predicting team level outcome score.

5. Conclusion

In conclusion, this project successfully fulfilled its aim of investigating the impact of students' communication skills on their final performance in virtual internship teams. Using the Nephrotex internship program as a research context, we collated chat transcripts from multiple teams, transformed individual-level discourse features into team-level data, and combined it with a machine learning model to process team-level aggregated features, and ultimately found that there is a significant correlation between communication features and team success.

In terms of model performance, without feature engineering, most of the classification models have low test accuracy and F1 scores, and there is obvious overfitting. After the introduction of natural language processing features, the model performance is significantly improved while the gap between the training set and the test set is notably reduced. Moreover, we find that sentiment consistency, technical topic distribution, and lexical diversity have the strongest predictive power for final performance. This indicates a significant increase in model prediction accuracy with the introduction of natural language processing features, further confirming the predictive power of these natural language features for team performance.

The results of the integrated model suggested that structured and technically oriented communication styles have a positive impact on final team performance. High-quality interactions between students, such as collaborative discussions around consistent topics such as experimental validation and design decisions, are more conducive for an effective outcome. In contrast, variables related to mentor involvement, such as the ratio of mentor-participant chats, had little predictive value, showing low predictive contributions across multiple models. This result suggested that student-led communication dynamics may be more decisive than mentor intervention in virtual internships and play a greater role in final performance. Overall, both problem statements, to be specific the communication pattern and mentor effect on student performance were being answered throughout the modeling process.

Given the findings of this study that collaborative communication among students significantly impacts team performance, thus future instructional design could place greater emphasis to foster in-depth discussion and meaningful communication around key content. It is also worth revisiting the role of the mentor, which should focus on stimulating students' thinking and interactions rather than directly dominating the discussion process and should facilitate the expression of ideas among students and enhance effective team interactions. Subsequent studies can start from other characteristics and combine different analytical models to provide more systematic theoretical support and empirical evidence for this developmental model.

References

AMD NextGen Engineer (2012). University of Wisconsin-Madison. In *Infusing Real World Experiences into Engineering Education* (p. 30). National Academies Press. https://nap.nationalacademies.org/read/18184/chapter/29

Colombelli, F., Kowalski, T. W., & Recamonde-Mendoza, M. (2022). A hybrid ensemble feature selection design for candidate biomarkers discovery from transcriptome profiles. *Knowledge-Based Systems, 254*, 109655. https://doi.org/10.1016/j.knosys.2022.109655

Ding, H., Chen, L., Dong, L., Fu, Z., & Cui, X. (2022). Imbalanced data classification: A KNN and generative adversarial networks-based hybrid approach for intrusion detection. *Future Generation Computer Systems*, *131*, 240–254. https://doi.org/10.1016/j.future.2022.01.026

Hambali, M. A., Oladele, T. O., & Adewole, K. S. (2020). Microarray cancer feature selection: Review, challenges and research directions. *International Journal of Cognitive Computing in Engineering, 1*, 78-97. https://doi.org/10.1016/j.ijcce.2020.11.001

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd edition.). Springer. https://doi.org/10.1007/978-0-387-84858-7

Iba, W., Langley, P., Sleeman, D., & Edwards, P. (1992). Induction of one-level decision trees. In *Machine Learning Proceedings 1992* (pp. 233–240). Elsevier Inc. https://doi.org/10.1016/B978-1-55860-247-2.50035-8

Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., & Zhao, L. (2019). Latent Dirichlet allocation (LDA) and topic modeling: Models, applications, a survey. *Multimedia Tools and Applications, 78*(11), 15169-15211. https://doi.org/10.1007/s11042-018-6894-4

Khurana, D., Koli, A., Khatter, K., & Singh, S. (2023). Natural language processing: State of the art, current trends and challenges. *Multimedia Tools and Applications*, *82*(3), 3713–3744. https://doi.org/10.1007/s11042-022-13428-4

Opitz, J. (2024). A closer look at classification evaluation metrics and a critical reflection of common evaluation practice. *Transactions of the Association for Computational Linguistics, 12*, 820-836. https://doi.org/10.48550/arxiv.2404.16958

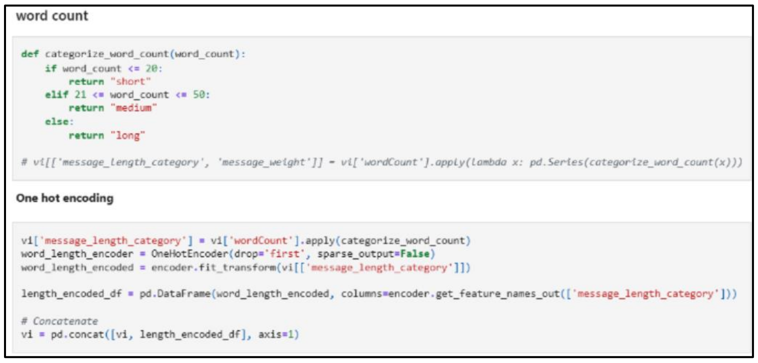
Papageorgiou, S. N. (2022). On correlation coefficients and their interpretation. *Journal of Orthodontics, 49*(3), 359-361. http://doi.org/10.1177/14653125221076142

Qaiser, S., & Ali, R. (2018). Text mining: Use of TF-IDF to examine the relevance of words to documents. *International Journal of Computer Applications, 181*(1). https://doi.org/10.5120/ijca2018917395

Tuma, T. T., Adams, J. D., Hultquist, B. C., & Dolan, E. L. (2021). The dark side of development: A systems characterization of the negative mentoring experiences of doctoral students. *CBE Life Sciences Education, 20*(2), ar16. https://doi.org/10.1187/cbe.20-10-0231

Virtual Internships. (n.d.). *Nephrotex.* https://www.virtualinterns.org/nephrotex/

Appendix A
Feature Engineering using NLP

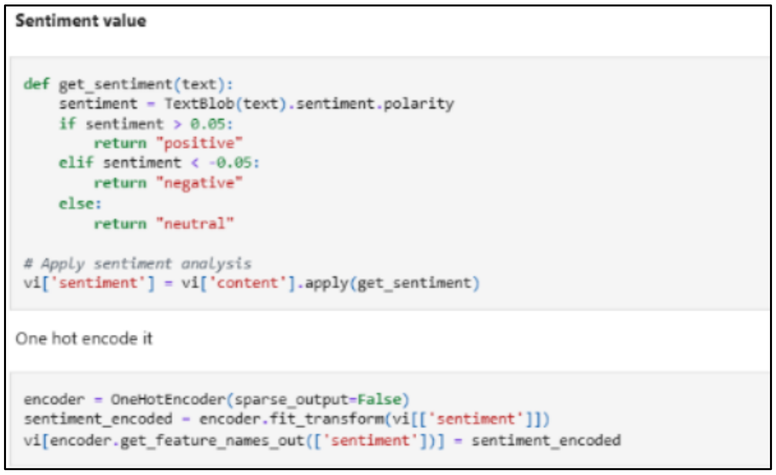

word count

```
def categorize_word_count(word_count):
    if word_count <= 20:
        return "short"
    elif 21 <= word_count <= 50:
        return "medium"
    else:
        return "long"

# vi[['message_length_category', 'message_weight']] = vi['wordCount'].apply(lambda x: pd.Series(categorize_word_count(x)))
```

One hot encoding

```
vi['message_length_category'] = vi['wordCount'].apply(categorize_word_count)
word_length_encoder = OneHotEncoder(drop='first', sparse_output=False)
word_length_encoded = encoder.fit_transform(vi[['message_length_category']])

length_encoded_df = pd.DataFrame(word_length_encoded, columns=encoder.get_feature_names_out(['message_length_category']))

# Concatenate
vi = pd.concat([vi, length_encoded_df], axis=1)
```

Figure A1: *Code for categorizing 'word_count'*

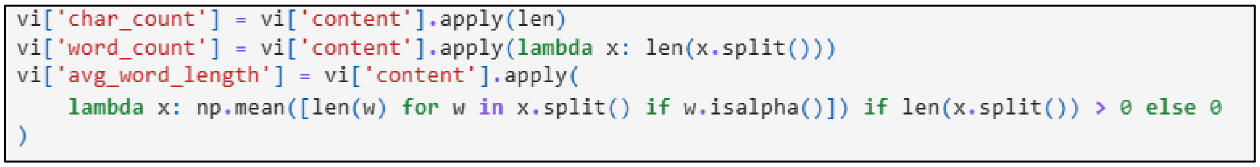

Sentiment value

```
def get_sentiment(text):
    sentiment = TextBlob(text).sentiment.polarity
    if sentiment > 0.05:
        return "positive"
    elif sentiment < -0.05:
        return "negative"
    else:
        return "neutral"

# Apply sentiment analysis
vi['sentiment'] = vi['content'].apply(get_sentiment)
```

One hot encode it

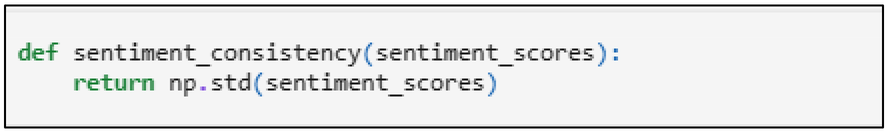
```
encoder = OneHotEncoder(sparse_output=False)
sentiment_encoded = encoder.fit_transform(vi[['sentiment']])
vi[encoder.get_feature_names_out(['sentiment'])] = sentiment_encoded
```

Figure A2: *Code for creating the feature 'sentiment_positive', sentiment_negative', sentiment_neutral'*



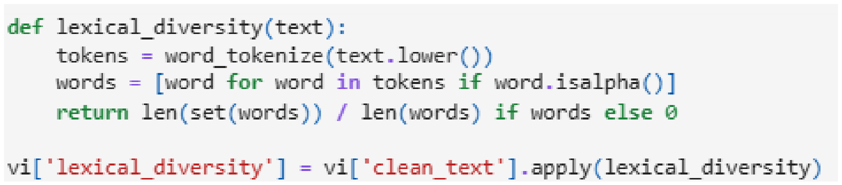
```
vi['char_count'] = vi['content'].apply(len)
vi['word_count'] = vi['content'].apply(lambda x: len(x.split()))
vi['avg_word_length'] = vi['content'].apply(
    lambda x: np.mean([len(w) for w in x.split() if w.isalpha()]) if len(x.split()) > 0 else 0
)
```

Figure A3: *Code for creating the feature 'team_char_count' and 'avg_word_length'*



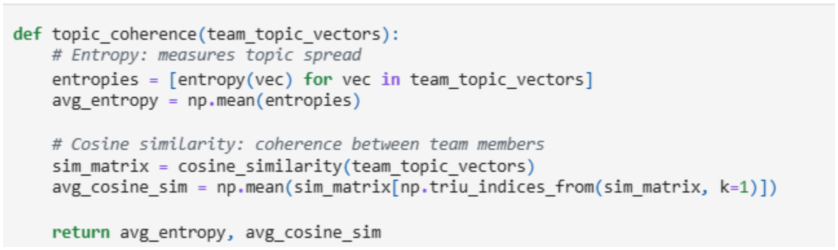
```python
def sentiment_consistency(sentiment_scores):
    return np.std(sentiment_scores)
```

Figure A4: *Code for creating the feature 'sentiment_consistency'*



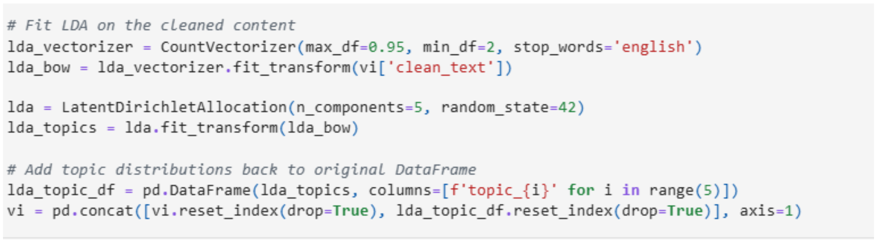
```python
def lexical_diversity(text):
    tokens = word_tokenize(text.lower())
    words = [word for word in tokens if word.isalpha()]
    return len(set(words)) / len(words) if words else 0

vi['lexical_diversity'] = vi['clean_text'].apply(lexical_diversity)
```

Figure A5: *Code for creating the feature 'lexical_diversity'*



```python
def topic_coherence(team_topic_vectors):
    # Entropy: measures topic spread
    entropies = [entropy(vec) for vec in team_topic_vectors]
    avg_entropy = np.mean(entropies)

    # Cosine similarity: coherence between team members
    sim_matrix = cosine_similarity(team_topic_vectors)
    avg_cosine_sim = np.mean(sim_matrix[np.triu_indices_from(sim_matrix, k=1)])

    return avg_entropy, avg_cosine_sim
```

Figure A6: *Code for creating the feature 'topic_coherence'*

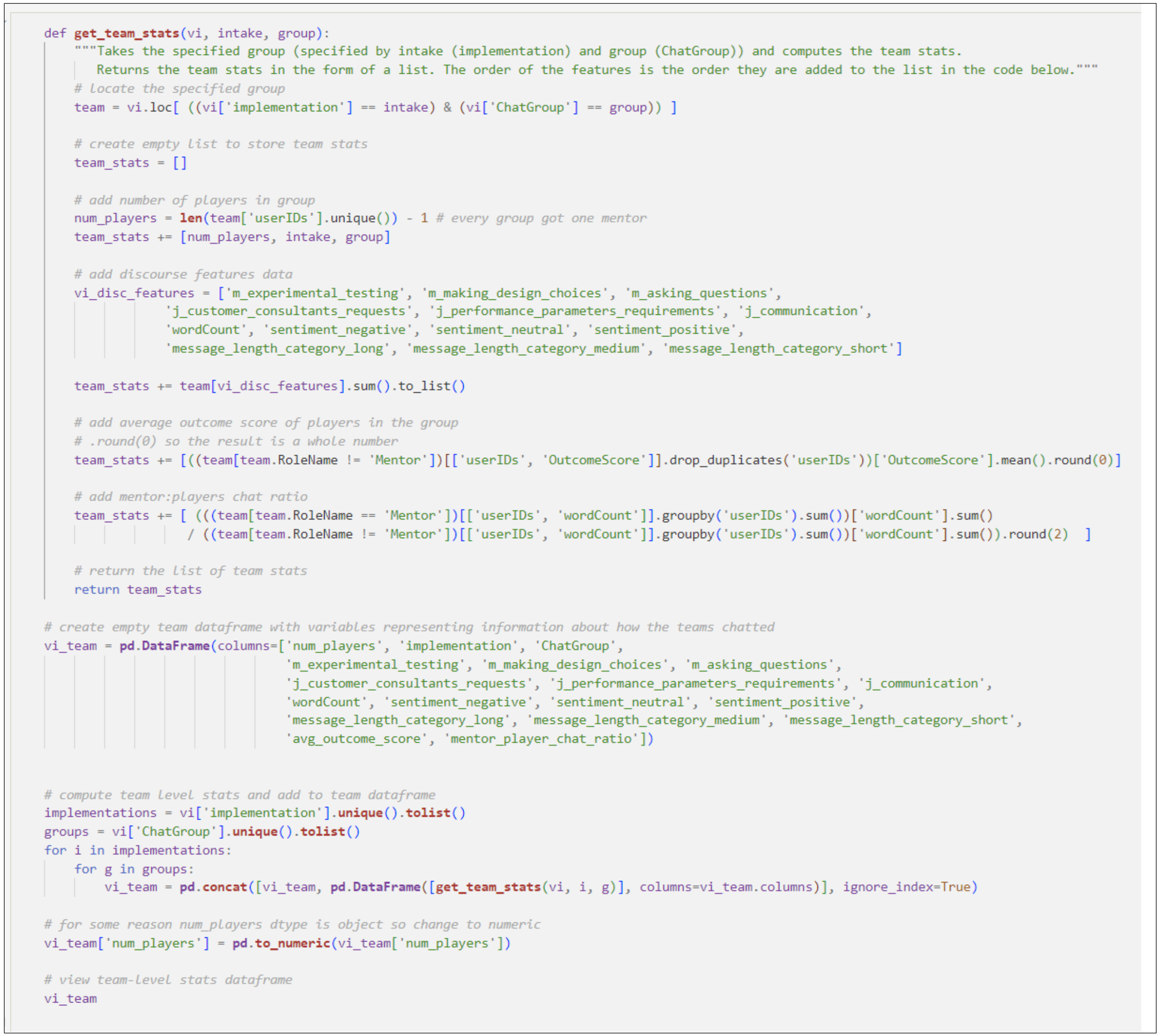


```python
# Fit LDA on the cleaned content
lda_vectorizer = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')
lda_bow = lda_vectorizer.fit_transform(vi['clean_text'])

lda = LatentDirichletAllocation(n_components=5, random_state=42)
lda_topics = lda.fit_transform(lda_bow)

# Add topic distributions back to original DataFrame
lda_topic_df = pd.DataFrame(lda_topics, columns=[f'topic_{i}' for i in range(5)])
vi = pd.concat([vi.reset_index(drop=True), lda_topic_df.reset_index(drop=True)], axis=1)
```

Figure A7: *Code for employing LDA to create 5 new features*

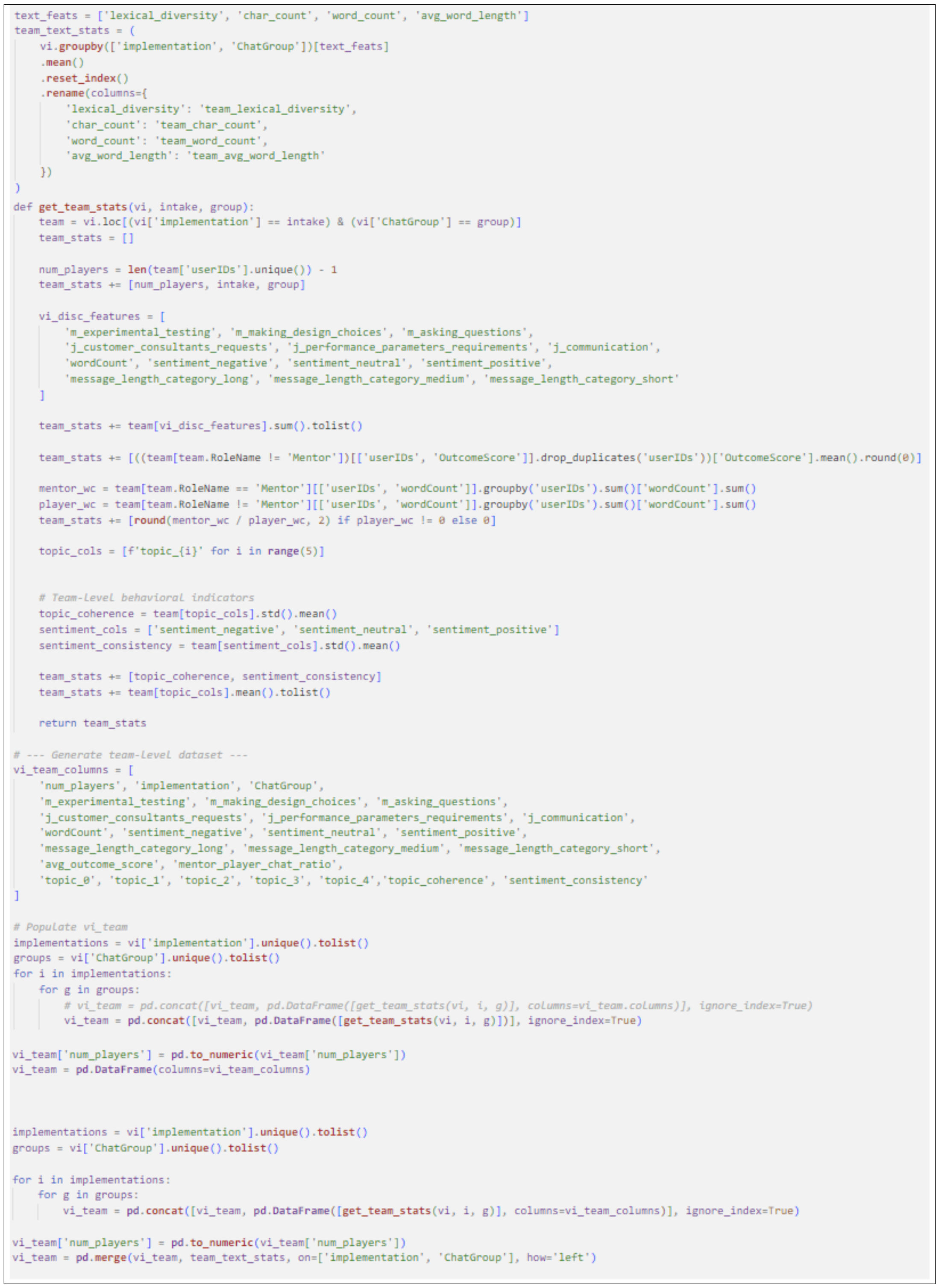
Appendix B

Team-level Statistics



```python
def get_team_stats(vi, intake, group):
    """Takes the specified group (specified by intake (implementation) and group (ChatGroup)) and computes the team stats.
       Returns the team stats in the form of a list. The order of the features is the order they are added to the list in the code below."""
    # locate the specified group
    team = vi.loc[ ((vi['implementation'] == intake) & (vi['ChatGroup'] == group)) ]

    # create empty list to store team stats
    team_stats = []

    # add number of players in group
    num_players = len(team['userIDs'].unique()) - 1 # every group got one mentor
    team_stats += [num_players, intake, group]

    # add discourse features data
    vi_disc_features = ['m_experimental_testing', 'm_making_design_choices', 'm_asking_questions',
                        'j_customer_consultants_requests', 'j_performance_parameters_requirements', 'j_communication',
                        'wordCount', 'sentiment_negative', 'sentiment_neutral', 'sentiment_positive',
                        'message_length_category_long', 'message_length_category_medium', 'message_length_category_short']

    team_stats += team[vi_disc_features].sum().to_list()

    # add average outcome score of players in the group
    # .round(0) so the result is a whole number
    team_stats += [((team[team.RoleName != 'Mentor'])[['userIDs', 'OutcomeScore']].drop_duplicates('userIDs'))['OutcomeScore'].mean().round(0)]

    # add mentor:players chat ratio
    team_stats += [ (((team[team.RoleName == 'Mentor'])[['userIDs', 'wordCount']].groupby('userIDs').sum())['wordCount'].sum()
                   / ((team[team.RoleName != 'Mentor'])[['userIDs', 'wordCount']].groupby('userIDs').sum())['wordCount'].sum()).round(2)  ]

    # return the list of team stats
    return team_stats

# create empty team dataframe with variables representing information about how the teams chatted
vi_team = pd.DataFrame(columns=['num_players', 'implementation', 'ChatGroup',
                                'm_experimental_testing', 'm_making_design_choices', 'm_asking_questions',
                                'j_customer_consultants_requests', 'j_performance_parameters_requirements', 'j_communication',
                                'wordCount', 'sentiment_negative', 'sentiment_neutral', 'sentiment_positive',
                                'message_length_category_long', 'message_length_category_medium', 'message_length_category_short',
                                'avg_outcome_score', 'mentor_player_chat_ratio'])


# compute team level stats and add to team dataframe
implementations = vi['implementation'].unique().tolist()
groups = vi['ChatGroup'].unique().tolist()
for i in implementations:
    for g in groups:
        vi_team = pd.concat([vi_team, pd.DataFrame([get_team_stats(vi, i, g)], columns=vi_team.columns)], ignore_index=True)

# for some reason num_players dtype is object so change to numeric
vi_team['num_players'] = pd.to_numeric(vi_team['num_players'])

# view team-level stats dataframe
vi_team
```

Figure B1: *Code for creating the team-level statistics dataframe (basic feature engineering version, containing 18 features)*



```python
text_feats = ['lexical_diversity', 'char_count', 'word_count', 'avg_word_length']
team_text_stats = (
    vi.groupby(['implementation', 'ChatGroup'])[text_feats]
    .mean()
    .reset_index()
    .rename(columns={
        'lexical_diversity': 'team_lexical_diversity',
        'char_count': 'team_char_count',
        'word_count': 'team_word_count',
        'avg_word_length': 'team_avg_word_length'
    })
)
def get_team_stats(vi, intake, group):
    team = vi.loc[(vi['implementation'] == intake) & (vi['ChatGroup'] == group)]
    team_stats = []

    num_players = len(team['userIDs'].unique()) - 1
    team_stats += [num_players, intake, group]

    vi_disc_features = [
        'm_experimental_testing', 'm_making_design_choices', 'm_asking_questions',
        'j_customer_consultants_requests', 'j_performance_parameters_requirements', 'j_communication',
        'wordCount', 'sentiment_negative', 'sentiment_neutral', 'sentiment_positive',
        'message_length_category_long', 'message_length_category_medium', 'message_length_category_short'
    ]

    team_stats += team[vi_disc_features].sum().tolist()

    team_stats += [((team[team.RoleName != 'Mentor'])[['userIDs', 'OutcomeScore']].drop_duplicates('userIDs'))['OutcomeScore'].mean().round(0)]

    mentor_wc = team[team.RoleName == 'Mentor'][['userIDs', 'wordCount']].groupby('userIDs').sum()['wordCount'].sum()
    player_wc = team[team.RoleName != 'Mentor'][['userIDs', 'wordCount']].groupby('userIDs').sum()['wordCount'].sum()
    team_stats += [round(mentor_wc / player_wc, 2) if player_wc != 0 else 0]

    topic_cols = [f'topic_{i}' for i in range(5)]


    # Team-level behavioral indicators
    topic_coherence = team[topic_cols].std().mean()
    sentiment_cols = ['sentiment_negative', 'sentiment_neutral', 'sentiment_positive']
    sentiment_consistency = team[sentiment_cols].std().mean()

    team_stats += [topic_coherence, sentiment_consistency]
    team_stats += team[topic_cols].mean().tolist()

    return team_stats

# --- Generate team-level dataset ---
vi_team_columns = [
    'num_players', 'implementation', 'ChatGroup',
    'm_experimental_testing', 'm_making_design_choices', 'm_asking_questions',
    'j_customer_consultants_requests', 'j_performance_parameters_requirements', 'j_communication',
    'wordCount', 'sentiment_negative', 'sentiment_neutral', 'sentiment_positive',
    'message_length_category_long', 'message_length_category_medium', 'message_length_category_short',
    'avg_outcome_score', 'mentor_player_chat_ratio',
    'topic_0', 'topic_1', 'topic_2', 'topic_3', 'topic_4','topic_coherence', 'sentiment_consistency'
]

# Populate vi_team
implementations = vi['implementation'].unique().tolist()
groups = vi['ChatGroup'].unique().tolist()
for i in implementations:
    for g in groups:
        # vi_team = pd.concat([vi_team, pd.DataFrame([get_team_stats(vi, i, g)], columns=vi_team.columns)], ignore_index=True)
        vi_team = pd.concat([vi_team, pd.DataFrame([get_team_stats(vi, i, g)])], ignore_index=True)

vi_team['num_players'] = pd.to_numeric(vi_team['num_players'])
vi_team = pd.DataFrame(columns=vi_team_columns)



implementations = vi['implementation'].unique().tolist()
groups = vi['ChatGroup'].unique().tolist()

for i in implementations:
    for g in groups:
        vi_team = pd.concat([vi_team, pd.DataFrame([get_team_stats(vi, i, g)], columns=vi_team_columns)], ignore_index=True)

vi_team['num_players'] = pd.to_numeric(vi_team['num_players'])
vi_team = pd.merge(vi_team, team_text_stats, on=['implementation', 'ChatGroup'], how='left')
```

Figure B2: *Code for creating the team-level statistics dataframe (extensive feature engineering version, containing 29 features)*

Appendix C

Correlation Heatmaps

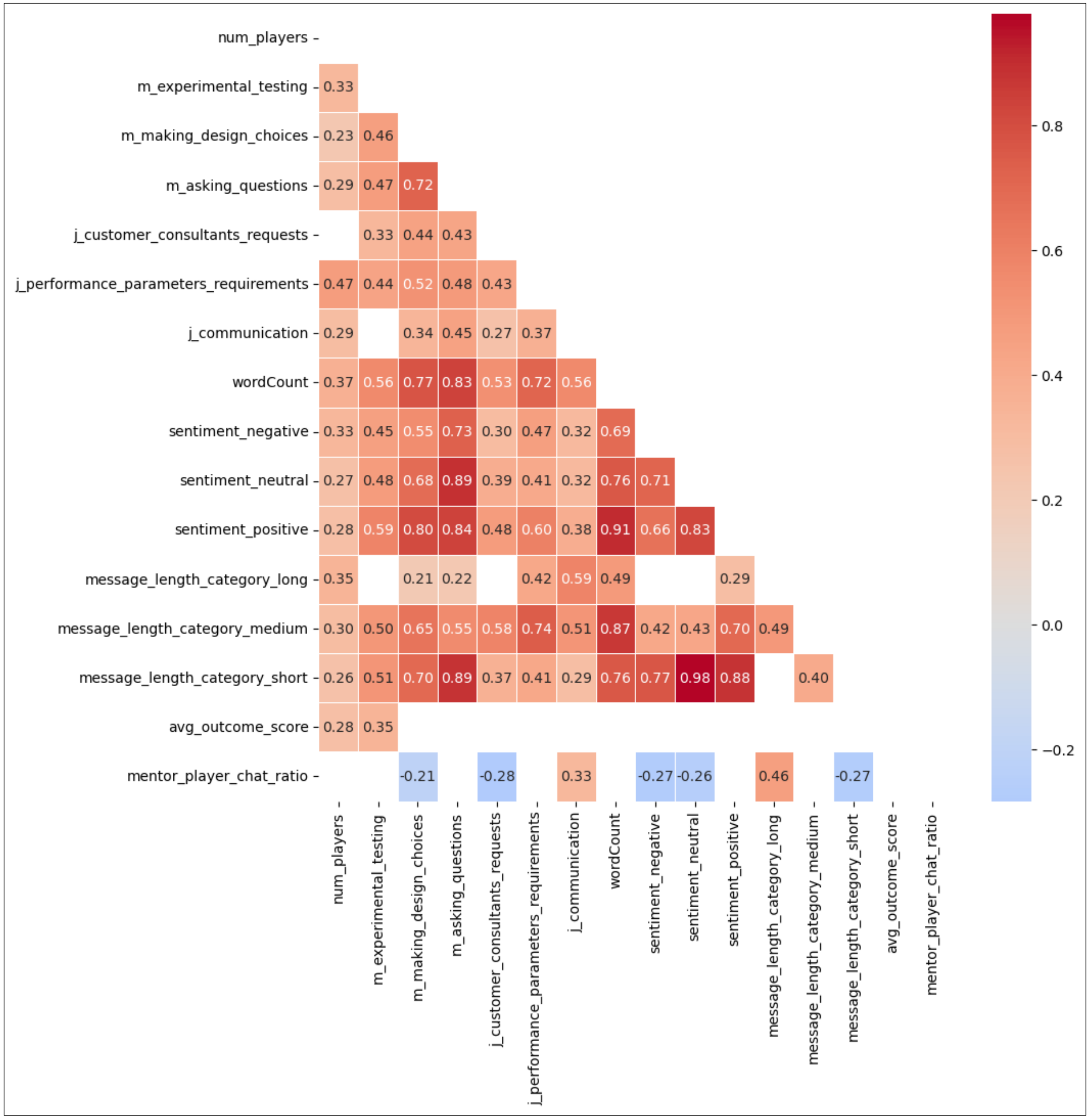


Figure C1: *Correlation heatmap between features of our team-level data (basic version)*

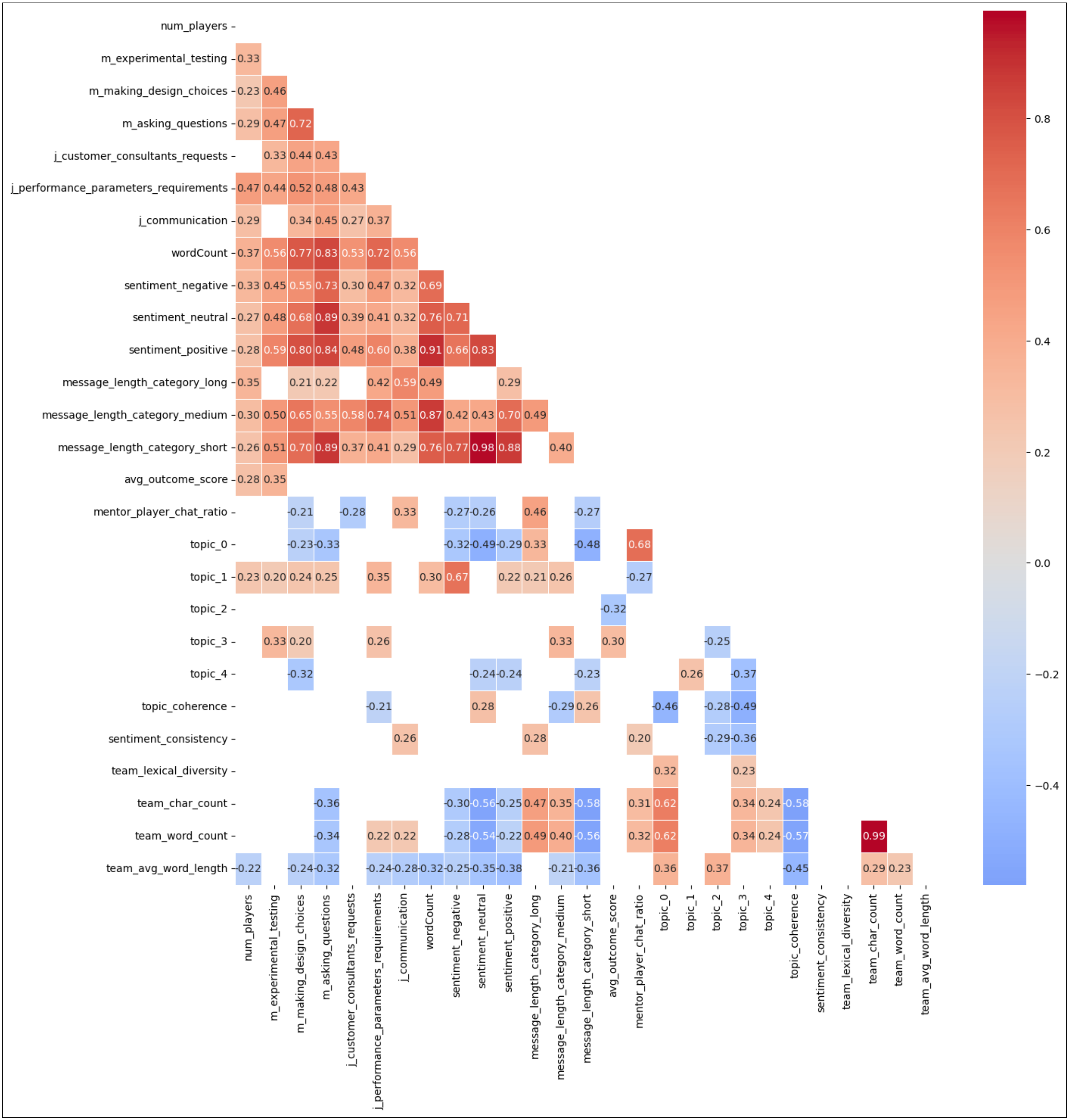
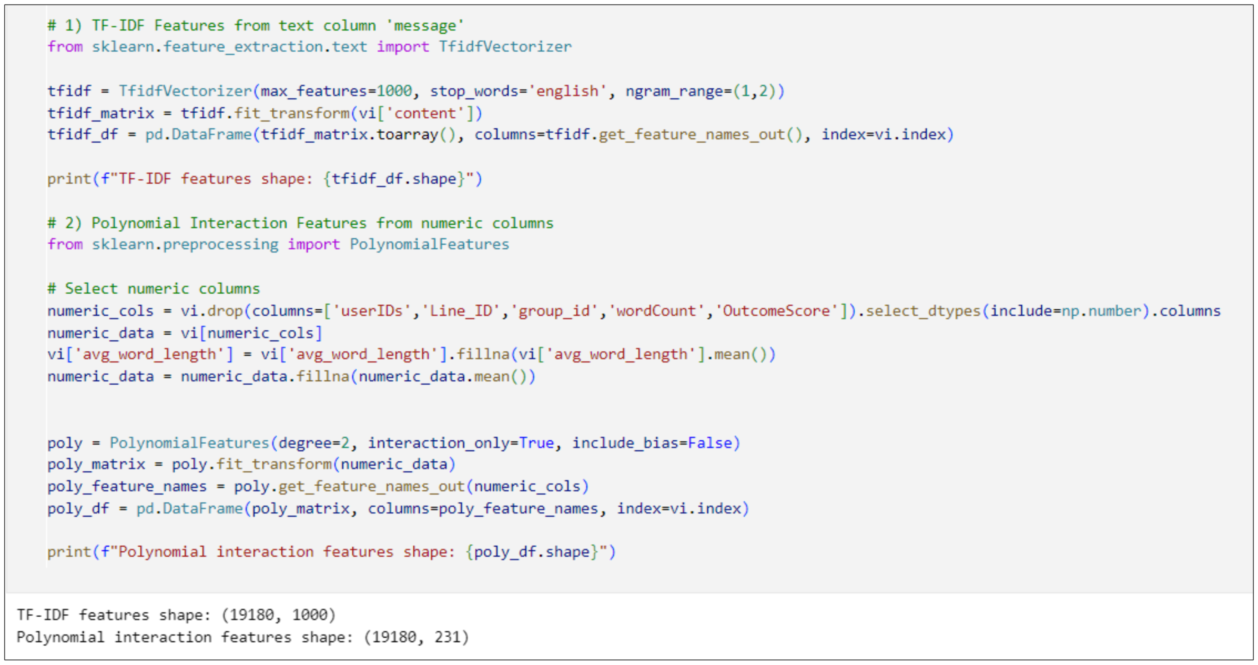


Figure C2: *Correlation heatmap between features of our team-level data (extended version)*

Appendix D

Features Engineering using TF-IDF and Polynomial Interaction Features



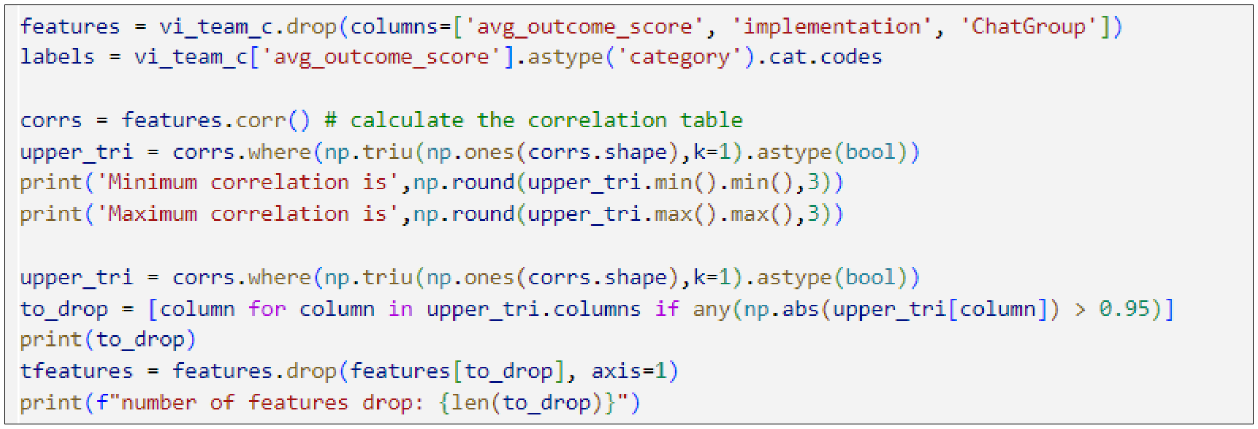
```
# 1) TF-IDF Features from text column 'message'
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=1000, stop_words='english', ngram_range=(1,2))
tfidf_matrix = tfidf.fit_transform(vi['content'])
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf.get_feature_names_out(), index=vi.index)

print(f"TF-IDF features shape: {tfidf_df.shape}")

# 2) Polynomial Interaction Features from numeric columns
from sklearn.preprocessing import PolynomialFeatures

# Select numeric columns
numeric_cols = vi.drop(columns=['userIDs','Line_ID','group_id','wordCount','OutcomeScore']).select_dtypes(include=np.number).columns
numeric_data = vi[numeric_cols]
vi['avg_word_length'] = vi['avg_word_length'].fillna(vi['avg_word_length'].mean())
numeric_data = numeric_data.fillna(numeric_data.mean())


poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
poly_matrix = poly.fit_transform(numeric_data)
poly_feature_names = poly.get_feature_names_out(numeric_cols)
poly_df = pd.DataFrame(poly_matrix, columns=poly_feature_names, index=vi.index)

print(f"Polynomial interaction features shape: {poly_df.shape}")
```

```
TF-IDF features shape: (19180, 1000)
Polynomial interaction features shape: (19180, 231)
```

Figure D1: *Code for generating more features using TF-IDF and Polynomial Interaction Features*

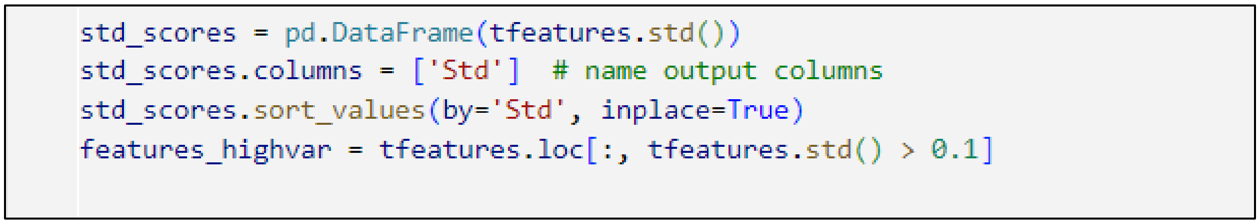
Appendix E

Filter and Wrapper methods



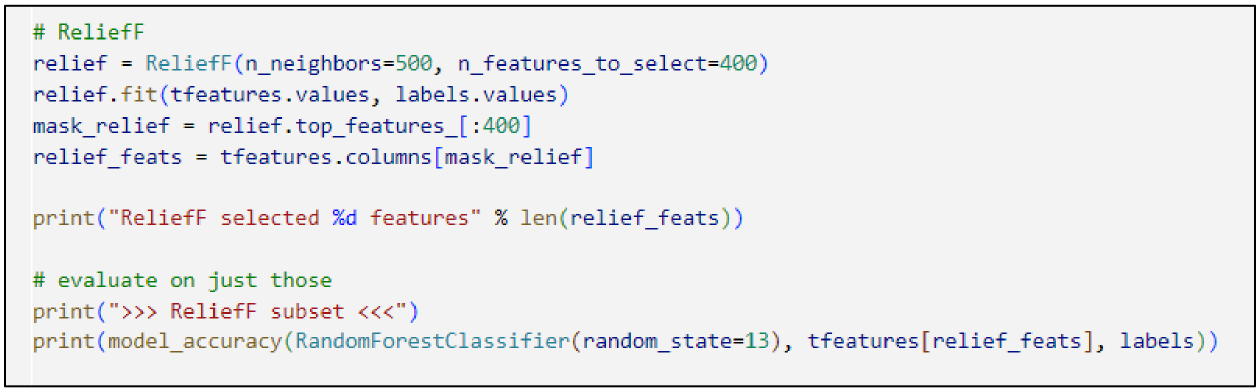
```python
features = vi_team_c.drop(columns=['avg_outcome_score', 'implementation', 'ChatGroup'])
labels = vi_team_c['avg_outcome_score'].astype('category').cat.codes

corrs = features.corr() # calculate the correlation table
upper_tri = corrs.where(np.triu(np.ones(corrs.shape),k=1).astype(bool))
print('Minimum correlation is',np.round(upper_tri.min().min(),3))
print('Maximum correlation is',np.round(upper_tri.max().max(),3))

upper_tri = corrs.where(np.triu(np.ones(corrs.shape),k=1).astype(bool))
to_drop = [column for column in upper_tri.columns if any(np.abs(upper_tri[column]) > 0.95)]
print(to_drop)
tfeatures = features.drop(features[to_drop], axis=1)
print(f"number of features drop: {len(to_drop)}")
```

Figure E1: *Code for multicollinearity*



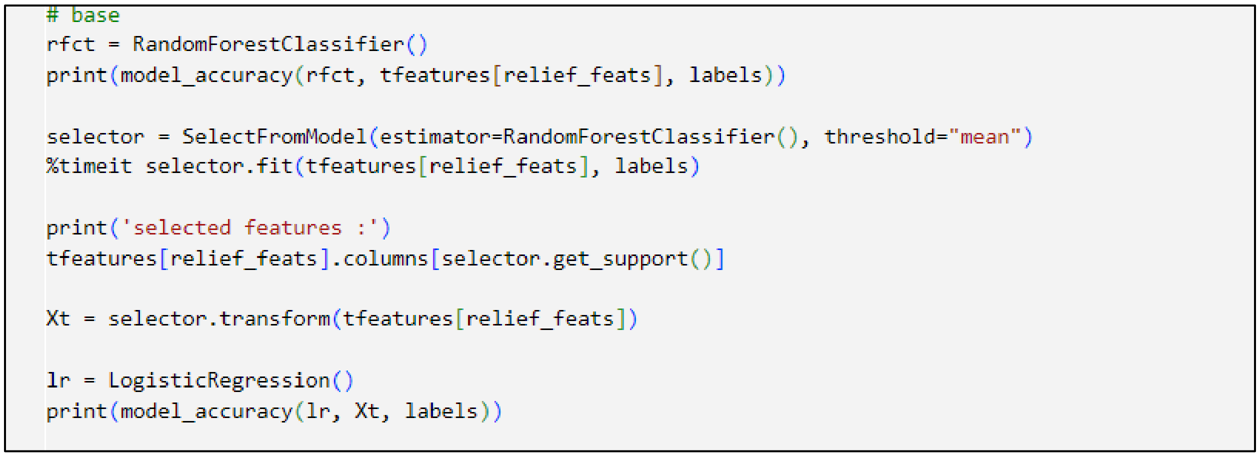
```python
std_scores = pd.DataFrame(tfeatures.std())
std_scores.columns = ['Std']  # name output columns
std_scores.sort_values(by='Std', inplace=True)
features_highvar = tfeatures.loc[:, tfeatures.std() > 0.1]
```

Figure E2: *Code for variance threshold*



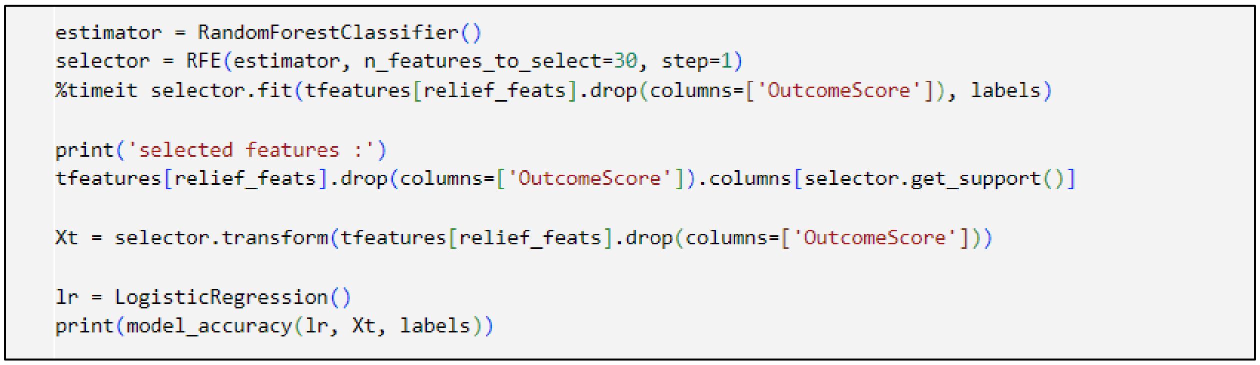
```python
# ReliefF
relief = ReliefF(n_neighbors=500, n_features_to_select=400)
relief.fit(tfeatures.values, labels.values)
mask_relief = relief.top_features_[:400]
relief_feats = tfeatures.columns[mask_relief]

print("ReliefF selected %d features" % len(relief_feats))

# evaluate on just those
print(">>> ReliefF subset <<<")
print(model_accuracy(RandomForestClassifier(random_state=13), tfeatures[relief_feats], labels))
```

Figure E3: *Code for ReliefF*



```
# base
rfct = RandomForestClassifier()
print(model_accuracy(rfct, tfeatures[relief_feats], labels))

selector = SelectFromModel(estimator=RandomForestClassifier(), threshold="mean")
%timeit selector.fit(tfeatures[relief_feats], labels)

print('selected features :')
tfeatures[relief_feats].columns[selector.get_support()]

Xt = selector.transform(tfeatures[relief_feats])

lr = LogisticRegression()
print(model_accuracy(lr, Xt, labels))
```

Figure E4: *Code for SelectFromModel*



```
estimator = RandomForestClassifier()
selector = RFE(estimator, n_features_to_select=30, step=1)
%timeit selector.fit(tfeatures[relief_feats].drop(columns=['OutcomeScore']), labels)

print('selected features :')
tfeatures[relief_feats].drop(columns=['OutcomeScore']).columns[selector.get_support()]

Xt = selector.transform(tfeatures[relief_feats].drop(columns=['OutcomeScore']))

lr = LogisticRegression()
print(model_accuracy(lr, Xt, labels))
```

Figure E5: *Code for RecursiveFeaturesElimination*