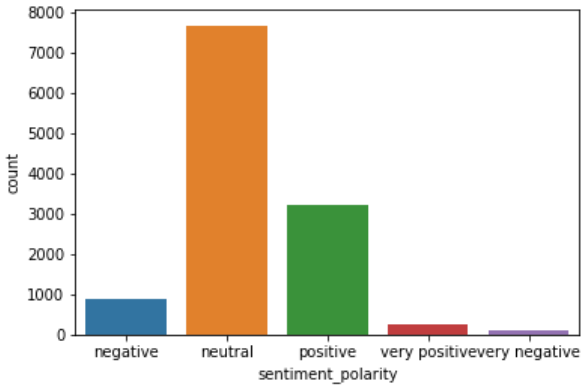
REDDIT DATA SENTIMENT ANALYSIS REPORT

This report summarises the Reddit data and reports on the sentiment analyzer build using a different classifier and vectorizer. Also, compares and contrasts the effects of various methods used for the final fine-tuned multiclass classifier.

Distribution of the class:
The distribution of labels in the training/validation/test sets is very imbalanced. The fig shows us that the 'neutral' class has a very high no. of samples compared to the other classes. The number of samples in the 'very positive' and 'very negative' classes is very low. This imbalance makes the classifier ignore small classes and mainly concentrate on classifying the large classes accurately. Evaluating the performance of the classifier through predictive accuracy, might not be very reliable in this case.



Different classifier with different Tokenization:

```
Evaluation for: Dummy_most_frequent_test
Classifier 'Dummy_most_frequent_test' has Acc=0.626 P=0.200 R=0.125 F1=0.154
```

```
Evaluation for: Dummy_stratified_test
Classifier 'Dummy_stratified_test' has Acc=0.470 P=0.206 R=0.205 F1=0.205
```

```
Evaluation for: LR_onehot_test
Classifier 'LR_onehot_test' has Acc=0.739 P=0.325 R=0.578 F1=0.349
```

```
Evaluation for: LR_TFIDF_test
Classifier 'LR_TFIDF_test' has Acc=0.739 P=0.325 R=0.578 F1=0.349
```

```
Evaluation for: SVC_test
Classifier 'SVC_test' has Acc=0.744 P=0.313 R=0.622 F1=0.331
```

A Dummy classifier is used as the baseline to compare the performance with the other useful classifiers. Stratified takes the training set class distribution into account to generate random predictions. The most_frequent strategy always predicts the most frequent label in the training set. Logistic Regression for

multiclass classification is implemented by training the classifier, one for each of the different classes in the training datasets. In the prediction phase, **one vs all** prediction is used to predict the highest probability of which class the test data belongs to. The logistic regression using one hot vectorizer and TfidfVectorizer pretty much performers similar. The main difference is the representation of the count matrix. Both Tfidf and count vectorizer produces term frequencies, TfidfTransformer uses normalization to transform the count matrix.
Similar to LogisticRegression, the support vector machine breaks the multiclass problem into multiple binary classifications called **one vs rest.** A binary classifier per each pair of classes. As we can see from the above results Logistic Regression outperforms the other classifiers, which considering the F1 score since the data is imbalanced.

```
Evaluation for: Random_forest_test
Classifier 'Random_forest_test' has Acc=0.717 P=0.630 R=0.538 macro-F1=0.567
              precision    recall  f1-score   support

           1       0.39      0.69      0.49        32
           2       0.52      0.42      0.47       282
           3       0.82      0.77      0.79      2514
           4       0.61      0.69      0.65      1102
           5       0.34      0.58      0.43        86

    accuracy                           0.72      4016
   macro avg       0.54      0.63      0.57      4016
weighted avg       0.73      0.72      0.72      4016

[[  22    4    6    0    0]
 [  19  119  126   18    0]
 [  16  100 1926  444   28]
 [   0    4  269  762   67]
 [   0    0   13   23   50]]
```

The pipeline used for this model contains three components a cleaner, a vectorizer, and a classifier.

- Cleaner: Cleaner uses the ‘predictors’ class object to clean and preprocess the text.
- Vectorizer: Here ‘CountVectorizer’ is used to create the bag of words matrix for our train_data[‘body’]. The custom tokenizer ‘spacy_tokenizer’ and ngram range (1,1) is used for the mean sequence of a single word.
- Classifier: Random forest classifier is used to perform the multiclass classification.

Overall this model correctly classified 71.7% of the classes. The overall macro F1 score is improved while comparing to our previous baseline classifier or any other previous models. We can also observe that the classification of the smaller classes is much better compared to the previous models. Preprocessing the text data using a cleaner object makes a significant difference in the overall performance.

Hyperparameter tuning of LogisticRegression-TFIDF:

- **C Parameter tuning:**
  C Parameter is considered as the regularization parameter C=1/λ. Where Lambda (λ) is the trade-off between increasing the model complexity while keeping it simple. Parameter C will work as the inverse of Lambda. For smaller values of C, the regularization strength is increased while the model complexity is simple and therefore underfits the data. For big C values, the regularization strength is reduced, which tends to increase the model complexity and therefore overfits the data. GridsearchCV is used to find the optimum values of the C parameter which tends to keep the model complexity and overfitting in balance. With the grid search, we use the parameter space that consists of C_param dictionaries which C values possibly from range $10^{-3} to 10^{5}$. **The optimum value for our data is C=100.**

- **TFIDF Parameter tuning:**
  TfidfVectorizer provides a convenient way to encode and transform texts into vectors. We can vary the parameters max_features and sublinear_tf and use GridSearchCV and pipeline objects to find the

optimum parameter tuning for our vectorizer. Max_features we tried, in this case, vary from None to 50K. This parameter builds a vocabulary that only considers the number of top max_features ordered by the term frequency in our respective corpus. Sublinear_tf is a scaling technique that replaces the tf to $1 + \log(tf)$ if given True. This is the other hyperparameter that is tuned to optimize our model. The grid search performed over these hyperparameters optimized the value of **max_features = 5000 and sublinear_tf= True.**

- **Classifier's solver parameter tuning:**
  Another hyperparameter tuning used here is figuring out the optimum solver parameter for the classifier. According to the GridSearch performed, **liblinear** is the best solver for our current model and its performance. Liblinear is limited to one vs rest classifications. The liblinear can handle L1 and L2 penalty but does not support the 'none' penalty. Liblinear solver cannot learn a multiclass model instead it disintegrates the problem into one vs rest to train. The logistic regression and linearSvc predict the negative class, instead, liblinear predicts the positive class, and the fit_intercept is assigned to be true to increase the intercept_scaling.

```
Evaluation for: Tuned_model
Classifier 'Tuned_model' has Acc=0.686 P=0.414 R=0.414 macro-F1=0.410
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.26      0.38      0.31       190
           3       0.79      0.76      0.77      2623
           4       0.59      0.60      0.60      1091
           5       0.43      0.33      0.38       111

    accuracy                           0.69      4016
   macro avg       0.41      0.41      0.41      4016
weighted avg       0.70      0.69      0.69      4016

[[   0    0    1    0    0]
 [   8   72   96   14    0]
 [  21  188 1990  401   23]
 [   2   18  391  654   26]
 [   1    4   36   33   37]]
```

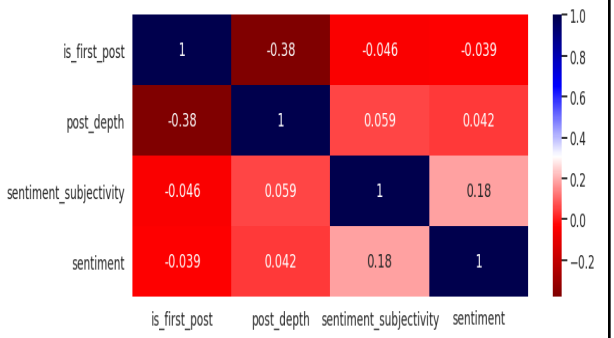
This is the final evaluation of the hyperparameters tuned model. From the above confusion matrix, we can infer that the class with low samples like class1, class2, and class3 is performed very poorly, even after the class_weights= 'balanced' is assigned to our logistic model. The accuracy is around 68% and the macro_f1 score is around 0.410 overall. Even though this tuned logistic regression model performs well than the baseline model, the Randomforest Classifier outperforms Logistic Regression in this case study.

Hypothesis on why Randomforest Classifier outperforms Logistic Regression is that Random Forest tends to prioritize feature selection, and weights feature according to their importance. And also does not assume the model is linear, unlike regression models does. Random forest samples randomly and forms decisions and averages the leaf nodes to get a better model thus uses ensemble learning. These could be some of the reasons why RandomForest performs better than Logistic Regression in this case.

Adding features to the Tuned Model and improving performance:

The strategy to find the best parameters to add to our pipeline is to use the trial and error method and verify the performance of our model after adding each feature significantly. This way we could find the best features to add to our tuned Logistic Regression pipeline without exhausting our pipeline and also avoiding the overfitting or the underfitting problems that might occur.

Plan 1: Adding a numerical feature to the pipeline




```
Tuned model with added 'is_fist_post' num feature
              precision    recall  f1-score   support

           1       0.53      0.31      0.39        32
           2       0.43      0.30      0.35       282
           3       0.76      0.80      0.78      2514
           4       0.59      0.58      0.59      1102
           5       0.44      0.28      0.34        86

    accuracy                           0.69      4016
   macro avg       0.55      0.46      0.49      4016
weighted avg       0.68      0.69      0.68      4016

[[  10    6   12    3    1]
 [   4   85  175   18    0]
 [   4  102 2014  384   10]
 [   1    4  434  644   19]
 [   0    0   20   42   24]]
```

The correlation matrix between the numerical features in the dataset. We can infer from the above heat map that sentiment_subjectivity and sentiment have a higher correlation between each other. Therefore adding them to the pipeline will results in little to no improvement. On the other side is_first_post has a negative correlation adds some new information for the model to work with. Therefore improves the overall working of the classifier. From the results above the f1 score has been improved significantly for the smaller classes and a better confusion matrix overall. So adding **is_first_post is a positive feature** to add to the pipeline.

Plan 2: Adding the categorial feature ‘subreddit to the pipeline:

Adding the feature ‘subreddit’ to the pipeline improves the performance, improves the overall performance compared to the tuned_model before but the result is a little bit lower than the is_first_post features. This might be because the subreddit and the body feature in the dataset will have a very high correlation since the context of both columns will be highly related to each other

```
Tuned model with added 'subreddit' cat feature
              precision    recall  f1-score   support

           1       0.42      0.25      0.31        32
           2       0.41      0.32      0.36       282
           3       0.76      0.79      0.77      2514
           4       0.58      0.58      0.58      1102
           5       0.42      0.26      0.32        86

    accuracy                           0.68      4016
   macro avg       0.52      0.44      0.47      4016
weighted avg       0.68      0.68      0.68      4016

[[   8    6   14    3    1]
 [   4   90  169   18    1]
 [   6  116 1986  394   12]
 [   1    5  436  643   17]
 [   0    0   19   45   22]]
```

Plan 3: Adding ‘post_depth’ as a num feature

Adding post_depth improves the tuned_model significantly better than the model with the single feature. This could be because as the post_depth increases the comment on the post gets more engaging and even after a heated conversation, the overall polarity would be ‘Neutral’. In very few cases the arguments will be one-sided positive or negative. This significantly classifies the smaller class better than the baseline or tuned model.

```
Tuned model with added 'post_depth' num feature
              precision    recall  f1-score   support

           1       0.50      0.28      0.36        32
           2       0.43      0.31      0.36       282
           3       0.76      0.80      0.78      2514
           4       0.59      0.59      0.59      1102
           5       0.42      0.27      0.33        86

    accuracy                           0.69      4016
   macro avg       0.54      0.45      0.48      4016
weighted avg       0.68      0.69      0.68      4016

[[   9    5   14    3    1]
 [   5   87  172   18    0]
 [   3  108 2007  384   12]
 [   1    4  432  646   19]
 [   0    0   21   42   23]]
```

Plan 4: Adding majority_type categorical features

Adding the majority_type categorical features definitely improves the overall classification of the model and as well as increases the F1 score for all the smaller as well as the major class. This might be because these features tell a lot above a particular sample either it is a question or an answer or any other categories. This adds more information for the model to learn and helps with the better classification of the sentiment_polarity.

```
Tuned model with added 'majority_type' cat feature
              precision    recall  f1-score   support

           1       0.45      0.28      0.35        32
           2       0.44      0.32      0.37       282
           3       0.76      0.80      0.78      2514
           4       0.59      0.59      0.59      1102
           5       0.43      0.27      0.33        86

    accuracy                           0.69      4016
   macro avg       0.53      0.45      0.48      4016
weighted avg       0.68      0.69      0.69      4016

[[   9    7   12    3    1]
 [   5   90  169   18    0]
 [   5  103 2009  386   11]
 [   1    3  432  647   19]
 [   0    0   21   42   23]]
```

Final Fine-tuning of the model with added features:

As per the above observations and the fine-tuning, the numerical features 'is_first_post' and the categorical feature 'majority_type' are added to the logistic regression model and the model is trained on the training data and the results are evaluated using the validation and the test data.

```
LogisticRegression(C=100,solver='liblinear',class_weight='balanced')
```

```
LogisticRegression(C=100, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

The final evaluation of the model using the test data is as follows:

```
Test set results
              precision    recall  f1-score   support

           1       0.07      0.31      0.11        32
           2       0.30      0.33      0.31       282
           3       0.77      0.73      0.75      2514
           4       0.64      0.57      0.60      1102
           5       0.19      0.40      0.25        86

    accuracy                           0.65      4016
   macro avg       0.39      0.47      0.41      4016
weighted avg       0.68      0.65      0.66      4016

[[  10    5   14    0    3]
 [  21   93  149   14    5]
 [  91  180 1844  316   83]
 [  19   29  369  628   57]
 [   2    3   25   22   34]]
```