# Code Project Spam Detection

By

Piyush Kumar MT2021092

Hussnain Asraf MT2021055

# Data Analysis

- Given data has 1685264 rows and 12 columns
- Column names with their description-
- ➢Qid : Id for each question posted
- ➢Post_date_time : Date and time at which question was posted
- ➢Karma : Total number of questions asked from that particular account
- ➢Num_answers : Total number of answers given for that particular question

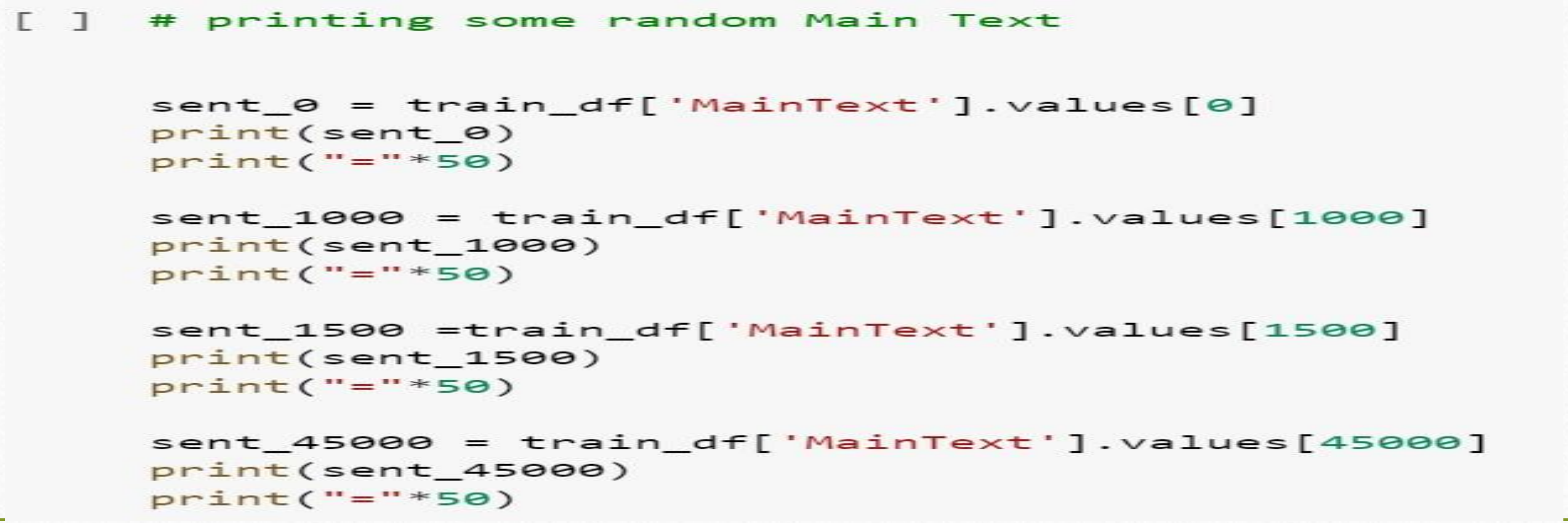
- Main text : Actual question asked
- Heading : Summary of question
- Primary Subject: Subject name of the question asked
- Secondary Subject: Chapter of subject
- Tertiary subject: Section of chapter
- Other Subject : More detailed description of topic names
- Target: 0,1,2,3,4 => labels for question asked

- 0 => valid
- 1 => kehna kya chahte ho
- 2 => irrelevant
- 3 => low quality
- 4 =>not reproducible

# EDA(Exploratory Data Analysis)

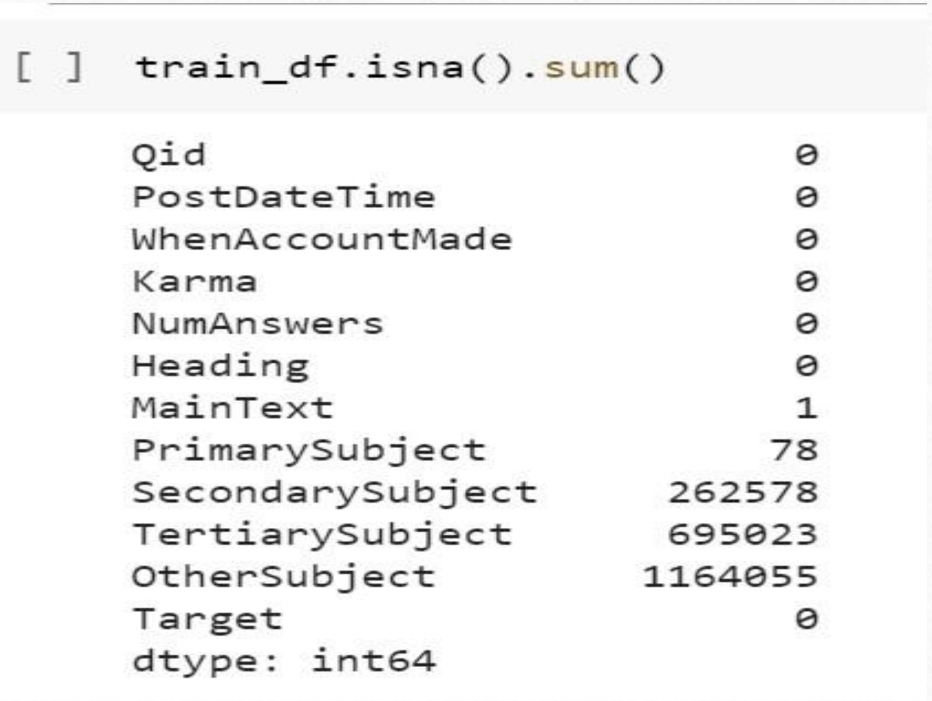
- Data Cleaning

1. Checking some random maintext and heading values



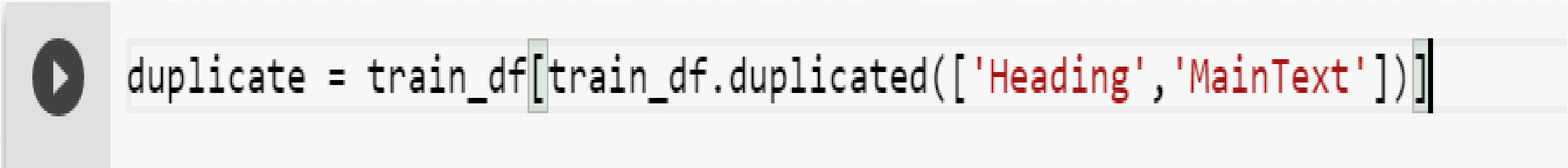
```
[ ]   # printing some random Main Text


      sent_0 = train_df['MainText'].values[0]
      print(sent_0)
      print("="*50)

      sent_1000 = train_df['MainText'].values[1000]
      print(sent_1000)
      print("="*50)

      sent_1500 =train_df['MainText'].values[1500]
      print(sent_1500)
      print("="*50)

      sent_45000 = train_df['MainText'].values[45000]
      print(sent_45000)
      print("="*50)
```

- 2. Null value count for each column

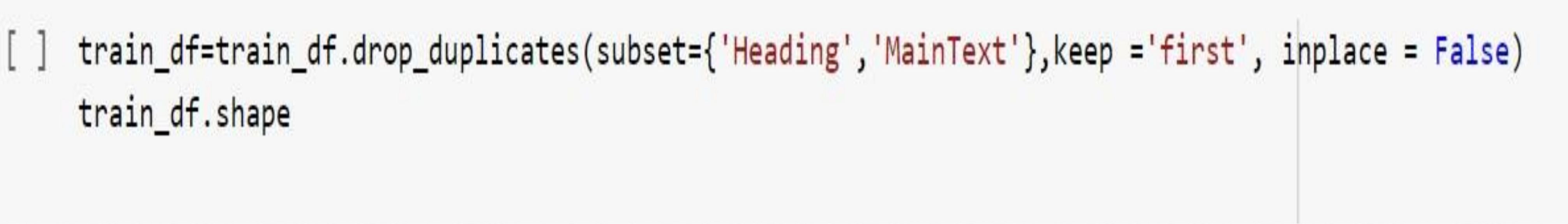


```
[ ]  train_df.isna().sum()

     Qid                        0
     PostDateTime               0
     WhenAccountMade            0
     Karma                      0
     NumAnswers                 0
     Heading                    0
     MainText                   1
     PrimarySubject            78
     SecondarySubject      262578
     TertiarySubject       695023
     OtherSubject         1164055
     Target                     0
     dtype: int64
```

- 3. Duplicate rows w.r.t heading and maintext ( keep first and remove all )

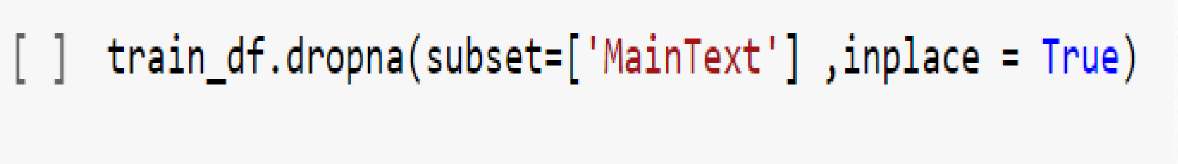


```
duplicate = train_df[train_df.duplicated(['Heading','MainText'])]
```



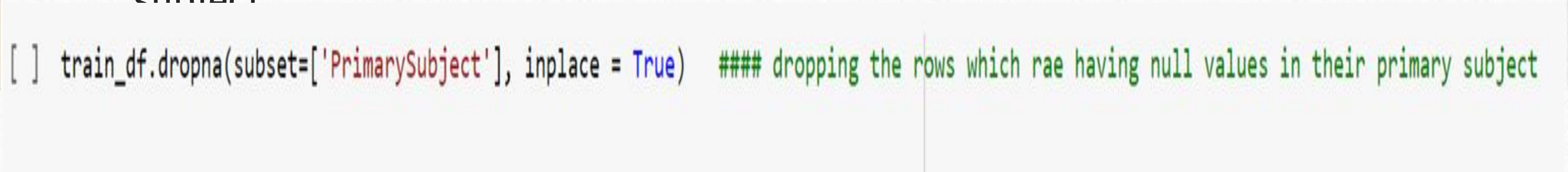
```
train_df=train_df.drop_duplicates(subset={'Heading','MainText'},keep ='first', inplace = False)
train_df.shape
```

- 4. Dropping row with no maintext ( only 1 such row is there )



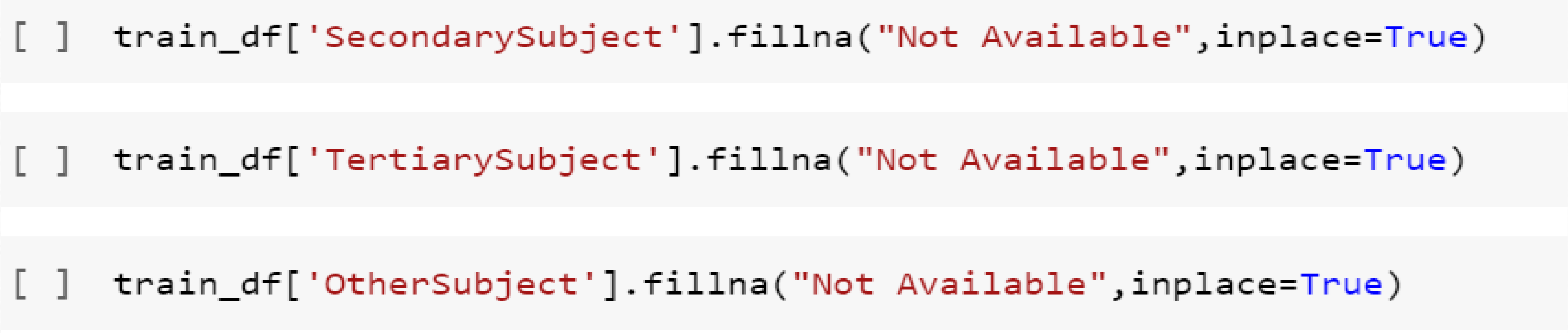
```
[ ]  train_df.dropna(subset=['MainText'] ,inplace = True)
```

- 5. Dropping rows with no primary subject



```
train_df.dropna(subset=['PrimarySubject'], inplace = True)   #### dropping the rows which rae having null values in their primary subject
```

- 6. Replacing NAN of Secondary, Tertiary and othersubject column with
  “Not Available”



```
[ ]  train_df['SecondarySubject'].fillna("Not Available",inplace=True)
```

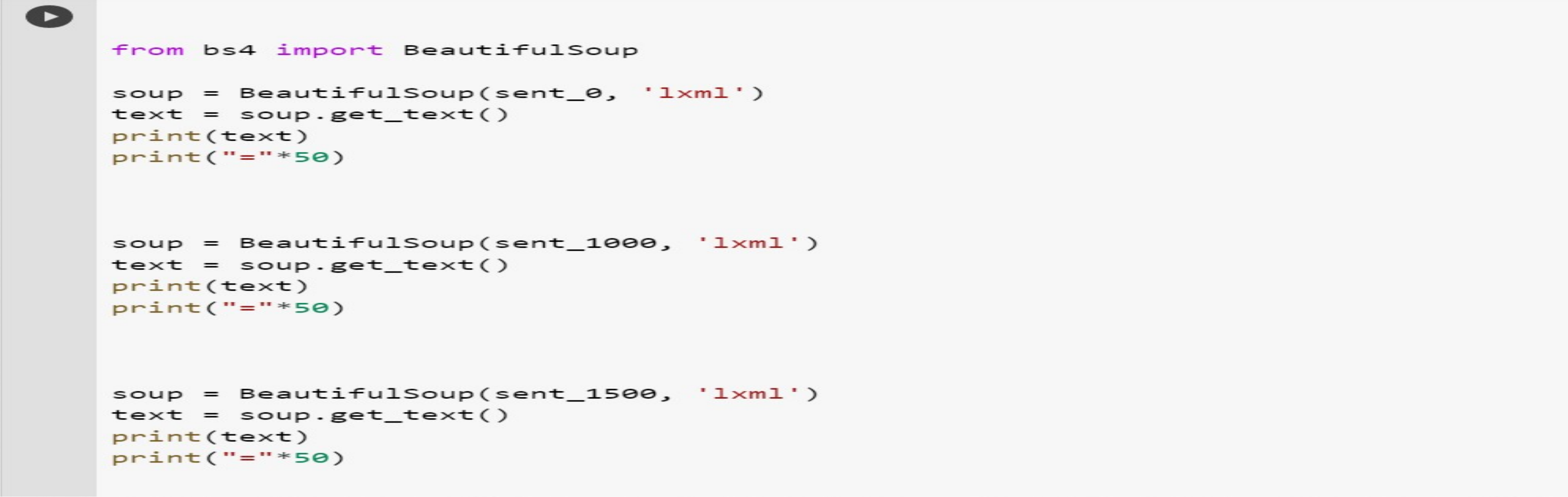
```
[ ]  train_df['TertiarySubject'].fillna("Not Available",inplace=True)
```

```
[ ]  train_df['OtherSubject'].fillna("Not Available",inplace=True)
```

1. Duplicate removal considering rows with different heading but same maintext
2. Same steps on test data too

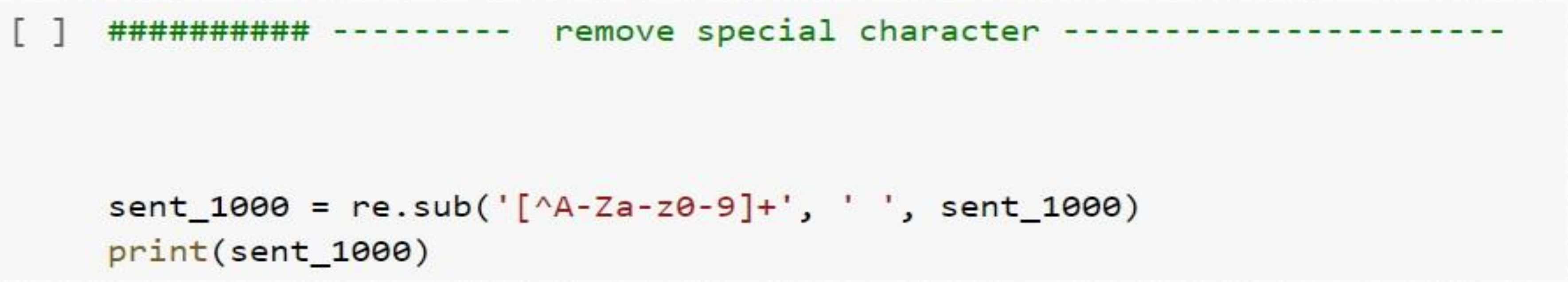
# MainText preprocessing

- 1. Removal of HTML and XML tags using BeautifulSoup library



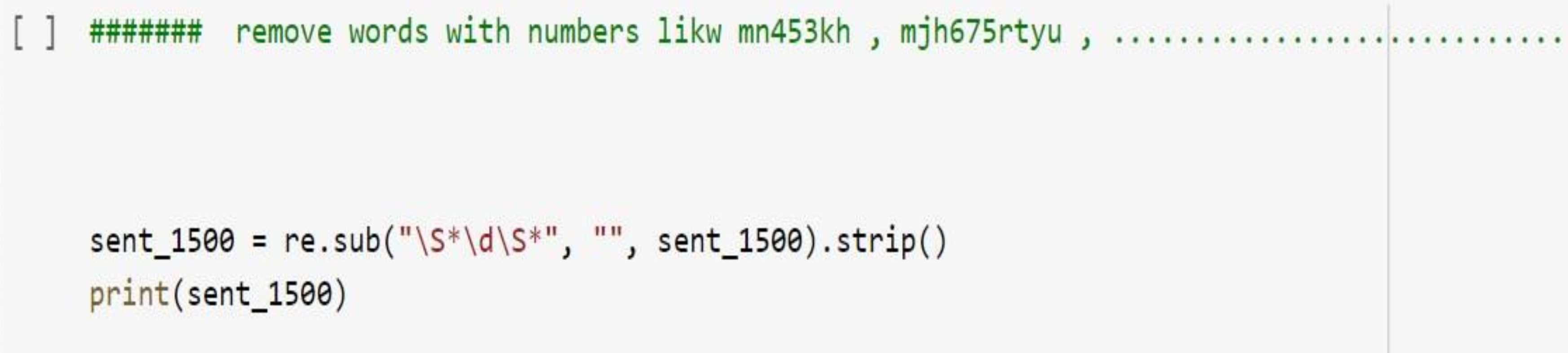
```
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)


soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)


soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)
```

- 2. Removal of punctuation and limited set of special characters



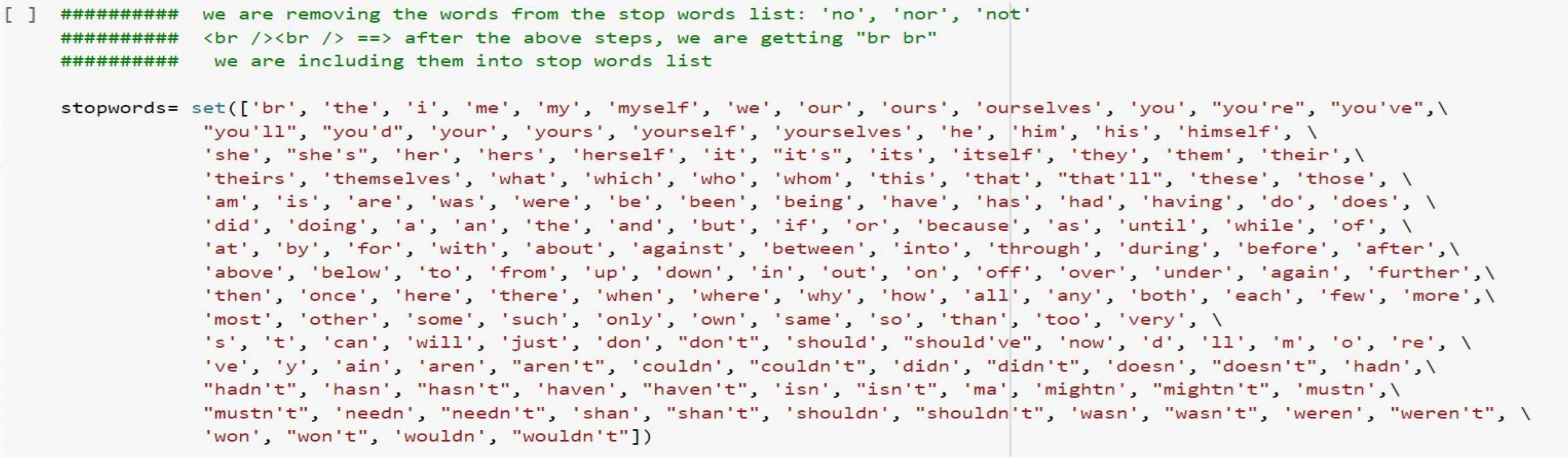
```
[ ]  ########## ---------  remove special character ----------------------



     sent_1000 = re.sub('[^A-Za-z0-9]+', ' ', sent_1000)
     print(sent_1000)
```

- 5. Removal of words having numbers



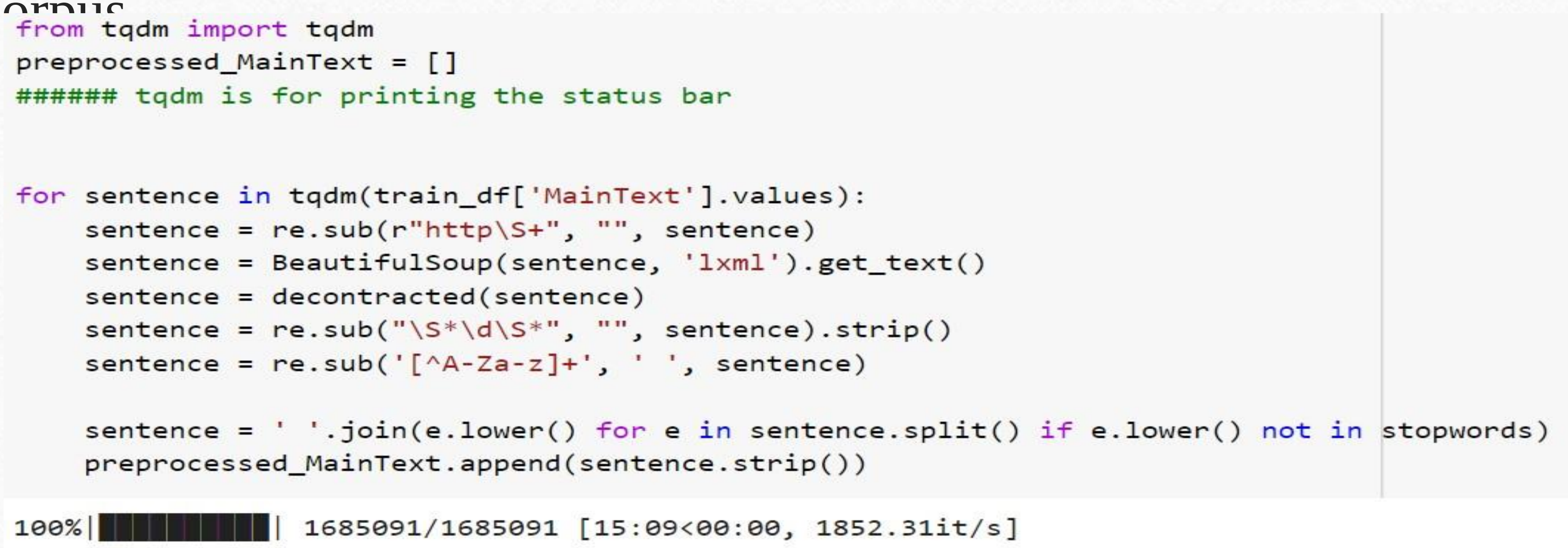
```
#######  remove words with numbers likw mn453kh , mjh675rtyu , .........................



sent_1500 = re.sub("\S*\d\S*", "", sent_1500).strip()
print(sent_1500)
```

## 6. Stopwords removal ( We are modifying Stopwords set as per need ), like removal of not and inclusion of br



```
[ ]  ##########  we are removing the words from the stop words list: 'no', 'nor', 'not'
     ##########  <br /><br /> ==> after the above steps, we are getting "br br"
     ##########   we are including them into stop words list

     stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
                 "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
                 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
                 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
                 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
                 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
                 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
                 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
                 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
                 'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
                 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
                 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
                 "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
                 "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
                 'won', "won't", 'wouldn', "wouldn't"])
```

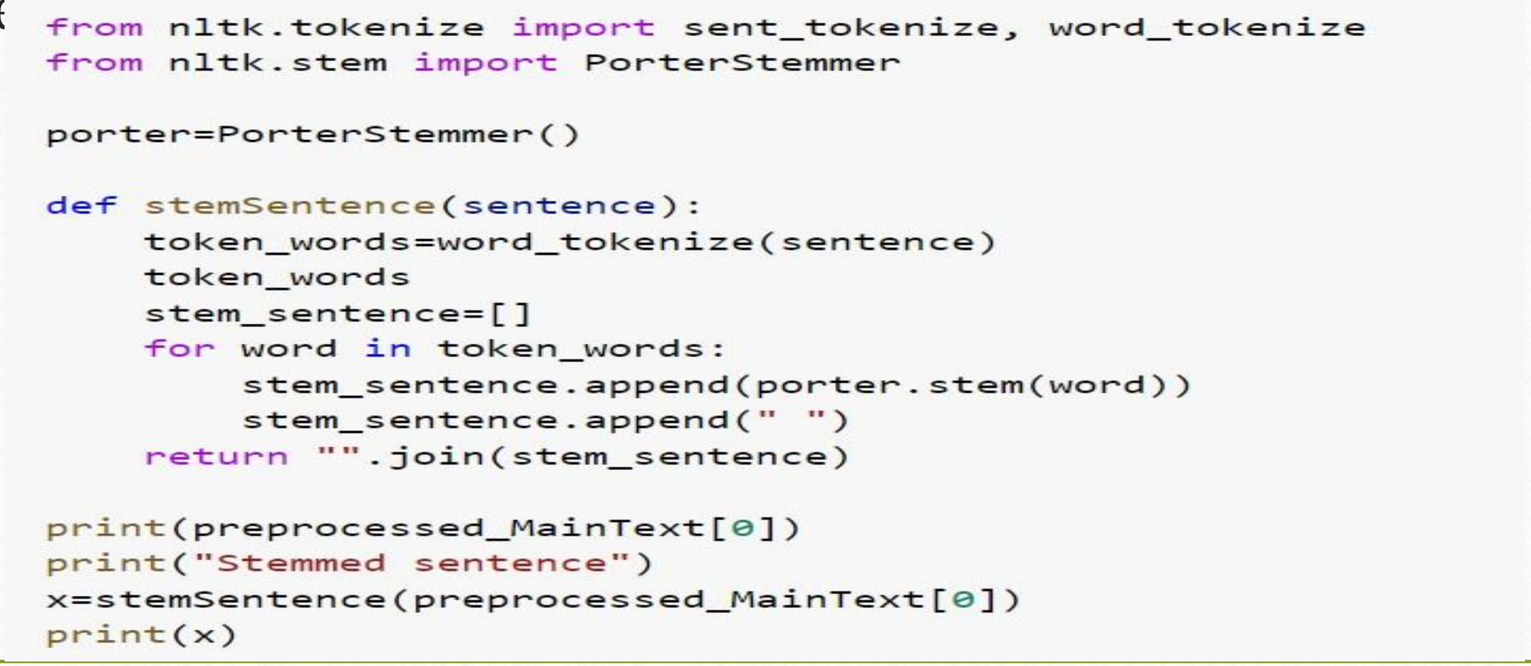
- 7. Atlast we are combining every preprocessing on all rows of data or corpus



```
from tqdm import tqdm
preprocessed_MainText = []
###### tqdm is for printing the status bar


for sentence in tqdm(train_df['MainText'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)

    sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
    preprocessed_MainText.append(sentence.strip())
```

```
100%|██████████| 1685091/1685091 [15:09<00:00, 1852.31it/s]
```

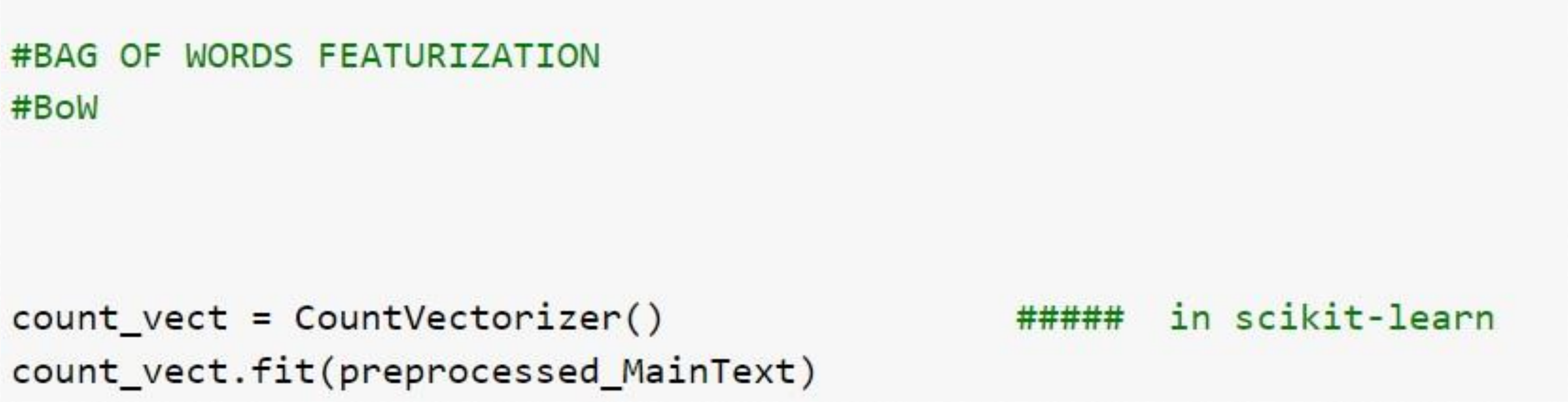
- 7. Stemming using SnowBall Stemme



```python
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import PorterStemmer

porter=PorterStemmer()

def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    token_words
    stem_sentence=[]
    for word in token_words:
        stem_sentence.append(porter.stem(word))
        stem_sentence.append(" ")
    return "".join(stem_sentence)

print(preprocessed_MainText[0])
print("Stemmed sentence")
x=stemSentence(preprocessed_MainText[0])
print(x)
```

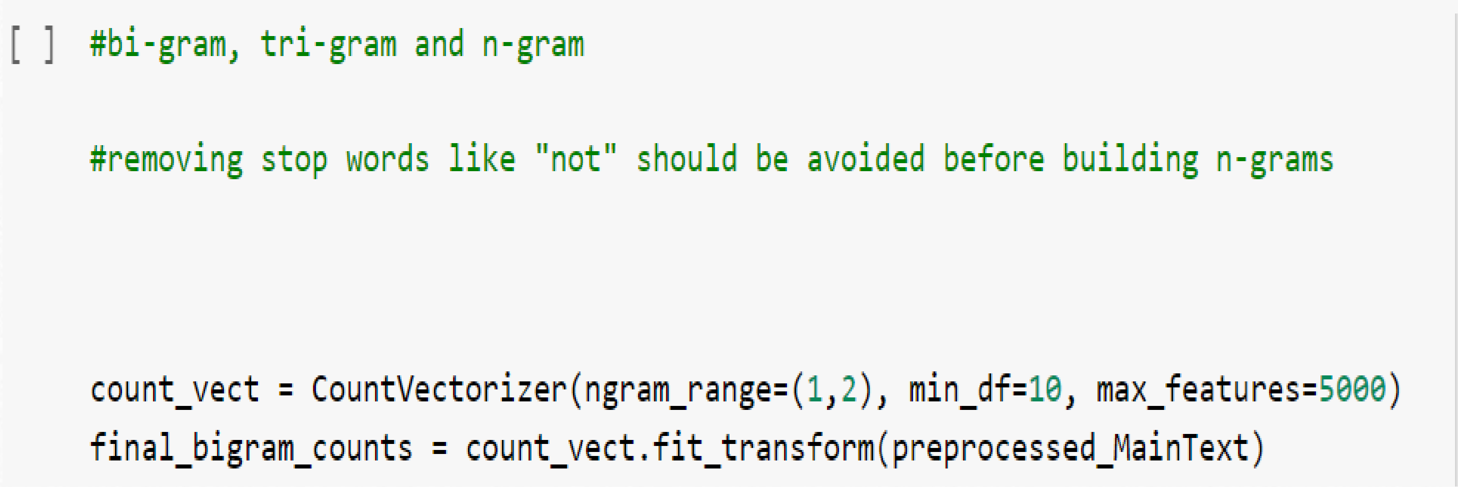
# Featurization

- Bag of words: Conversion of text into vector



```
#BAG OF WORDS FEATURIZATION
#BoW



count_vect = CountVectorizer()                    #####  in scikit-learn
count_vect.fit(preprocessed_MainText)
```

- Bi-gram

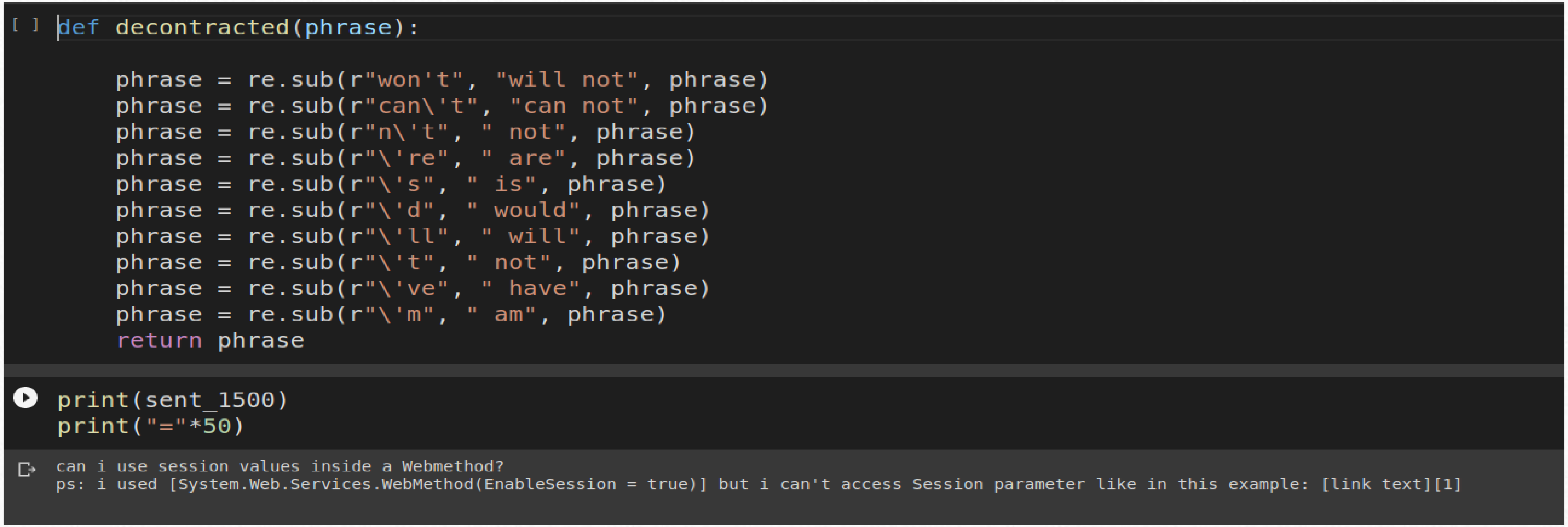


```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams



count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_MainText)
```

FINAL EVALUATION -By Team LORD OF RINGS

Preprocessing :-

1.Decontraction of words

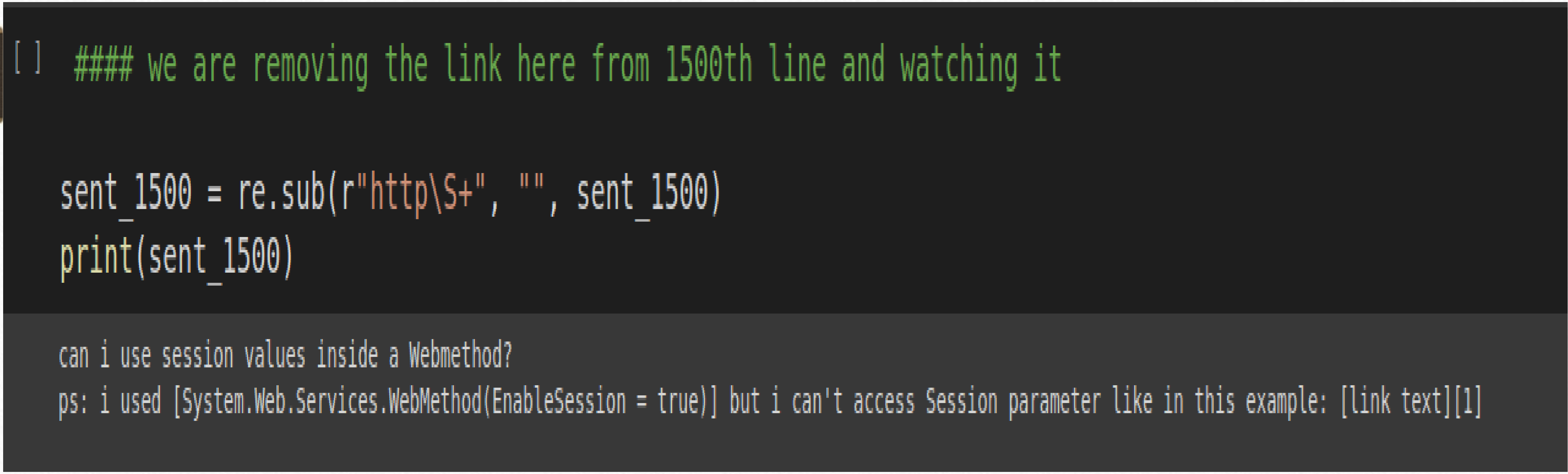


```
def decontracted(phrase):

    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```
print(sent_1500)
print("="*50)
```

```
can i use session values inside a Webmethod?
ps: i used [System.Web.Services.WebMethod(EnableSession = true)] but i can't access Session parameter like in this example: [link text][1]
```

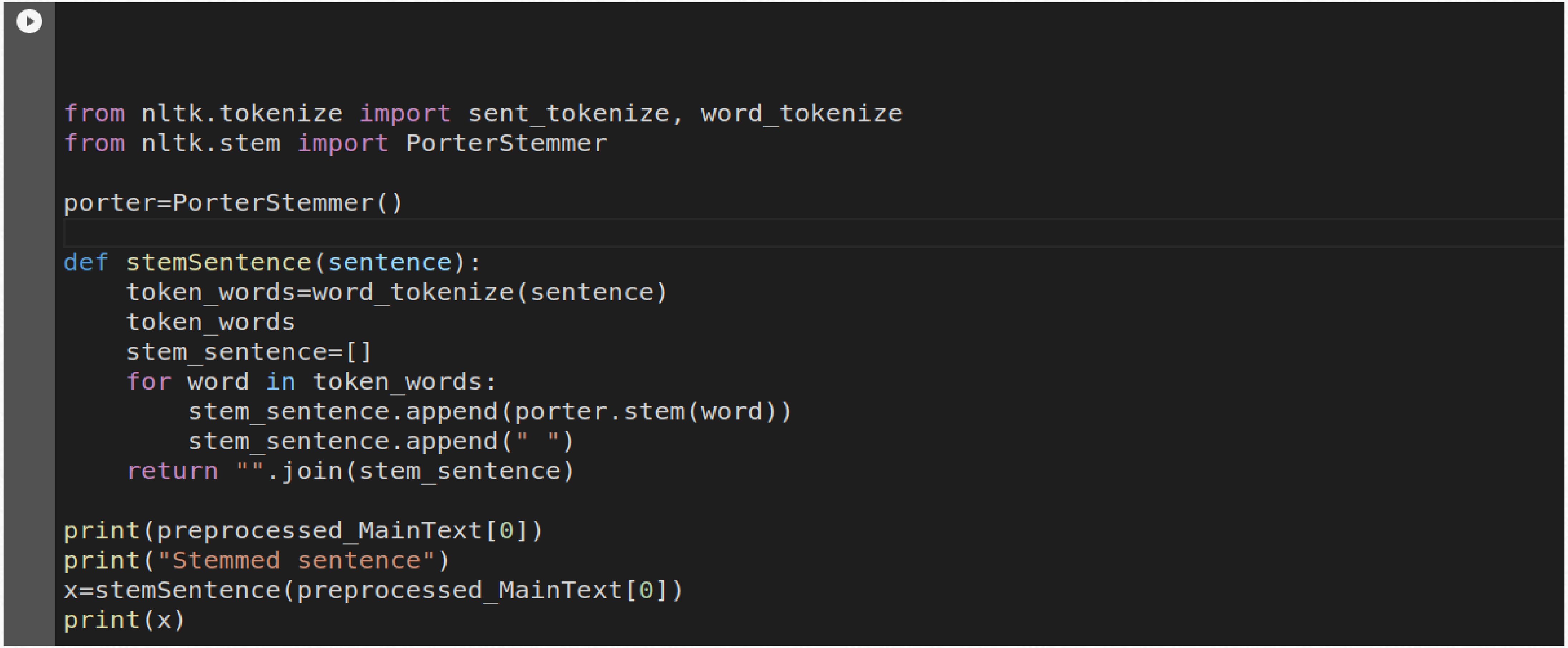
## 2.Link Removal from Data



```
#### we are removing the link here from 1500th line and watching it


sent_1500 = re.sub(r"http\S+", "", sent_1500)
print(sent_1500)
```

```
can i use session values inside a Webmethod?
ps: i used [System.Web.Services.WebMethod(EnableSession = true)] but i can't access Session parameter like in this example: [link text][1]
```

# 3. Stemming and lemmatization



```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import PorterStemmer

porter=PorterStemmer()

def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    token_words
    stem_sentence=[]
    for word in token_words:
        stem_sentence.append(porter.stem(word))
        stem_sentence.append(" ")
    return "".join(stem_sentence)

print(preprocessed_MainText[0])
print("Stemmed sentence")
x=stemSentence(preprocessed_MainText[0])
print(x)
```

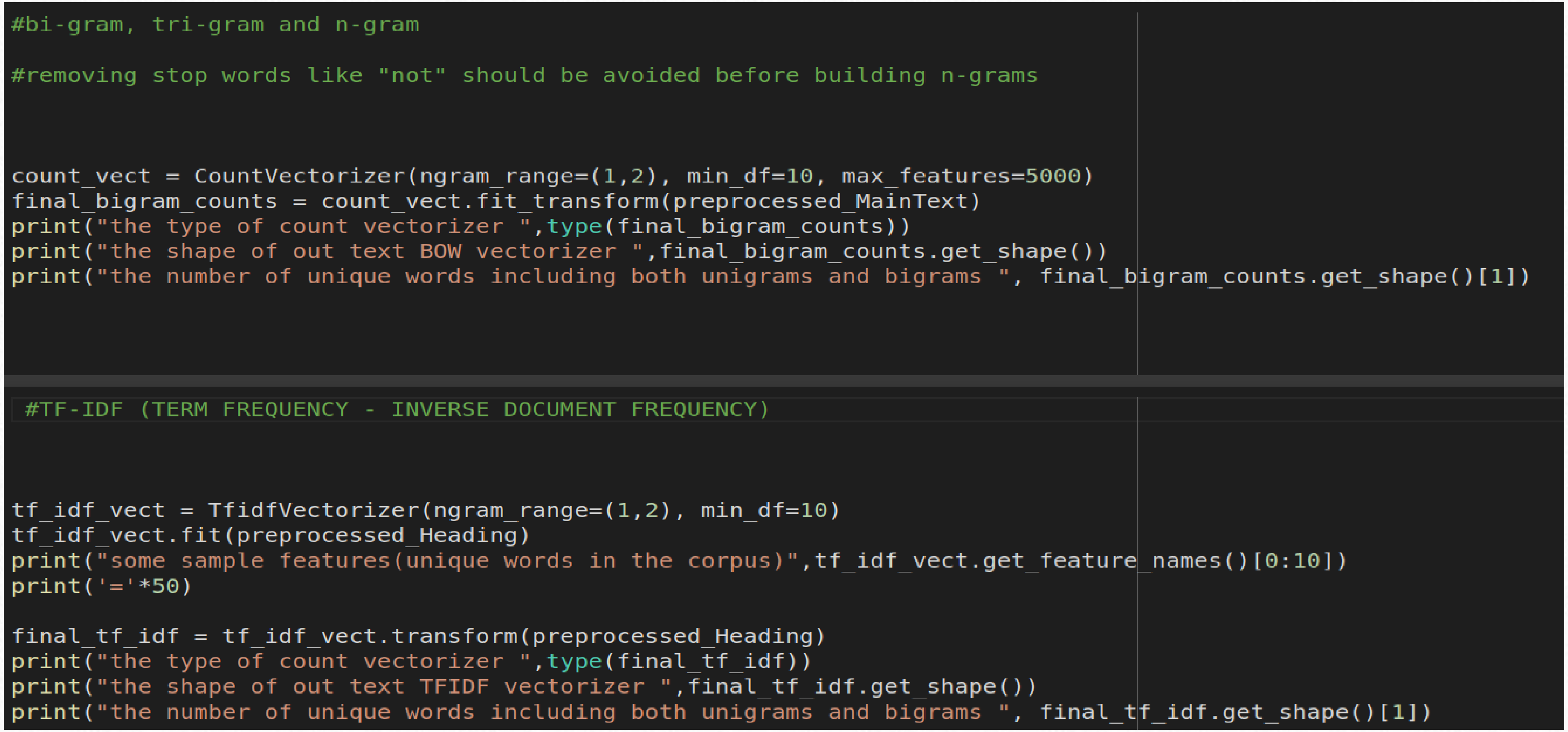
4.Adding up all the primary subjects ,secondary subjects ,teritary subjects and others  then done preprocessing over it.

5.Mostly we hav taken the consideration of Heading,Main Texts,Subjects for  training of our model.

# FEATURIZATION

1. Previously we have used “Bag of Words”(BOW) but this time we are using TF-IDF for Preprocessing our data as a featurization step.

2. We have also used the Bi-grams,Tri-grams,N-grams  as the featurization for the curiosity .



```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams



count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_MainText)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
 #TF-IDF (TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY)



tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_Heading)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_Heading)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1])
```

# MODEL LEARNING

1. we have used the Logistic Regression with OVR(One vs Rest) model for training of our model Then predicted our test data which performs well and gave around 20% accuracy

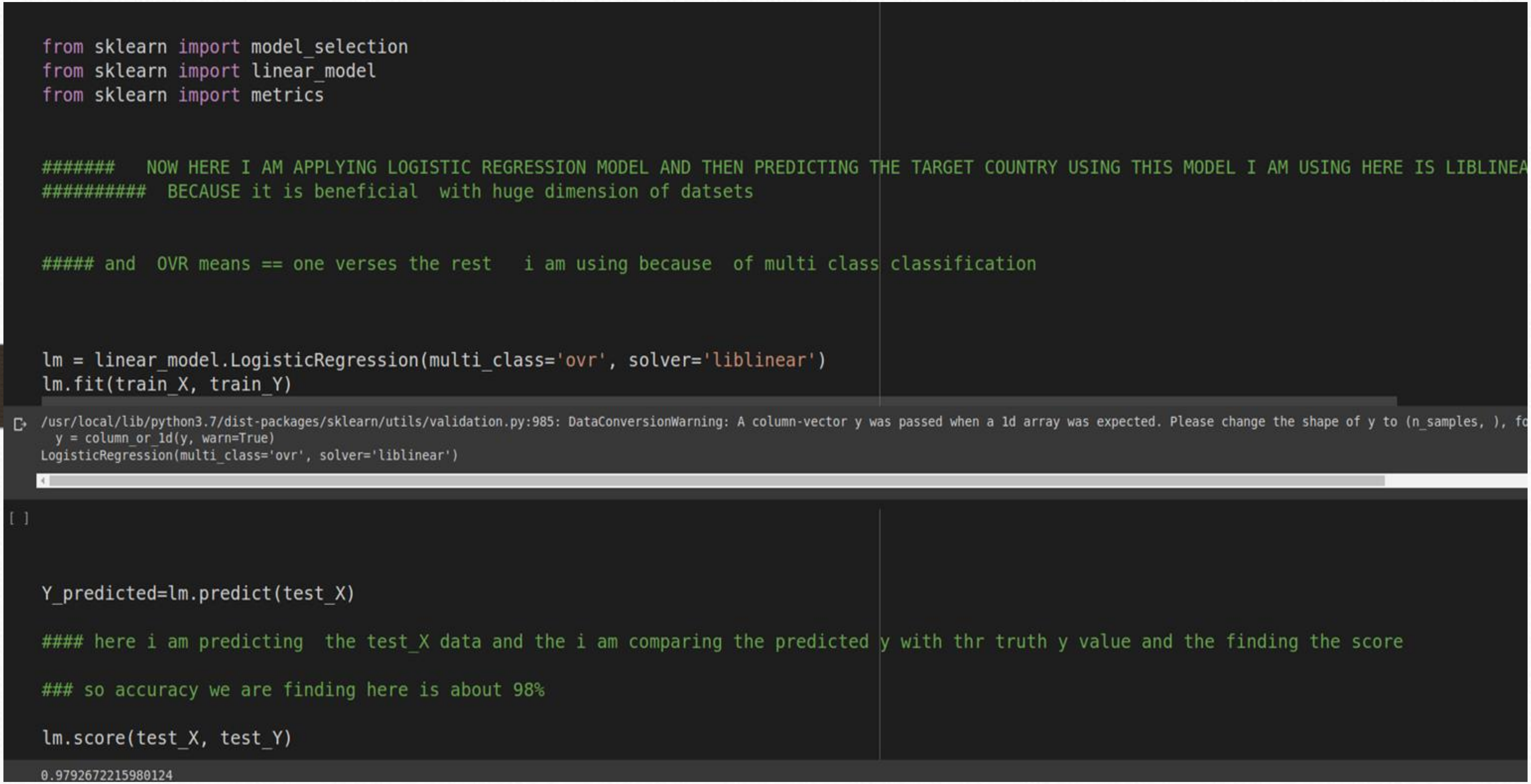
2. we have used the SVM using linear SVC which took so much of time like 30 minutes

3 After that we have used Decision Tree algorithm to train our model which goes exceptionally Well on our training data which gave around 23% accuracy on kaggle score board by Estimating the best Depth of the decision tree.

4.After that we have also used the Random forest to train our model but it was taking too much Of time even at lesser no of estimators

| Models Used | Score(Approx) |
|---|---|
| Logistic Regression( with OVR) | 20.2 |
| SVM Using Linear SVC | 19.8 |
| Decision Tree | 22.9 |
| Random Forest | 22.8 |

# DECISION TREE



```
from sklearn import model_selection
from sklearn import linear_model
from sklearn import metrics


#######   NOW HERE I AM APPLYING LOGISTIC REGRESSION MODEL AND THEN PREDICTING THE TARGET COUNTRY USING THIS MODEL I AM USING HERE IS LIBLINEA
##########  BECAUSE it is beneficial  with huge dimension of datsets


##### and  OVR means == one verses the rest   i am using because  of multi class classification



lm = linear_model.LogisticRegression(multi_class='ovr', solver='liblinear')
lm.fit(train_X, train_Y)
```

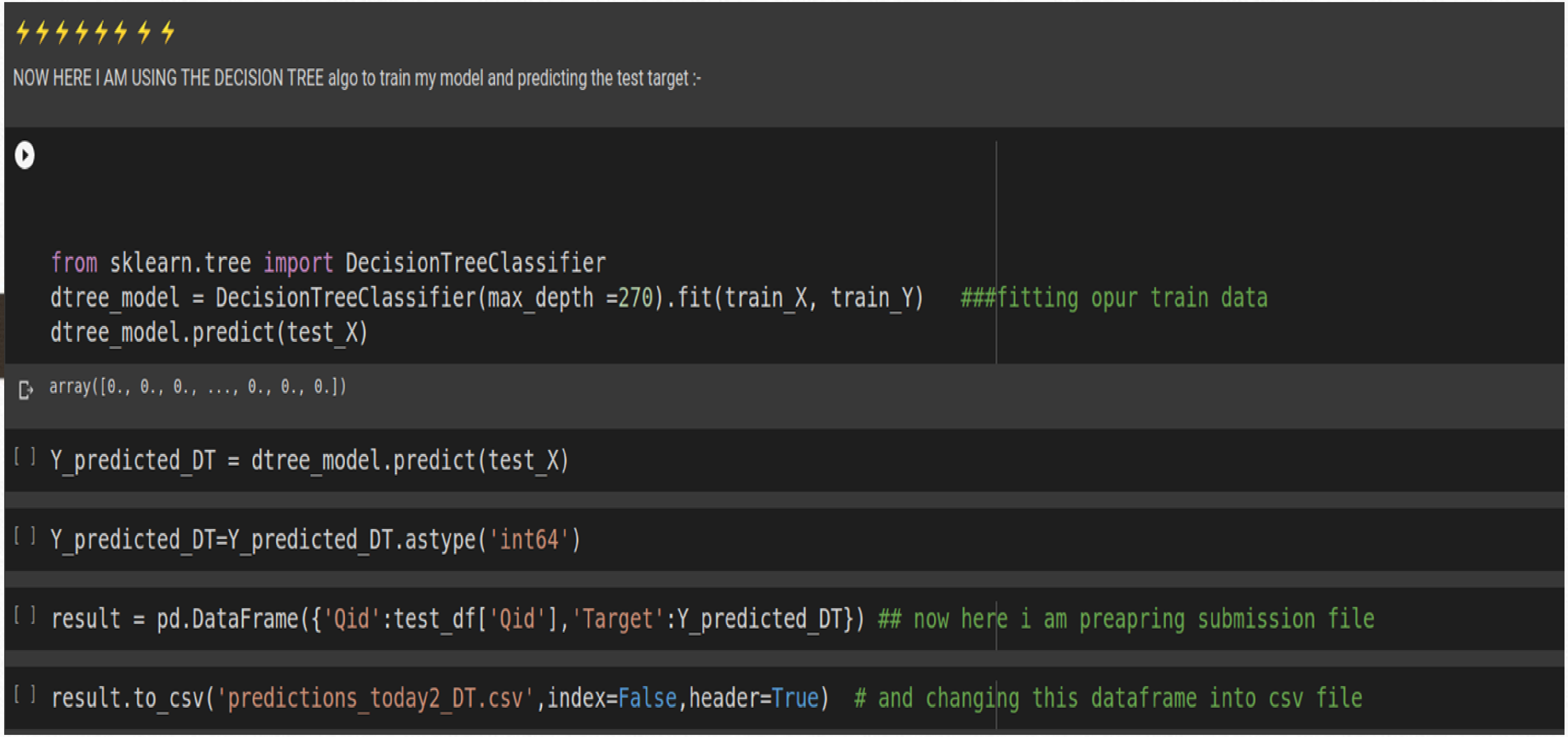
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:985: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), fo
  y = column_or_1d(y, warn=True)
LogisticRegression(multi_class='ovr', solver='liblinear')
```

[ ]

```
Y_predicted=lm.predict(test_X)

#### here i am predicting  the test_X data and the i am comparing the predicted y with thr truth y value and the finding the score

### so accuracy we are finding here is about 98%

lm.score(test_X, test_Y)
```

```
0.9792672215980124
```

# DECISION TREE



⚡⚡⚡⚡⚡⚡⚡⚡

NOW HERE I AM USING THE DECISION TREE algo to train my model and predicting the test target :-

```
from sklearn.tree import DecisionTreeClassifier
dtree_model = DecisionTreeClassifier(max_depth =270).fit(train_X, train_Y)   ###fitting opur train data
dtree_model.predict(test_X)
```

```
array([0., 0., 0., ..., 0., 0., 0.])
```

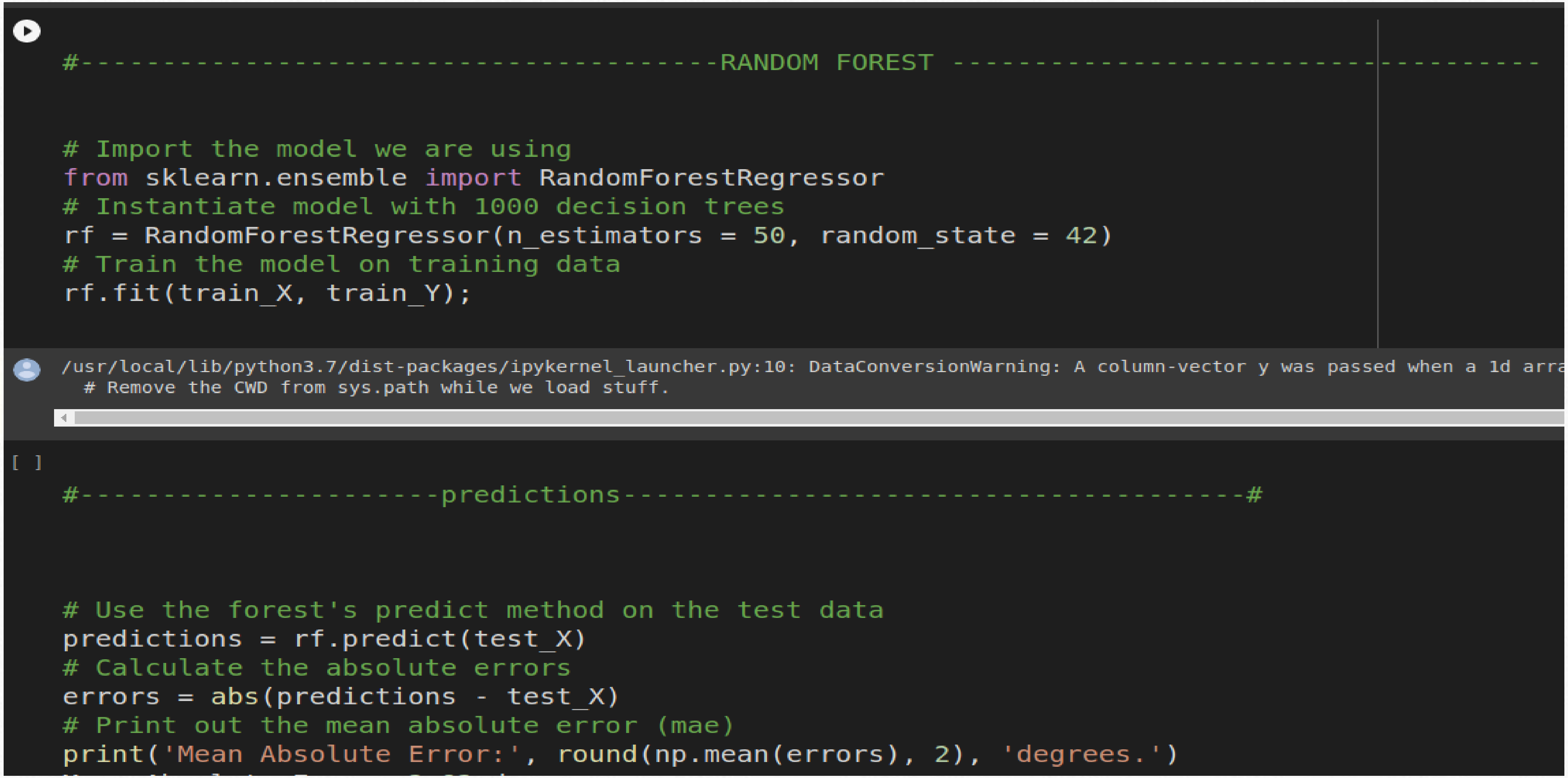
```
Y_predicted_DT = dtree_model.predict(test_X)
```

```
Y_predicted_DT=Y_predicted_DT.astype('int64')
```

```
result = pd.DataFrame({'Qid':test_df['Qid'],'Target':Y_predicted_DT}) ## now here i am preapring submission file
```

```
result.to_csv('predictions_today2_DT.csv',index=False,header=True)  # and changing this dataframe into csv file
```

# RANDOM FOREST

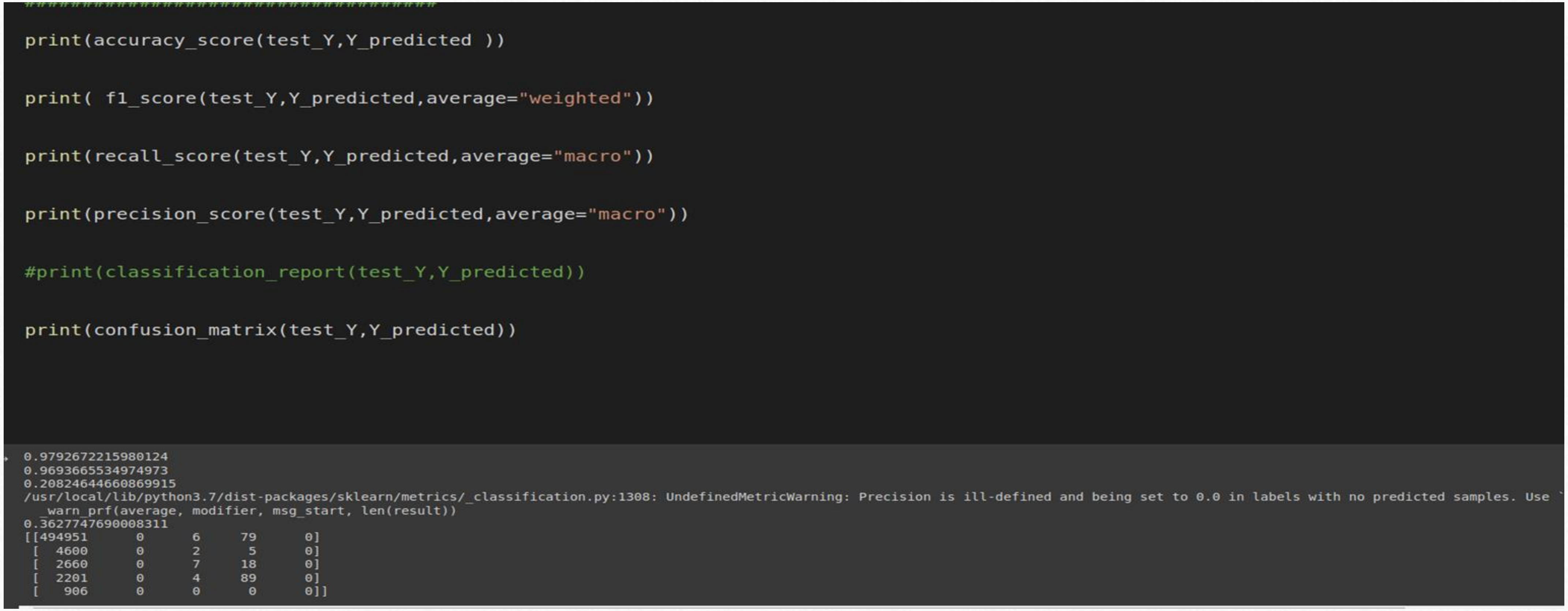


```
#---------------------------------------RANDOM FOREST ------------------------------------


# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 50, random_state = 42)
# Train the model on training data
rf.fit(train_X, train_Y);
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: DataConversionWarning: A column-vector y was passed when a 1d arra
  # Remove the CWD from sys.path while we load stuff.
```

```
#----------------------predictions--------------------------------------#



# Use the forest's predict method on the test data
predictions = rf.predict(test_X)
# Calculate the absolute errors
errors = abs(predictions - test_X)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

# METRICS



```
########################################

print(accuracy_score(test_Y,Y_predicted ))

print( f1_score(test_Y,Y_predicted,average="weighted"))

print(recall_score(test_Y,Y_predicted,average="macro"))

print(precision_score(test_Y,Y_predicted,average="macro"))

#print(classification_report(test_Y,Y_predicted))

print(confusion_matrix(test_Y,Y_predicted))
```

```
0.9792672215980124
0.9693665534974973
0.20824644660869915
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `
  _warn_prf(average, modifier, msg_start, len(result))
0.3627747690008311
[[494951      0      6     79      0]
 [  4600      0      2      5      0]
 [  2660      0      7     18      0]
 [  2201      0      4     89      0]
 [   906      0      0      0      0]]
```

# What More?

- We can use Emsemble Learning methods in future which comprises of BAGGING, BOOTSTRAPING AND AGGREGATING.

# REFERENCES

- https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089
- https://towardsdatascience.com/text-vectorization-bag-of-words-bow-441d1bfce897
- https://towardsdatascience.com/build-and-compare-3-models-nlp-sentiment-prediction-67320979de61