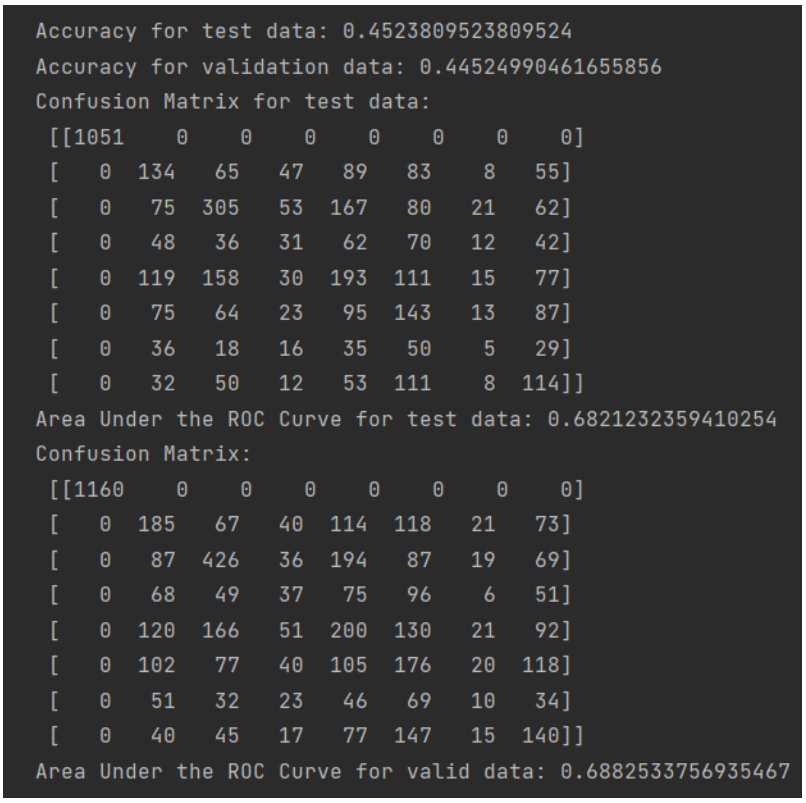
Q775 Assignment 2

Name: Ahana Malhotra, Student ID:400380436

Date of Submission: 10 April 2023

Question 1

1. Data is split into 30% of validation data, 20% of test data and 50% training data.
2. All categorical variables are converted into binary variable using one-hot encoding.
3. All numerical variables are standardized with mean 0 and standard deviation 1.
4. Since this data contains categorical and numerical variables, therefore decision tree is used for prediction of disease.
5. Finally, prediction is made for validation and test data.
6. For test data accuracy is 0.4523809523809524.
7. For validation data accuracy is 0.44524990461655856.
8. Confusion Matrix and Area under the ROC curve is given in the figure below.

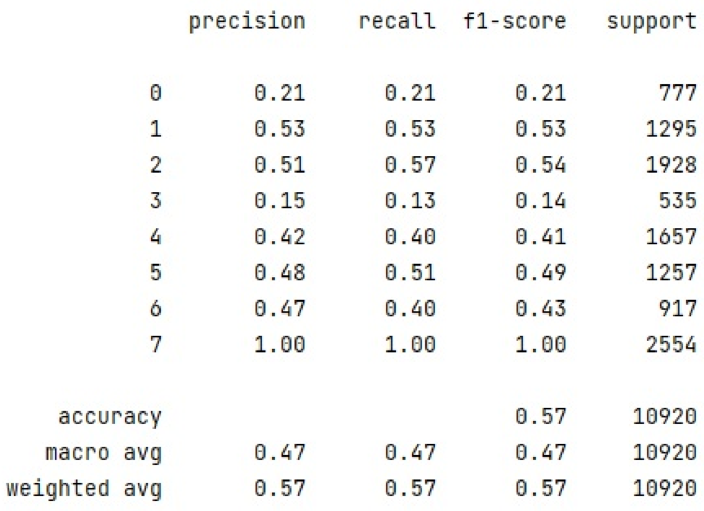


```
Accuracy for test data: 0.4523809523809524
Accuracy for validation data: 0.44524990461655856
Confusion Matrix for test data:
 [[1051    0    0    0    0    0    0    0]
 [   0  134   65   47   89   83    8   55]
 [   0   75  305   53  167   80   21   62]
 [   0   48   36   31   62   70   12   42]
 [   0  119  158   30  193  111   15   77]
 [   0   75   64   23   95  143   13   87]
 [   0   36   18   16   35   50    5   29]
 [   0   32   50   12   53  111    8  114]]
Area Under the ROC Curve for test data: 0.6821232359410254
Confusion Matrix:
 [[1160    0    0    0    0    0    0    0]
 [   0  185   67   40  114  118   21   73]
 [   0   87  426   36  194   87   19   69]
 [   0   68   49   37   75   96    6   51]
 [   0  120  166   51  200  130   21   92]
 [   0  102   77   40  105  176   20  118]
 [   0   51   32   23   46   69   10   34]
 [   0   40   45   17   77  147   15  140]]
Area Under the ROC Curve for valid data: 0.6882533756935467
```

Figure 1: Confusion Matrix and Area under the ROC Curve for cdc data

Then, we used Bagging (classification tree) to train and test data. The results for Question 1 Bagging is based on the Python file ‘A2QBagging.ipynb’ in attachment file. Figure 2, Figure 3, Figure 4 shows the scores, confusion matrix and ROC respectively. The prediction column are converted to number 1 to 8, where these numbers determine the 8 classes of diseases:

- DIPHTHERIA: class 0
- MUMPS: class 1
- POLIO: class 2
- MEASLES: class 3
- RUBELLA: class 4
- PERTUSSIS: class 5
- HEPATITIS A: class 6
- SMALLPOX: class 7

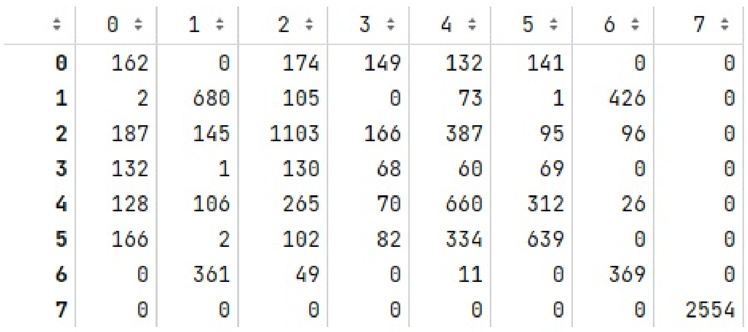


```
              precision    recall  f1-score   support

           0       0.21      0.21      0.21       777
           1       0.53      0.53      0.53      1295
           2       0.51      0.57      0.54      1928
           3       0.15      0.13      0.14       535
           4       0.42      0.40      0.41      1657
           5       0.48      0.51      0.49      1257
           6       0.47      0.40      0.43       917
           7       1.00      1.00      1.00      2554

    accuracy                           0.57     10920
   macro avg       0.47      0.47      0.47     10920
weighted avg       0.57      0.57      0.57     10920
```

Figure 2: Bagging classification report



÷	0 ÷	1 ÷	2 ÷	3 ÷	4 ÷	5 ÷	6 ÷	7 ÷
0	162	0	174	149	132	141	0	0
1	2	680	105	0	73	1	426	0
2	187	145	1103	166	387	95	96	0
3	132	1	130	68	60	69	0	0
4	128	106	265	70	660	312	26	0
5	166	2	102	82	334	639	0	0
6	0	361	49	0	11	0	369	0
7	0	0	0	0	0	0	0	2554

Figure 3: Confusion matrix for Bagging

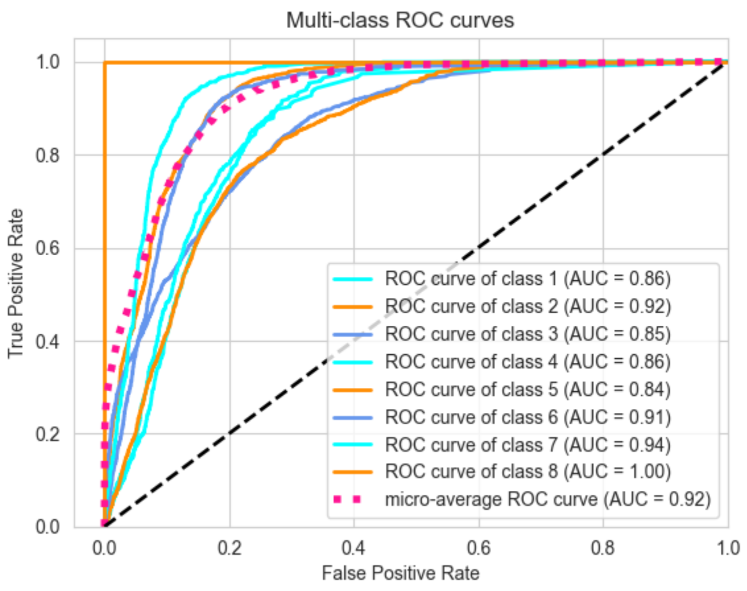


Figure 4: ROC curve for Bagging

The report shows that the model performed relatively well for classes 1, 2, 4, 5, and 7, with f1-scores ranging from 0.41 to 1.0. However, the model performed poorly for classes 0, 3, and 6, with f1-scores ranging from 0.14 to 0.43. This shows that the model may have difficulty accurately predicting samples from these classes. The overall accuracy of the model was 0.57 or 57%, which is not very high. Both macro and weighted averages for precision, recall, and f1-score were around 0.47 and 0.57 respectively, which indicates that the model's overall performance was moderate. The AUC scores for classes 1-7 range from 0.84 to 0.94, showing that the model performs almost well at distinguishing between positive and negative samples in each class. The AUC score for class 8 is 1, which indicates that the model can perfectly distinguish between positive and negative samples in this class. The micro average AUC score is 0.92, which is higher than the AUC scores for individual classes, indicating that the model performs well on average across all classes.

Question 2

The results for Question 2 is based on the Python file 'A2Q2.ipynb' in attachment file.
The Ecoli dataset has eight attributes. The first seven attributes represent the amino acid composition of membrane proteins and the last attribute represents the localization site of the protein. We remove the last attribute since it represents the localization site of the protein. In other words, it indicates the cellular component in which the protein is located. Since we are interested in clustering the membrane proteins based on their amino acid composition.But, The localization site does not provide any information about the amino acid composition of the protein. Therefore, including this attribute in the clustering algorithm could actually reduce the accuracy of the clustering by introducing irrelevant information. Also, the aim of clustering is often to uncover underlying patterns or structures in the data. If we already have the localization site information, it is likely that the clustering result will simply replicate this information rather than revealing any new insights. We also normalize the data using min-max normalization. The elbow method involves plotting the sum of squared distances (SSE) between each data point and its assigned centroid for different values of K and identifying the point where the decrease in SSE starts to level off. We use elbow method, and as Figure 5 shows the k is equal to 2 and 4. We consider $k = 4$ for following analysis.

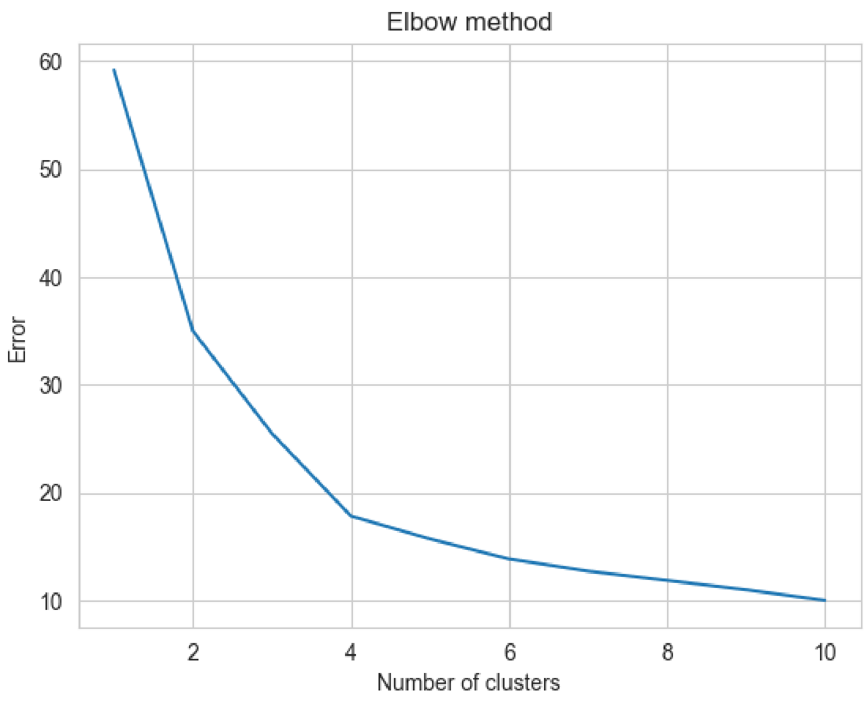


Figure 5: Elbow method

we use the PCA dimension reduction method to reduce the attributes for facilitating the plotting phase. Then, we apply the K-means clustering algorithm to the data using the optimal number of clusters. Finally, we plot the clusters and their centroids to visualize the result. Figure 6 shows the four cluster and their centroids.

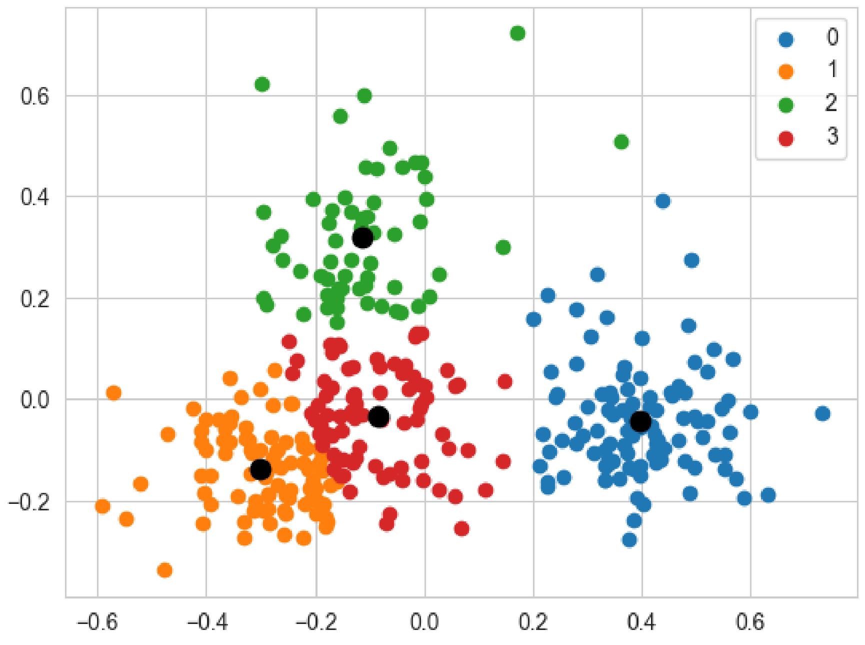
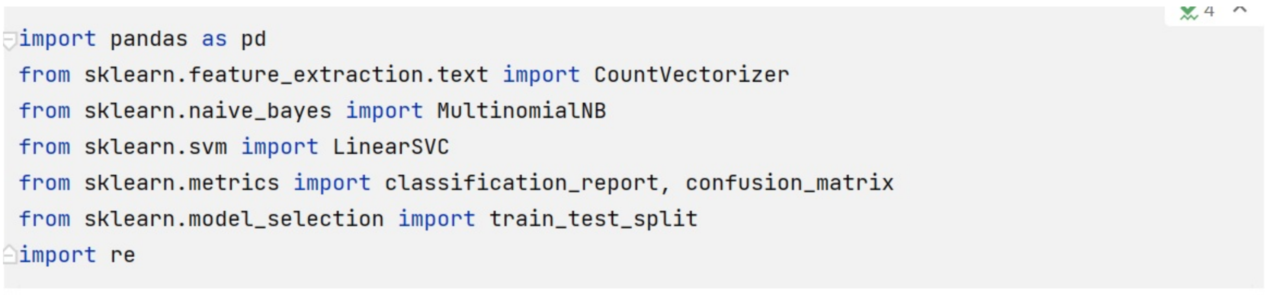


Figure 6: Elbow method

Question 3

The results for Question 3 is based on the Python file ‘A2Q3.ipynb’ in attachment file. First, we import the necessary modules and libraries:

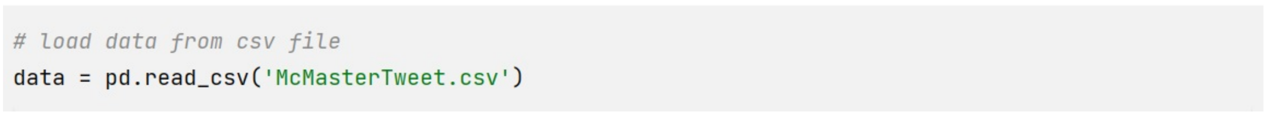


```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import re
```

Figure 7: Importing the necessary modules and libraries

pandas is a library for data manipulation and analysis. We’ll use it to read in and process our data. numpy is a library for numerical computing. We’ll use it to manipulate arrays of data. re is a module for regular expressions. We’ll use it to clean our text data.

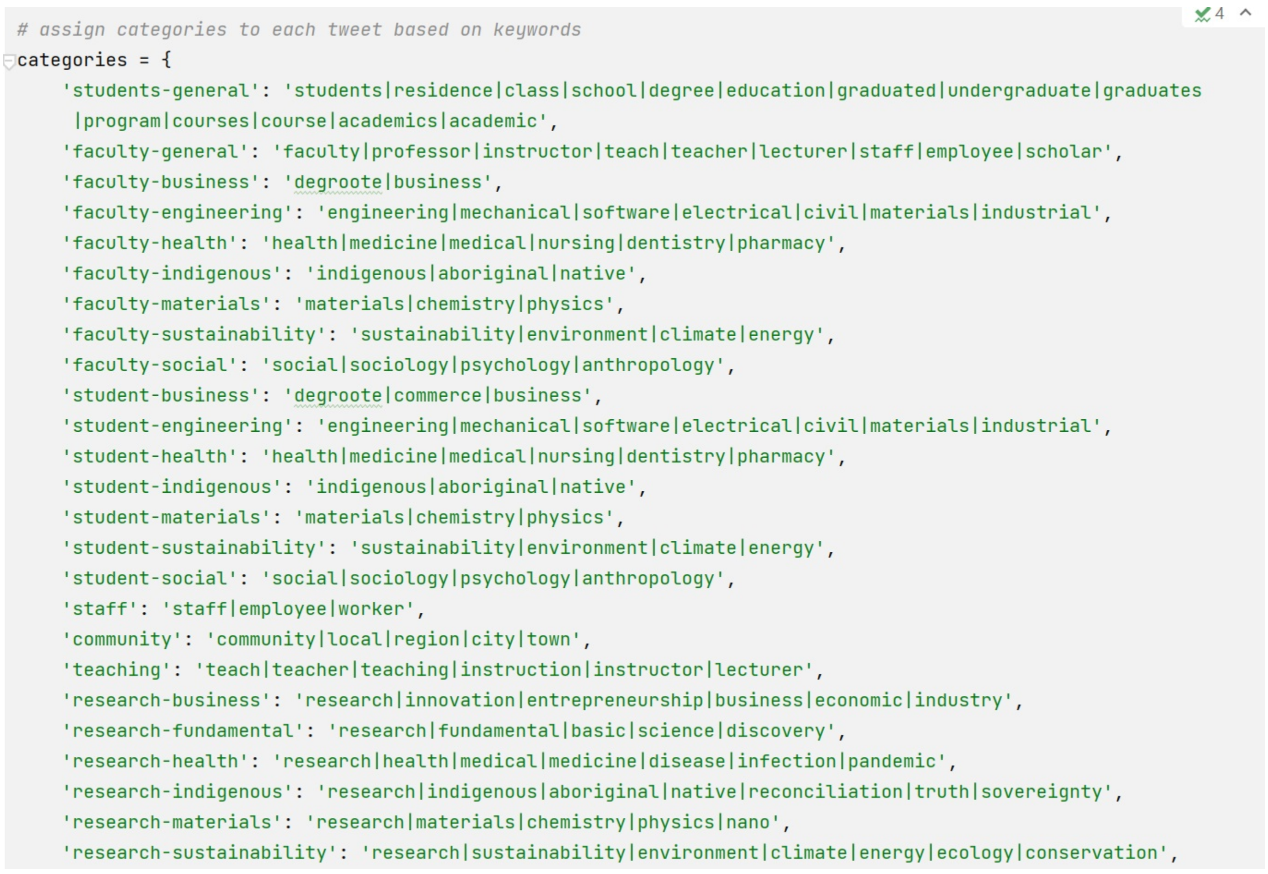
Next, we read in the data



```python
# load data from csv file
data = pd.read_csv('McMasterTweet.csv')
```

Figure 8: Reading the data

Then we define a dictionary called categories that assigns a category name to a set of keywords:



```python
# assign categories to each tweet based on keywords
categories = {
    'students-general': 'students|residence|class|school|degree|education|graduated|undergraduate|graduates
     |program|courses|course|academics|academic',
    'faculty-general': 'faculty|professor|instructor|teach|teacher|lecturer|staff|employee|scholar',
    'faculty-business': 'degroote|business',
    'faculty-engineering': 'engineering|mechanical|software|electrical|civil|materials|industrial',
    'faculty-health': 'health|medicine|medical|nursing|dentistry|pharmacy',
    'faculty-indigenous': 'indigenous|aboriginal|native',
    'faculty-materials': 'materials|chemistry|physics',
    'faculty-sustainability': 'sustainability|environment|climate|energy',
    'faculty-social': 'social|sociology|psychology|anthropology',
    'student-business': 'degroote|commerce|business',
    'student-engineering': 'engineering|mechanical|software|electrical|civil|materials|industrial',
    'student-health': 'health|medicine|medical|nursing|dentistry|pharmacy',
    'student-indigenous': 'indigenous|aboriginal|native',
    'student-materials': 'materials|chemistry|physics',
    'student-sustainability': 'sustainability|environment|climate|energy',
    'student-social': 'social|sociology|psychology|anthropology',
    'staff': 'staff|employee|worker',
    'community': 'community|local|region|city|town',
    'teaching': 'teach|teacher|teaching|instruction|instructor|lecturer',
    'research-business': 'research|innovation|entrepreneurship|business|economic|industry',
    'research-fundamental': 'research|fundamental|basic|science|discovery',
    'research-health': 'research|health|medical|medicine|disease|infection|pandemic',
    'research-indigenous': 'research|indigenous|aboriginal|native|reconciliation|truth|sovereignty',
    'research-materials': 'research|materials|chemistry|physics|nano',
    'research-sustainability': 'research|sustainability|environment|climate|energy|ecology|conservation',
```

Figure 9: Defining a mapping of categories to keywords

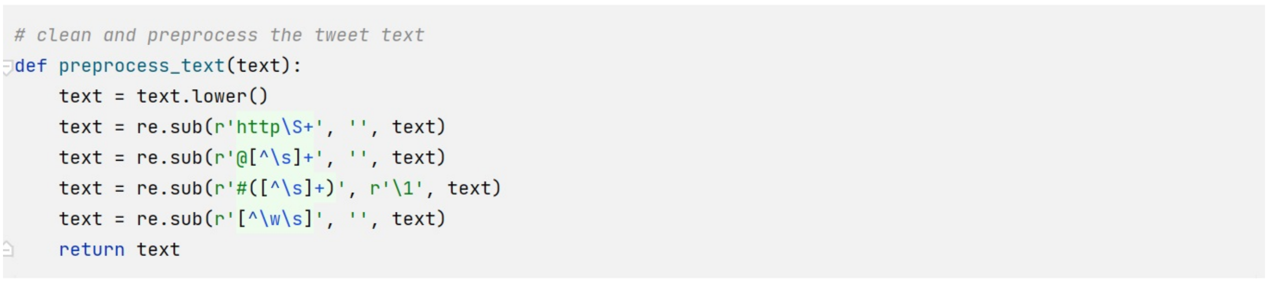
This dictionary is used to categorize tweets based on their content.

Each category is represented by a string key, such as ’students-general’, ’faculty-business’, ’research-health’, etc. The value for each key is a string that contains a list of keywords associated with that category.

For example, the ’students-general’ category is associated with keywords such as ’students’, ’residence’, ’class’, ’school’, ’degree’, etc., which are relevant to topics related to students in general. Similarly, the ’faculty-business’ category is associated with the keywords ’degroote’ and ’business’, which are likely relevant to topics related to business faculty.

The keywords are separated by the | symbol, which means that any of these keywords can be present in a tweet to be assigned to the corresponding category.

Then, we preprocess the data with a function which is useful for standardizing the text format and removing noisy or irrelevant information from tweet text data.



```python
# clean and preprocess the tweet text
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'http\S+', '', text)
    text = re.sub(r'@[^\s]+', '', text)
    text = re.sub(r'#([^\s]+)', r'\1', text)
    text = re.sub(r'[^\w\s]', '', text)
    return text
```

Figure 10: Preprocessing the data

We use a function called preprocess text that takes in a string parameter text and returns a preprocessed version of that text. The function uses regular expressions to remove certain patterns from the text.

text = text.lower(): Convert the text to lowercase.

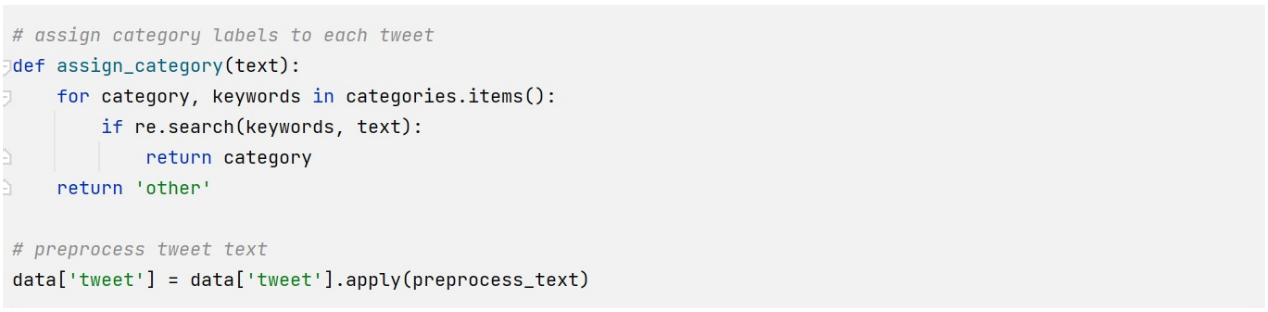
text = re.sub(r‘http\S+’, ‘’, text): Remove any URLs from the text.

text = re.sub(r‘@[\s]+’, ‘’, text): Remove any mentions of Twitter users (e.g., @username) from the text.

text = re.sub(r‘([\s]+)’, r‘\1’, text): Remove any hashtags from the text, but keep the text that followed the ‘’ character.

text =re.sub(r‘[\w\s]’, ‘’, text): Remove any non-alphanumeric characters (i.e., characters that are neither letters nor digits) from the text.

After preprocessing, we assigns category labels to tweet text data using the categories dictionary and the assign-category() function. The code also pre-processes the tweet text data by applying the preprocess-text() function to the ’tweet’ column of the data DataFrame.

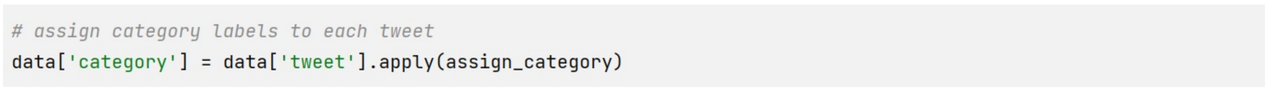


```python
# assign category labels to each tweet
def assign_category(text):
    for category, keywords in categories.items():
        if re.search(keywords, text):
            return category
    return 'other'

# preprocess tweet text
data['tweet'] = data['tweet'].apply(preprocess_text)
```

Figure 11: Assigning category labels to each tweet

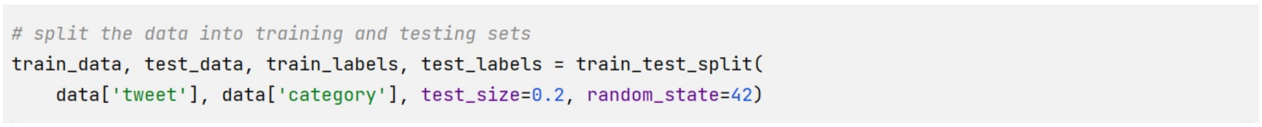
So, we assigns category labels to tweet text data using the assign-category() function and adds the resulting category labels as a new column called ‘category’ to the data DataFrame.



```python
# assign category labels to each tweet
data['category'] = data['tweet'].apply(assign_category)
```

Figure 12: Categorizing tweet data and adding the resulting category labels to the data DataFrame

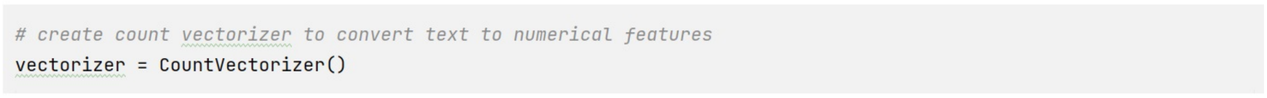
Now, we splits the tweet data into training and testing sets using the train- test- split() function from the Scikit-learn library. The resulting data is stored in four variables: train-data, test-data, train-labels, and test-labels.



```python
# split the data into training and testing sets
train_data, test_data, train_labels, test_labels = train_test_split(
    data['tweet'], data['category'], test_size=0.2, random_state=42)
```

Figure 13: Spliting the tweet data into training and testing sets

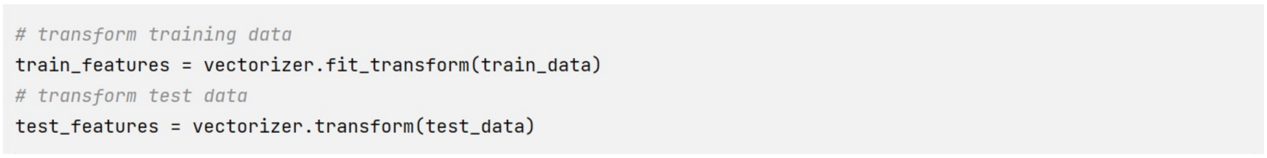
Then, we creates a CountVectorizer object from the Scikit-learn library.



```
# create count vectorizer to convert text to numerical features
vectorizer = CountVectorizer()
```

Figure 14: Creates a CountVectorizer object

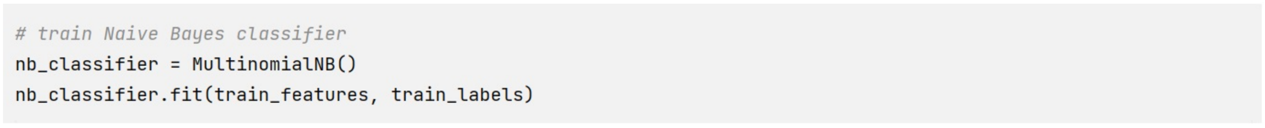
Then, we transforms the train and test data using the CountVectorizer object created in the previous code block.



```
# transform training data
train_features = vectorizer.fit_transform(train_data)
# transform test data
test_features = vectorizer.transform(test_data)
```

Figure 15: Transforms the training data using the CountVectorizer object

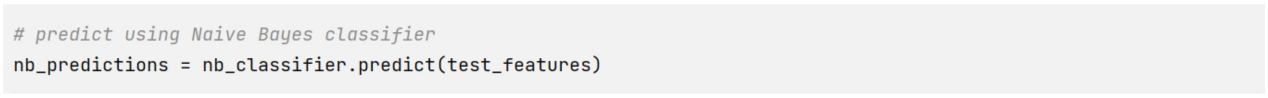
Then, we trains a Naive Bayes classifier using the MultinomialNB class from the Scikit-learn library.



```
# train Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(train_features, train_labels)
```

Figure 16: Trains a Naive Bayes classifier

Then, we uses the Naive Bayes classifier that was trained on the training data to make predictions on the test data.



```
# predict using Naive Bayes classifier
nb_predictions = nb_classifier.predict(test_features)
```

Figure 17: Making predictions on the test data

Figure 18 shows the confusion matrix for NB.

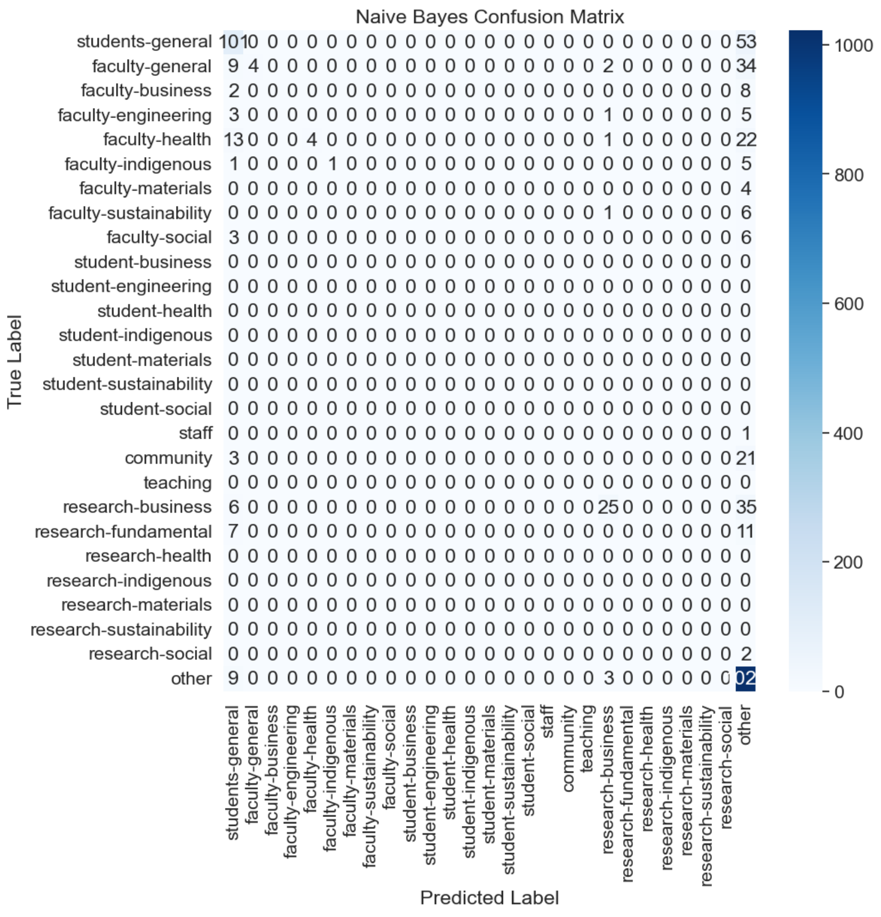


Figure 18: Confusion matrix for NB

Then, we trains a Support Vector Machine classifier and make predictions on the test data.

```
# train SVM classifier
svm_classifier = LinearSVC()
svm_classifier.fit(train_features, train_labels)

# predict using SVM classifier
svm_predictions = svm_classifier.predict(test_features)

# print confusion matrix and classification report for SVM
svm_confusion_matrix = confusion_matrix(test_labels, svm_predictions, labels=list(categories.keys()))
print("SVM Confusion Matrix:")
print(svm_confusion_matrix)
print()
print("SVM Classification Report:")
print(classification_report(test_labels, svm_predictions))
```

Figure 19: Training and predicions with SVM on the data

Figure 20 shows the confusion matrix for NB.

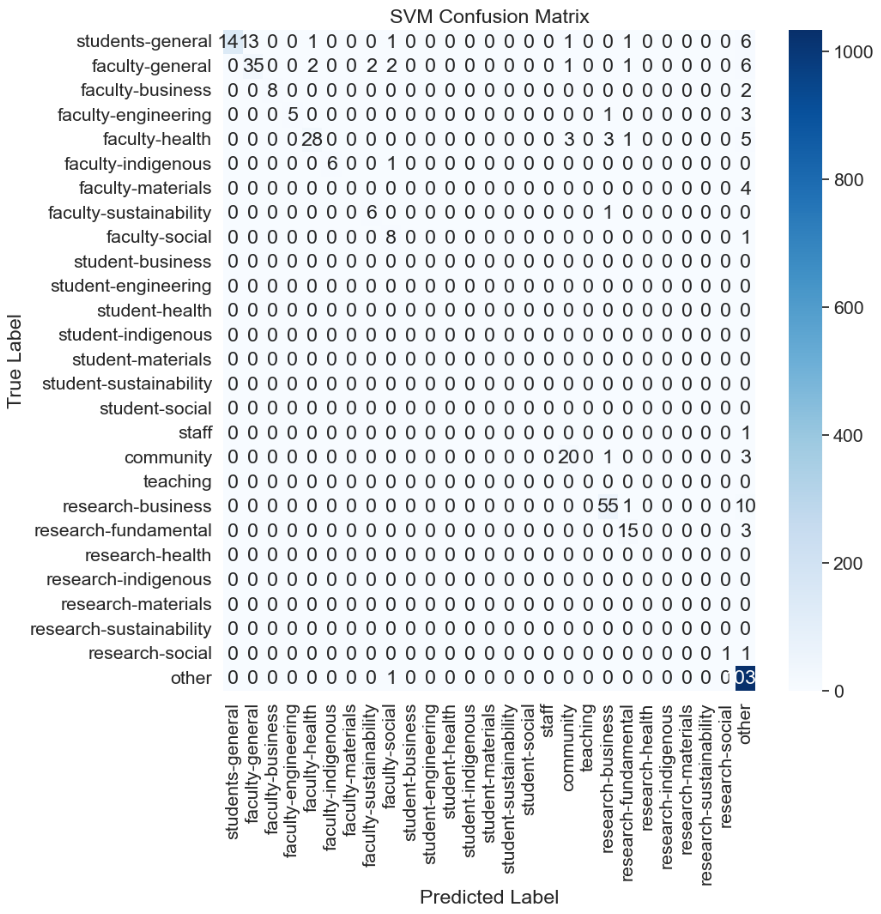
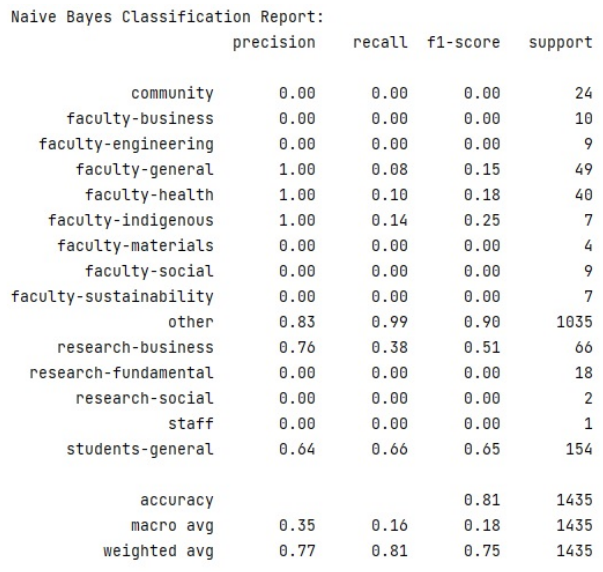


Figure 20: Confusion matrix for NB

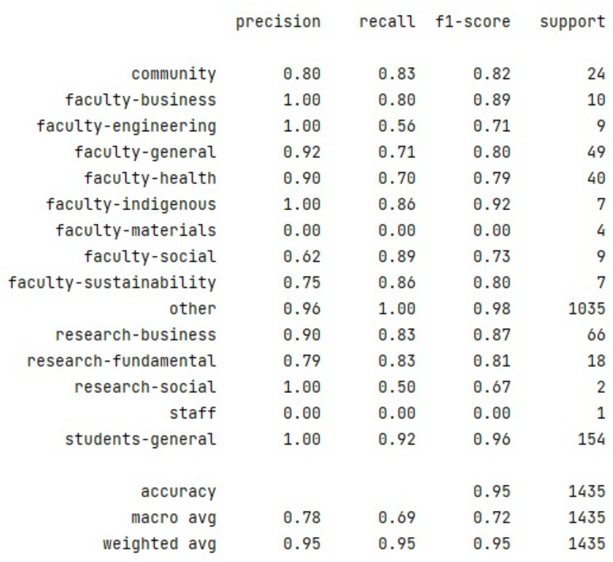


```
Naive Bayes Classification Report:
                        precision    recall  f1-score   support

             community       0.00      0.00      0.00        24
      faculty-business       0.00      0.00      0.00        10
   faculty-engineering       0.00      0.00      0.00         9
       faculty-general       1.00      0.08      0.15        49
        faculty-health       1.00      0.10      0.18        40
    faculty-indigenous       1.00      0.14      0.25         7
     faculty-materials       0.00      0.00      0.00         4
        faculty-social       0.00      0.00      0.00         9
faculty-sustainability       0.00      0.00      0.00         7
                 other       0.83      0.99      0.90      1035
     research-business       0.76      0.38      0.51        66
  research-fundamental       0.00      0.00      0.00        18
       research-social       0.00      0.00      0.00         2
                 staff       0.00      0.00      0.00         1
      students-general       0.64      0.66      0.65       154

              accuracy                           0.81      1435
             macro avg       0.35      0.16      0.18      1435
          weighted avg       0.77      0.81      0.75      1435
```

Figure 21: NB Classification Report



```
                         precision    recall  f1-score   support

              community       0.80      0.83      0.82        24
       faculty-business       1.00      0.80      0.89        10
    faculty-engineering       1.00      0.56      0.71         9
        faculty-general       0.92      0.71      0.80        49
         faculty-health       0.90      0.70      0.79        40
     faculty-indigenous       1.00      0.86      0.92         7
      faculty-materials       0.00      0.00      0.00         4
         faculty-social       0.62      0.89      0.73         9
 faculty-sustainability       0.75      0.86      0.80         7
                  other       0.96      1.00      0.98      1035
      research-business       0.90      0.83      0.87        66
   research-fundamental       0.79      0.83      0.81        18
        research-social       1.00      0.50      0.67         2
                  staff       0.00      0.00      0.00         1
       students-general       1.00      0.92      0.96       154

               accuracy                           0.95      1435
              macro avg       0.78      0.69      0.72      1435
           weighted avg       0.95      0.95      0.95      1435
```

Figure 22: SVM Classification Report

The classification report shows the precision, recall, and F1-score for each class, as well as the support (number of instances) for each class. Precision is the ratio of true positives to the total number of positive predictions, recall is the ratio of true positives to the total number of actual positives, and F1-score is the harmonic mean of precision and recall.

Based on 21 and 22, the main findings are as follows: All in all, the “final accuracy” and the “average accuracy” are higher in SVM comparing to NB. Also, The weighted average F1-score for SVM is 0.95, which is a good measure of overall performance. so it can be concluded that SVM is a better classifier for McMaster University tweets.

Based on confusion matrix, there is some class that have not any tweets, e.i. none of the tweets are classified in those classes. It’s because that some of the categories did not appear in the test set, and therefore do not have true labels associated with them. If we had a bigger data, we could divide the train and test data set instead of $20\%, 80\%$ with for instance $50\%, 50\%$, so we can have a more reliable results.

Question 4

The results for Question 4 is based on Excel file ‘A2Q4DataGeneration’, and Python files ‘A2Q4Tuning.ipynb’ and ‘A2Q4.ipynb’ in attachment file.

For solving this problem, total number of items (n) is assumed to be 5. In the data generation phase, 1000 records are created using the “rand()” function of Excel. The detailed procedure is as follows:

1. Random numbers between 1 and 1000 are generated for ci and ai.

2. To generate data for parameter b, as we know, if $b \geq \sum_i a_i$, the solution is obvious, and all the items will be selected. So, to have more appropriate data, we have considered $b = \alpha \sum_i a_i . \alpha$ is assumed to be equal to 0.1, 0.2, 0.3, 0.4,0.4, 0.5, 0.6, 0.7, 0.8, and 0.9 for each 100 consecutive samples (α for records number 1 to 100 is 0.1, it is 0.2 for 101-200 samples, and so on).

3. Then all the problems solved using GAMS software using compercial solver CPLEX, and the GAMS results are entered to the Excel file. Then, we randomly shuffled data. The results are presented in the “FinalData” Sheet.

Then, we add the following columns:

Ratio of item values to item weights (c1/a1, c2/a2, c3/a3, c4/a4, c5/a5): These columns will enable you to study the effect of the value-to-weight ratio on the objective function.

Ratio of item weights to knapsack capacity (a1/b, a2/b, a3/b, a4/b, a5/b): These columns will enable you to study the effect of the weight-to-capacity ratio on the objective function.

Total weight: sumproduct of ci and ai.

Weithed Priority ci: which is calculated with formulla $\frac{c_i}{\sum_i}$.

Min ci/ai: which is the minimum amount of $\frac{c_i}{a_i}$.

Max ci/ai: which is the maximum amount of $\frac{c_i}{a_i}$.

Min ai/b: which is the minimum amount of $\frac{a_i}{b}$.

Max ai/b: which is the maximum amount of $\frac{a_i}{b}$.

The shape of our data is 1000 object with 32 variables.

In the next phase, we want to predict the target variable ‘profit’, based on other variables in our data. As we can see in Figure 23 the relationship between the predictor variables and the target variable is nonlinear and complex. This is confirm with the result of leaner regression with PCA, that the

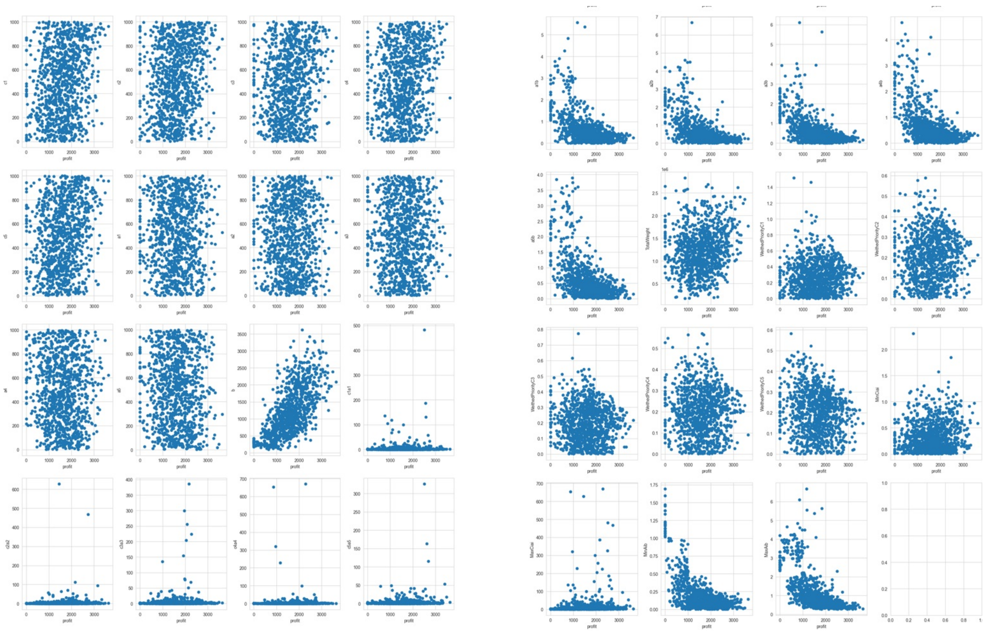


Figure 23: Relationship between the predictor variables and the target variable Profit

performance is not good.

Regression Trees are a powerful and flexible technique for predicting a target variable, and they are particularly useful when we have nonlinear relationships between target variable and other variables. They can capture these relationships in a flexible and interpretable way. On the other hand, since there are a large number of predictor variables, and it is difficult to understand which ones are most important, regression trees are sutable .But, it is important to note that regression trees can be overfitted, so we tune the parameter 'max depth' and it is equal to 7.

The results can be seen in the Table 1, and it is not satisfying. So, we apply random forest on the data. Random Forests is an ensemble learning method based on constructing multiple decision trees. Random Forests can be used both for classification and regression. Random Forests correct the overfitting of decision trees. In the Random Forests, a number of decision trees are built on bootstrapped training samples, and for each split, in a tree, a random sample of predictors, m, is chosen from the full set of p predictors. The number of predictors considered at each split is typically equal to the square root of the total number of predictors. The Random Forests has some pros and cons. This method is great with high dimensional data; even though each decision tree has a high variance since we use the average of all the trees, i.e. we are using the average of variance, which means a moderate variance model with low bias. Random Forests is robust since it limits

Machine Learning Algorithem	MAE	RMSE	R^2
Linear Regression with PCA	455.1301	563.1818	0.4248
Regression Tree	317.9768	420.1666	0.6799
Random Forest	237.4708	300.6857	0.8360

Table 1: Results from Machine Learning Algorithm

over-fitting and yield useful results, and the most important parameter is the number of variable in each split. Other parameters for tuning are the number of trees to grow, and it is better not to be set small. The maximum depth of the tree - meaning the longest path between the root node and the leaf node. The minimum number of samples required to split an internal node. The maximum number of leaf nodes a decision tree can have. The minimum number of samples required to be at a leaf node. The fraction of the original dataset that is given to any individual tree.

Then, According to the thumb rule, we can choose $p/3$ variables at a split when building a random forest of regression trees, and $\sqrt{p}$ variables at a split when building a random forest of classification trees. Also we tune the parameter 'max depth' and 'n estimators' which are equal to 20 and 200 respectively. We can see in the Figure 24 the importance of variable for random forest algorithm. Based on this, the variable $Maxai/b$, b , and $Maxci/ai$ is most important first three factors. As, we can see in the Table 1 Random Forest has best performance. We also applied the XGBoost, but it gives us very bad results. So, we do not inter the results form this to our results. It could happens if we do insufficient hyperparameter tuning. When we try tunning it gets too long time to respond. On the other hand, insufficient feature engineering may lead to a week performance of XGBoost. XGBoost is a powerful model, but it can only do as well as the features provided to it. If the features are not informative or relevant, the model may not perform well.Also, insufficient data may leas to week performance of XGBoost, since it requires a large amount of data to train effectively. If the dataset is too small, the model may not be able to learn the underlying patterns in the data and perform well.

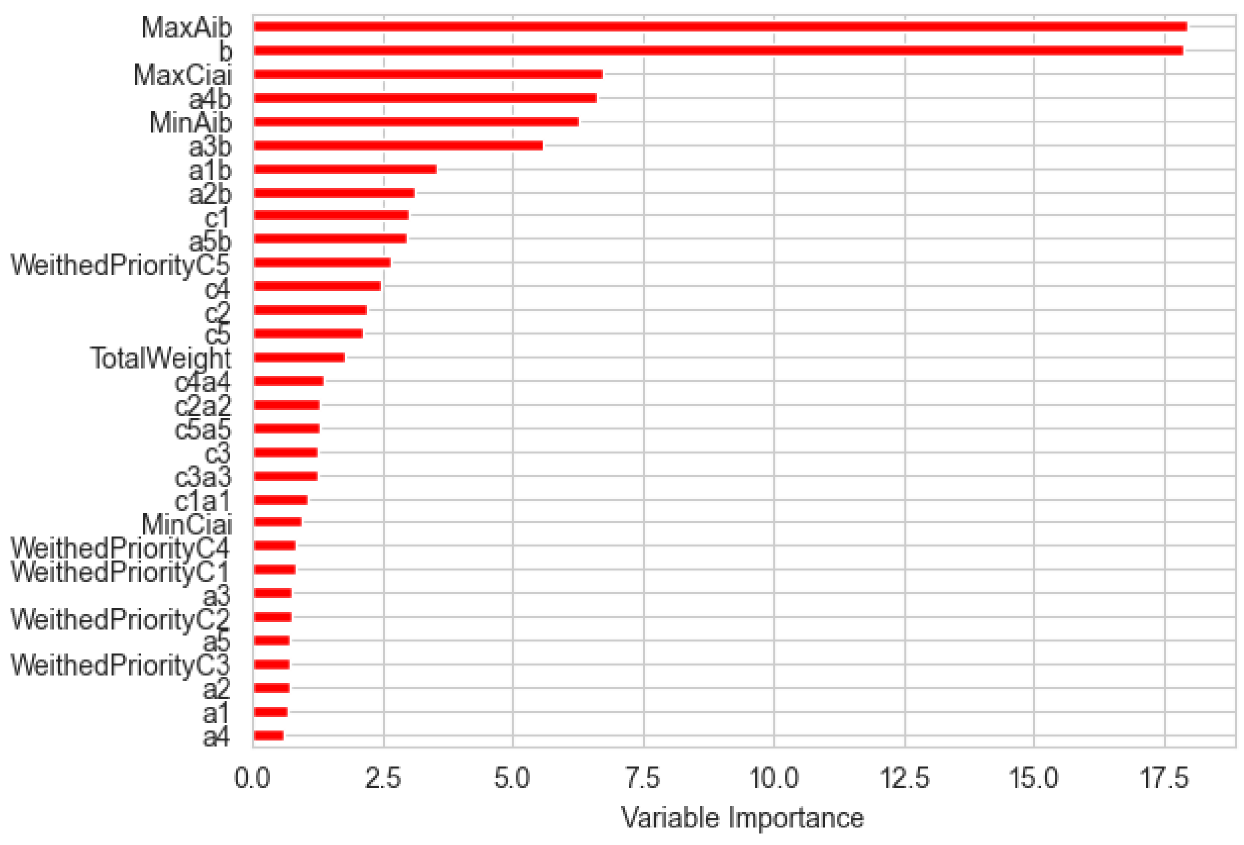


Figure 24: Variable importance in Random Forest

We test the Random Forest for an unseen data, the predict objective function is equal to 2103.20 and the objective function form exact solver is equal to 2293, so it means RF has an error near 8% for this single test that can be considered acceptable.

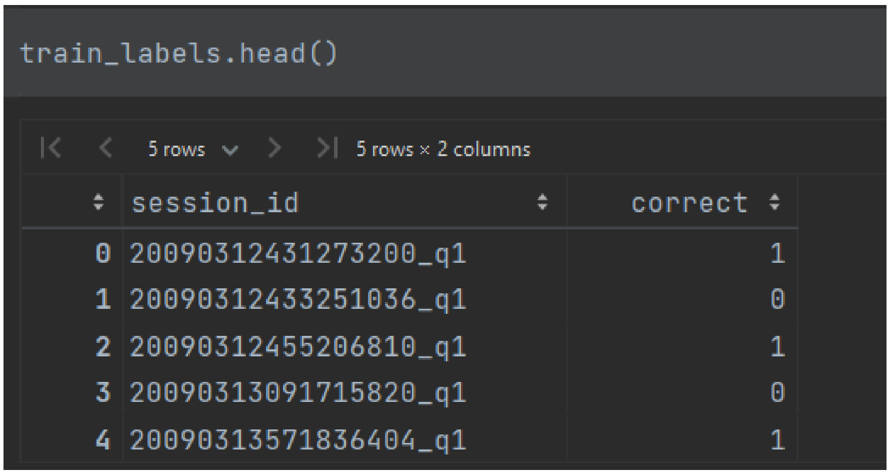
Question 5

(i). Assessing data sets and labelling training data.

1. Downloaded the data set and read it from my laptop (all three data sets: train, train_labels and test).

2. Randomly selected 1/10000 of the data because data was extremely big.

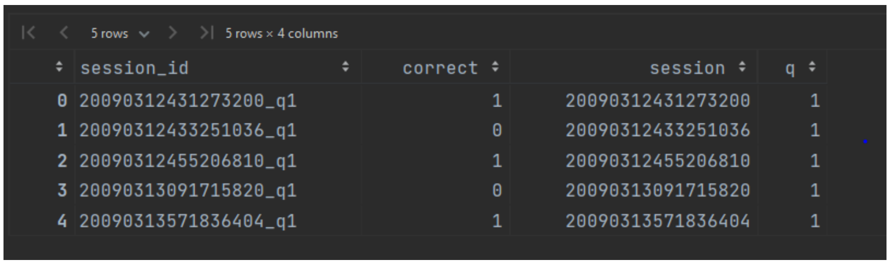
3. To label the data, first train_label data set column named 'session_id' was split into two columns showing session_id and question separately.



```
train_labels.head()

5 rows × 2 columns
   session_id          correct
0  20090312431273200_q1      1
1  20090312433251036_q1      0
2  20090312455206810_q1      1
3  20090313091715820_q1      0
4  20090313571836404_q1      1
```

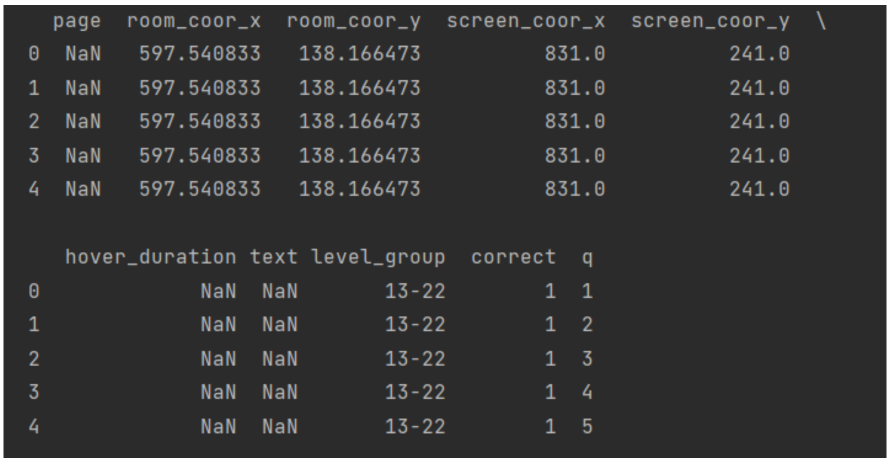
Figure 25: Train_labels data before splitting



```
5 rows × 4 columns
   session_id            correct  session            q
0  20090312431273200_q1        1  20090312431273200  1
1  20090312433251036_q1        0  20090312433251036  1
2  20090312455206810_q1        1  20090312455206810  1
3  20090313091715820_q1        0  20090313091715820  1
4  20090313571836404_q1        1  20090313571836404  1
```

Figure 26: Train_labels data after splitting

4. Train data and train_labels data has been merged on session_id to develop predictive model for student performance. Merged data is named as train_with_labels.



```
  page  room_coor_x  room_coor_y  screen_coor_x  screen_coor_y  \
0  NaN   597.540833   138.166473          831.0          241.0
1  NaN   597.540833   138.166473          831.0          241.0
2  NaN   597.540833   138.166473          831.0          241.0
3  NaN   597.540833   138.166473          831.0          241.0
4  NaN   597.540833   138.166473          831.0          241.0

   hover_duration text level_group  correct  q
0             NaN  NaN       13-22        1  1
1             NaN  NaN       13-22        1  2
2             NaN  NaN       13-22        1  3
3             NaN  NaN       13-22        1  4
4             NaN  NaN       13-22        1  5
```

Figure 27: Merging train and train_labels data sets

(ii). Data management, cleaning and ready for training and validation

1. Dealing with missing variables.

a. First Checking missing variable in both train_with_labels and test data.

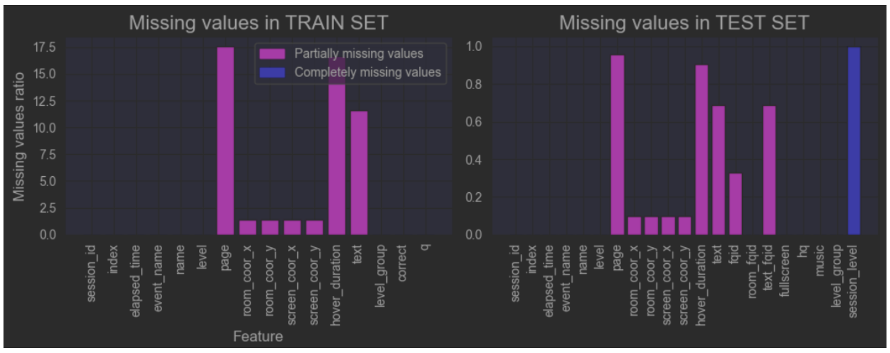


Figure 28: Checking missing data in train_with_labels and test data sets

b. Dropping all the columns which have large missing values and where few data was missing, there average value is set for the missing values.

c. Rechecking if there are any missing values.

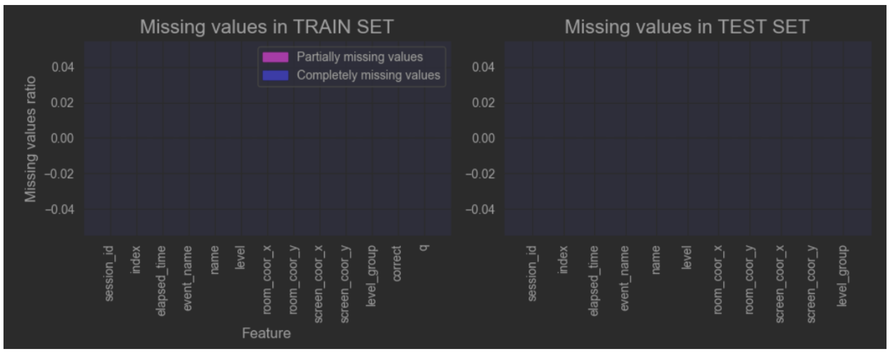
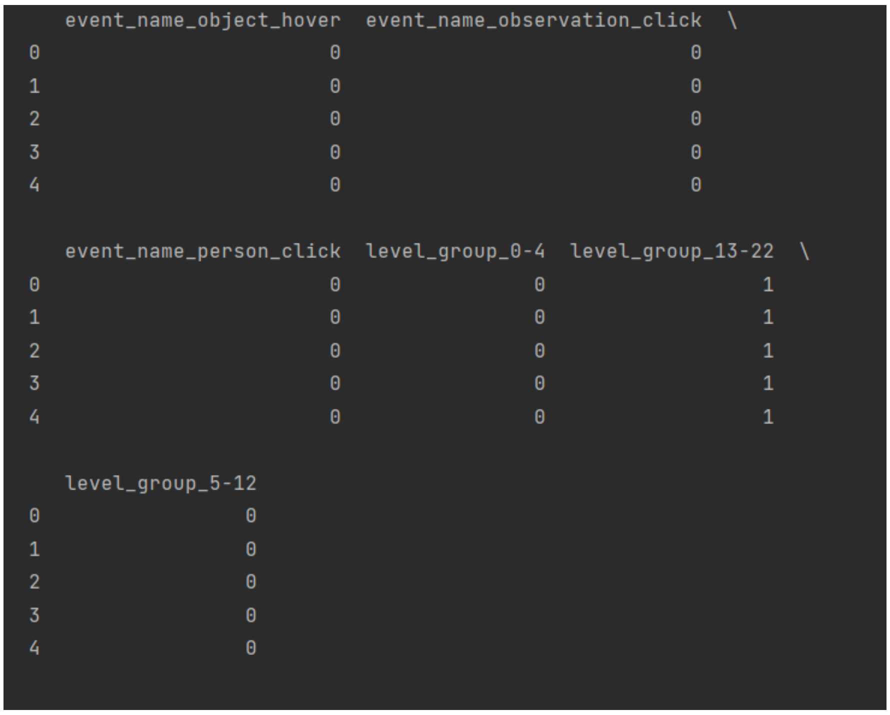


Figure 29: Rechecking missing data in train_with_labels and test data sets

2. Used one-hot encoding to convert categorical variables (columns: event_name, level_group) into binary.



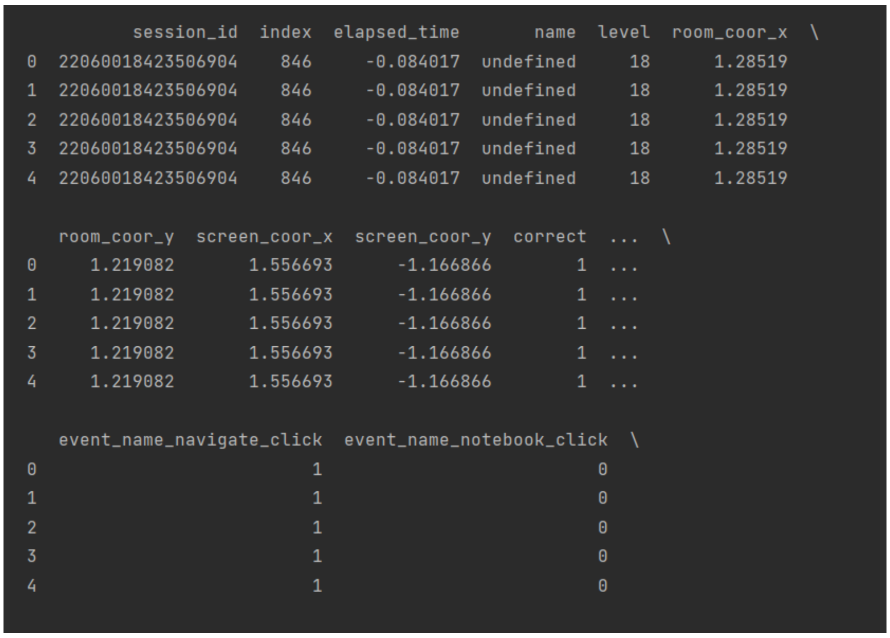
```
   event_name_object_hover  event_name_observation_click  \
0                        0                             0
1                        0                             0
2                        0                             0
3                        0                             0
4                        0                             0

   event_name_person_click  level_group_0-4  level_group_13-22  \
0                        0                0                  1
1                        0                0                  1
2                        0                0                  1
3                        0                0                  1
4                        0                0                  1

   level_group_5-12
0                 0
1                 0
2                 0
3                 0
4                 0
```

Figure 30: Applied one-hot encoding

3. Performed Scaling on numerical variables using standardization.(other than this we could have performed normalization instead of standardization, with the use of MinMaxScaler class from the same sklearn.preprocessing module.)



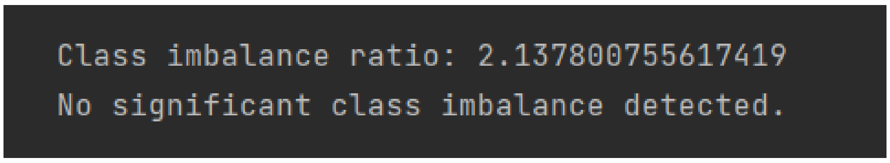
```
         session_id  index  elapsed_time       name  level  room_coor_x  \
0  22060018423506904    846     -0.084017  undefined     18      1.28519
1  22060018423506904    846     -0.084017  undefined     18      1.28519
2  22060018423506904    846     -0.084017  undefined     18      1.28519
3  22060018423506904    846     -0.084017  undefined     18      1.28519
4  22060018423506904    846     -0.084017  undefined     18      1.28519

   room_coor_y  screen_coor_x  screen_coor_y  correct  ...  \
0     1.219082       1.556693      -1.166866        1  ...
1     1.219082       1.556693      -1.166866        1  ...
2     1.219082       1.556693      -1.166866        1  ...
3     1.219082       1.556693      -1.166866        1  ...
4     1.219082       1.556693      -1.166866        1  ...

   event_name_navigate_click  event_name_notebook_click  \
0                          1                          0
1                          1                          0
2                          1                          0
3                          1                          0
4                          1                          0
```

Figure 31: Performed Scaling on numerical variables

4. Checking class imbalance. It help us know how many correct and incorrect questions are there in sample data. In this sample data, no imbalance is found.



```
Class imbalance ratio: 2.137800755617419
No significant class imbalance detected.
```

Figure 32: Checked class imbalance

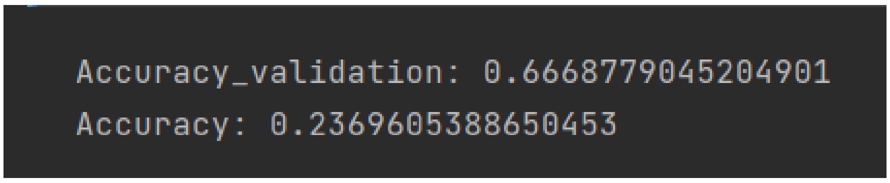
5. Similarly, we performed the analysis over test data. Ideally, complete training data should have used to develop model for the prediction and test data should have been used for testing.

But due to heavy data it is ideal to use training data only for validation and testing.

(iii) Train an appropriate predictive model for student performance

Ideal situation to do analysis(complete training data should have used):

When train data is used for training and test data given is used for testing, then it will not give appropriate results and accuracy will be extremely low. To show, we have build predictive model and found accuracy of 0.23 on test data. This is happening because training data is very small as compared to test data. (Refer code: for more details)



```
Accuracy_validation: 0.6668779045204901
Accuracy: 0.2369605388650453
```

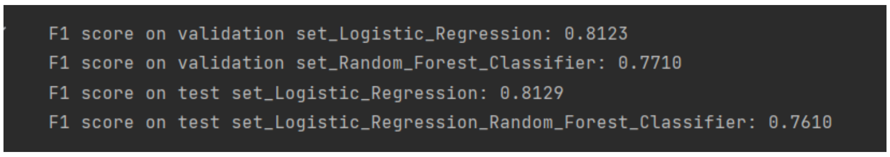
Figure 33: Accuracy on test data given in Kaggle site

If complete training data would have been used for building predictive model and then we would have got better accuracy on test data.

Real scenario (Randomly selecting training data):

So, we are using training data for validating and testing.

We have developed predictive model using Logistic Regression and Random Forest Classifier and found logistic regression perform better than random forest classifier. It is because number of noise variables are less than or equal to the number of explanatory variables in this case.

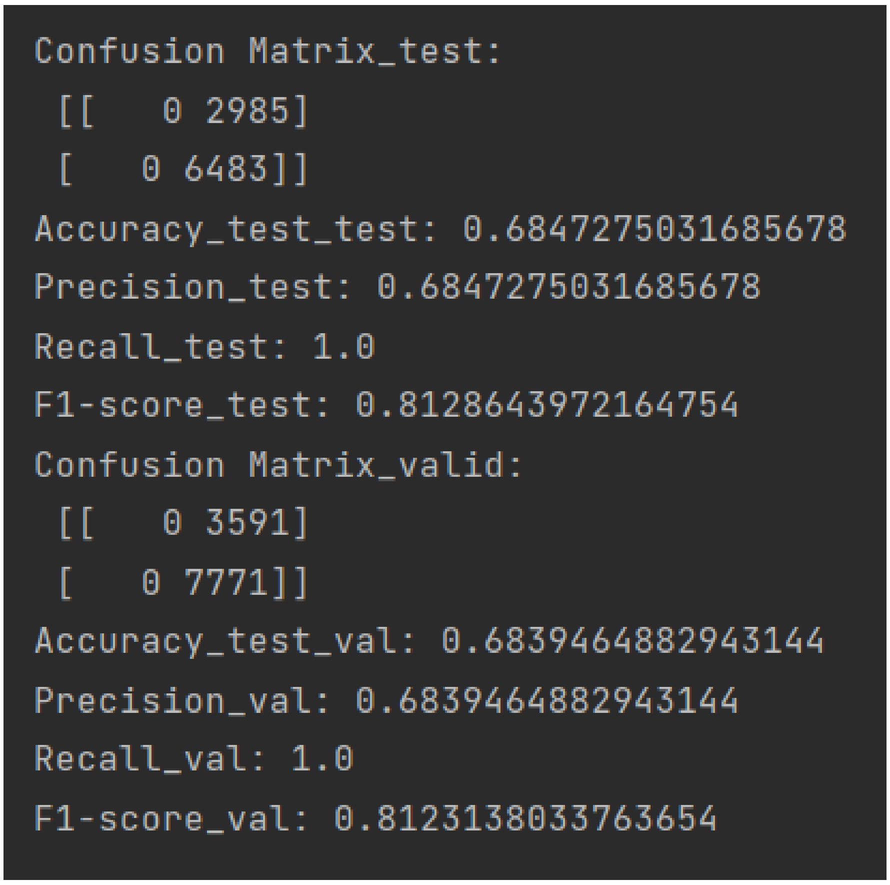


```
F1 score on validation set_Logistic_Regression: 0.8123
F1 score on validation set_Random_Forest_Classifier: 0.7710
F1 score on test set_Logistic_Regression: 0.8129
F1 score on test set_Logistic_Regression_Random_Forest_Classifier: 0.7610
```

Figure 34: F1 score of logistic regression and random forest classifier

(iv). Model evaluations with confusion matrix and F1 scores.

Since the logistic regression performed better, so models is evaluated for logistic regression with confusion matrix and F1 Score.



```
Confusion Matrix_test:
 [[   0 2985]
 [   0 6483]]
Accuracy_test_test: 0.6847275031685678
Precision_test: 0.6847275031685678
Recall_test: 1.0
F1-score_test: 0.8128643972164754
Confusion Matrix_valid:
 [[   0 3591]
 [   0 7771]]
Accuracy_test_val: 0.6839464882943144
Precision_val: 0.6839464882943144
Recall_val: 1.0
F1-score_val: 0.8123138033763654
```

Figure 35: Confusion matrix and F1 score of logistic regression model for evaluation

Here confusion matrix states that 2985 of true negative and 6483 of true positive has been identifying while testing data with the accuracy of 68%. In confusion matrix, there is no data for one class which shows model may not be performing well. So, other test methods are used to do analysis like Recall test and F1 test. Recall test is used to identify sensitivity and it shows that it is 1 showing all positive instances have been identified correctly. In addition, F1 Score is also high.

So, we conclude that model is performing well.