Fake News Detection Project

Submitted By:

Khushboo Khatri

ACKNOWLEDGMENT

I would like to appreciate FlipRobo for giving us the opportunity to work on such exciting project. It helps us to broaden our portfolio and give us the chance to work on different fields of machine learning.

I am grateful for DataTrained which give us the knowledge about Datascience and Machine Learning without which it will be nearly impossible to deal with such Projects. Their technical team really help us to understand every concept and support us with our doubts.

Last but not the least I would like to show my gratitude towards our SME Shwetank Mishra, for helping us in every possible way.

I would also acknowledge the help taken from various websites like stackflow.com, youtube.com, geekforgeek.com.

INTRODUCTION

- ## Business Problem Framing:

  Fake news has become one of the biggest problems of our age. It has serious impact on our online as well as offline discourse. One can even go as far as saying that, to date, fake news poses a clear and present danger to western democracy and stability of the society. So, our objective is to build a model which helps us to classify/ Identify Fake News from Real News.

- ## Conceptual Background of the Domain Problem:

  What is Fake News?

  Fake news's simple meaning is to incorporate information that leads people to the wrong path. Nowadays fake news spreading like water and people share this information without verifying it. This is often done to further or impose certain ideas and is often achieved with political agendas.

  For media outlets, the ability to attract viewers to their websites is necessary to generate online advertising revenue. So, it is necessary to detect fake news.

- ## Review of Literature:

  In this fake news problem, we need to apply Natural Processing Language algorithm. In this project, we are using some machine learning and Natural language processing libraries like NLTK, re (Regular Expression), Scikit Learn.

  Natural Language Processing

  Machine learning data only works with numerical features so we have to convert text data into numerical columns. So we have to preprocess the text and that is called natural language processing.

In-text preprocess we are cleaning our text by steaming, lemmatization, remove stopwords, remove special symbols and numbers, etc. After cleaning the data, we have to feed this text data into a vectorizer which will convert this text data into numerical features.

- Motivation for the Problem Undertaken:
  We did this project as it helps in understanding the concept of NLP, also Fake News is a widely spread cause of untrust and certain ideas floats in the society as a result of fake news only.
  It is a really interesting project as here we deal with both the news text and news title to detect the fake news. This project will help to verify the authenticity of the news and hence serves helpful to the business as well.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem
  We got two different dataset one consist of fake news and other dataset consist of real/True news.
  We add a column name Label consisting of single value Fake /Real respectively to both the dataset.

```
#adding label column to the true news dataframe
df_true['Label']= 'real'


# chceking the dataframe
df_true.head(3)
```

	title	text	subject	date	Label
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017	real
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017	real
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017	real

```
# adding label to fakenews dataframe
df_fake['Label'] = 'fake'

# checking the dataframe
df_fake.head(3)
```

	title	text	subject	date	Label
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017	fake

After this we merge the dataset using simple panda function.

```
# we can see that both the dataframe has same columns, let's merge them to make single dataframe,
# this single dataframe consist of both true and fake news. this will be our final dataframe.

df_final= df_true.append(df_fake).sample(frac=1).reset_index().drop(columns=['index'])


df_final.tail(3)
```

:

	title	text	subject	date	Label
44895	Brexit transition deal will require new UK leg...	LONDON (Reuters) - The British government is p...	worldnews	October 26, 2017	real
44896	Carter blasts Trump's ban on transgender, says...	WASHINGTON (Reuters) - Ash Carter, who was U.S...	politicsNews	July 26, 2017	real
44897	Highlights: The Trump presidency on March 31 a...	(Reuters) - Highlights of the day for U.S. Pre...	politicsNews	March 31, 2017	real

Here we randomly merge the two dataframe and resetting their index.

```
# getting dimension of final dataframe
df_final.shape
```

: (44898, 5)

From here we can see that we have 44898 data which consist of both fake and real news, we will use this data to build our model. The primary feature here is title and text, where title news the news headline while text denotes the content of the news. subject column consists of general topic on which the news is on. And date denotes the release date of news.
The label column denotes whether the news is fake or real. This is our final dataset, on which we need to perform all the model building approach.
Let's start with statistical analysis of data.

```
# statistical analysis of data
df_final.describe(include= 'object')
```

:

	title	text	subject	date	Label
count	44898	44898	44898	44898	44898
unique	38729	38646	8	2397	2
top	Factbox: Trump fills top jobs for his administ...		politicsNews	December 20, 2017	fake
freq	14	627	11272	182	23481

From here it is visible that we have a very wide range of date i.e., near about 2397 unique date count is there. The subject column consists of 8 unique values. Label column which is our target column has two unique value which clearly indicates that it is a

classification problem. The title and text column is our main text features, in which NLP pre-processing need to be perform.

- Data Sources and their formats:

  We got this dataset from our internship, which suggest that this dataset is downloaded from Kaggle. There are two datasets one for fake news and one for true news. In true news, there is 21417 news, and in fake news, there is 23481 news. You have to insert one label column zero for fake news and one for true news. We are combined both datasets using pandas built-in function. And here is the general summary of our final dataset.

```
# Determining the general summary of the final datset.

df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44898 entries, 0 to 44897
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   title    44898 non-null  object
 1   text     44898 non-null  object
 2   subject  44898 non-null  object
 3   date     44898 non-null  object
 4   Label    44898 non-null  object
dtypes: object(5)
memory usage: 1.7+ MB
```

From here we can see that, in our dataset we don't have any null values, as all 44898 entries are non null. Each column including the label column is object data type.

For title and text column we need to perform NLP pre processing technique. We can easily encode our label column. for Subject column we need to check and then do encoding accordingly. For date column we either convert it into proper numerical column, to use it in our dataset.

- Data Pre-processing Done:

  In this step, we explore data and gain insights such as shape, accuracy, and initial patterns in the data.

First, we check the information of the given dataset and extract information about the dataset. Here in our dataset it consist of 44898 rows and 5 columns. Each column is of Object datatype including the Label column which is our target column.
The main feature column here is text & title which are basically sentence columns on which we need to apply NLP pre processing techniques.
We tried to perform feature extraction on date column, and plot some graphs and later realise that, there is no direct connection between date and Label column, so later we drop it.
One more thing which we realise that subject column also don't show much of connection between fake/real news.

Let's starts with text and title column pre processing. Following are the steps involve in pre processing our features:

1. Data Cleaning: We check for duplicates in the dataset and drop them in order to get clean data.

```
# Checking for duplicated data
df_final.duplicated().sum()
```

```
: 213
```

```
# dropping duplicates
data= df_final.drop_duplicates()

data.shape
```

```
]: (44685, 4)
```

2. Now let's start with actual text: Here we perform tokenization, we build a function which put our text into proper English form.

```
#expanding english language contractions:
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", "s", 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)
    phrase = re.sub(r"\'u", " you", phrase)

    return phrase
```

3. Then we define stop words and lemmatization.

```
# defining stop words and lemmatizer
stop_words=set(stopwords.words('english')+ ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure', 'dun'])
lemmatizer= WordNetLemmatizer()
```

4. Convert entire text into lower case.

```
# Convert to lowercase

data['text'] = data['text'].apply(lambda x: x.lower())
data['title'] = data['title'].apply(lambda x: x.lower())
data.head(2)
```

We are performing similar steps on both title and text column.

5. Then we remove punctuation and stop words from the text and title column.

```
# remove punctuation
def punctuation_removal(text):
    all_list = [char for char in text if char not in string.punctuation]
    clean_str = ''.join(all_list)
    return clean_str

data['text'] = data['text'].apply(punctuation_removal)
data['title'] = data['title'].apply(punctuation_removal)
```

```
# removing stopwords
data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
data['title'] = data['title'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))

data.head(3)
```

6. Then we pass our data into a loop in which we remove all the hyperlinks, email address, special characters, stopword if any still present and then we finally lemmatize our text to give a fully pre processed text.

```
preprocessed_text=[]
from tqdm import tqdm
#tqdm is for printing the status bar
for sentance in tqdm(data['text'].values):
    sent= decontracted(sentance)
    sent = re.sub(r'https?:\/\/.*[\r\n]*', '', sent)    #remove hyperlinks
    sent = re.sub(r'^.+@[^\.].*\.[a-z]{2,}$','', sent)    # remove emails
    sent = re.sub(r'[^\w\d\s]', '',sent )      # removing punctuation if any still present
    sent = re.sub('[^A-Za-z]+', ' ', sent)      #remove special characters, numbers

    sent = ' '.join(lemmatizer.lemmatize(e) for e in sent.split())   # lemmatization
    preprocessed_text.append(sent.lower().strip()) # converting the text to lower case
```

```
100%|██████████| 44685/44685 [01:04<00:00, 692.17it/s]
```

This is our clean data, we perform similar operation on title column as well. We are using tqdm to print the status bar.

```
# similar cleaning technique for title as well
preprocessed_title= []
for sentance in tqdm(data['title'].values):
    sent= decontracted(sentance)
    sent = re.sub(r'https?:\/\/.*[\r\n]*', '', sent)    #remove hyperlinks
    sent = re.sub(r'^.+@[^\.].*\.[a-z]{2,}$','', sent)    # remove emails
    sent = re.sub(r'[^\w\d\s]', '',sent )      # removing punctuation if any still present
    sent = re.sub('[^A-Za-z]+', ' ', sent)      #remove special characters, numbers

    sent = ' '.join(lemmatizer.lemmatize(e) for e in sent.split())   # lemmatization
    preprocessed_title.append(sent.lower().strip()) # converting the text to lower case
```

```
100%|██████████| 44685/44685 [00:03<00:00, 12162.01it/s]
```

7. As our data is pre processed, now we assign them back to our dataset.

```
data['title']= preprocessed_title
data['text']= preprocessed_text     # assigning clean value to the dataset

# chceking dataset
data.head(3)
```

]:

	title	text	subject	Label
0	breaking obama veto bill unanimously passed co...	come surprise one yesterday th anniversary wor...	politicsNews	fake
1	democrat liberal medium uncovered agreement ve...	wikileaks email incredible study political des...	politicsNews	fake
2	bill maher light joint get real pot legalizati...	bill maher hardcore supporter legalization mar...	worldnews	fake

Now we can move ahead with our pre processed dataset.

8. We encode our pre processed data, in terms of NLP we vectorized our pre processed data using TFIDF vectorizer. And our Label column by simple mapping method while subject by label encoder.

```
# we need to encode our subject column and Label column
# Using Label Encoder for subject column, as we don't want to increase the number of columns
from sklearn.preprocessing import LabelEncoder
lab_enc= LabelEncoder()

data['subject']= lab_enc.fit_transform(data['subject'])


# encoding Label column
data['Label']= data['Label'].map({'real': 0, 'fake': 1})

# chceking the dataset
data.head(3)
```

9. Before performing tfidf vectorization we split the data into three set train, cv, and test dataset in order to improve accuracy and avoiding data leakage.

```
#importing library

from sklearn.model_selection import train_test_split
```

```
# Separating Features and Label.
x= data.drop('Label', axis=1)     #feature
output= data.Label              #label
```

```
# Splitting data into three set
train,test,train_output,test_output= train_test_split(x,output,test_size=0.3, random_state=0, stratify= output)


train_modified,cv,train_output_modified,cv_output = train_test_split(train, train_output,test_size=0.3,
                                                                     stratify= train_output,random_state=0)
```

Here we split dataset into three different set and will use the second split for cross validation. As this gives more accuracy and also improves the efficiency. here we use train data for training the dataset, test data for testing the dataset, and cv for cross validation purpose. We don't want any data leakage, that's the reason we are doing in this way.

10. Vectorization of text data, which is similar as encoding as it convert data into machine understandable form.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
#initiation of module
text_tfidf_vectorizer = TfidfVectorizer(min_df= 5)
```

```
# Encoding train text
train_text_tfidf= text_tfidf_vectorizer.fit_transform(train['text'].values)
```

```
# Encoding CV and test text
cv_text_tfidf= text_tfidf_vectorizer.transform(cv['text'].values)
test_text_tfidf= text_tfidf_vectorizer.transform(test['text'].values)
```

```
# perform similar vectorization for title also
train_title_tfidf= text_tfidf_vectorizer.transform(train['title'].values)
cv_title_tfidf= text_tfidf_vectorizer.transform(cv['title'].values)
test_title_tfidf = text_tfidf_vectorizer.transform(test['title'].values)
```

Using TFIDF vectorizer we don’t need to scale the data, as the final output of TFIDF vectorizer is already scaled, standardized data. This is our all prepared, encode data, which we can use in model building as it is.

11. Here we can see that text and title columns are vectorized separately, we need to join them to form single feature dataframe. For that we will use hstack.

```
from scipy.sparse import hstack
```

```
# combining TFIDF encoded features
train_data_final_tfidf= hstack((train_title_tfidf, train_text_tfidf))
cv_data_final_tfidf= hstack((cv_title_tfidf, cv_text_tfidf))
test_data_final_tfidf= hstack((test_title_tfidf, test_text_tfidf))
```

```
train_data_final_tfidf.shape, cv_data_final_tfidf.shape, test_data_final_tfidf.shape    #chceking the dimension after vectorization
```

```
3]: ((31279, 67080), (9384, 67080), (13406, 67080))
```

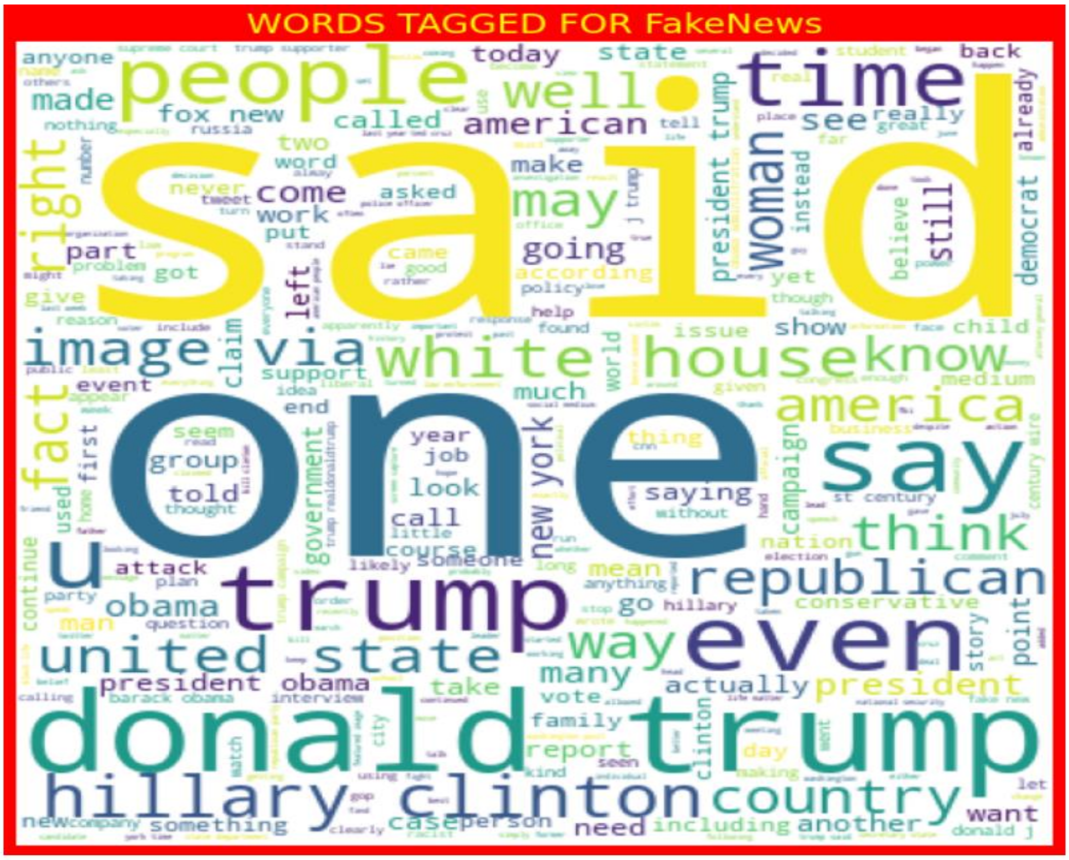
Now this will be our final train, test and cv feature data, which we will use in model building.

- Data Inputs- Logic- Output Relationships:

For our NLP project we plot wordcloud which shows us the relation between the words which appears most in that particular label class.

In our case we got two classes for our Label name Fake and Real respectively. It is also taken into consideration that title column and text column plays almost equal weightage for determining whether a news is fake or real, so we plot a graph individually keeping them into mind.

Let’s start with Fake News Text and Title. The large font words are those which appears more frequently in the text. They could be the determining factor.

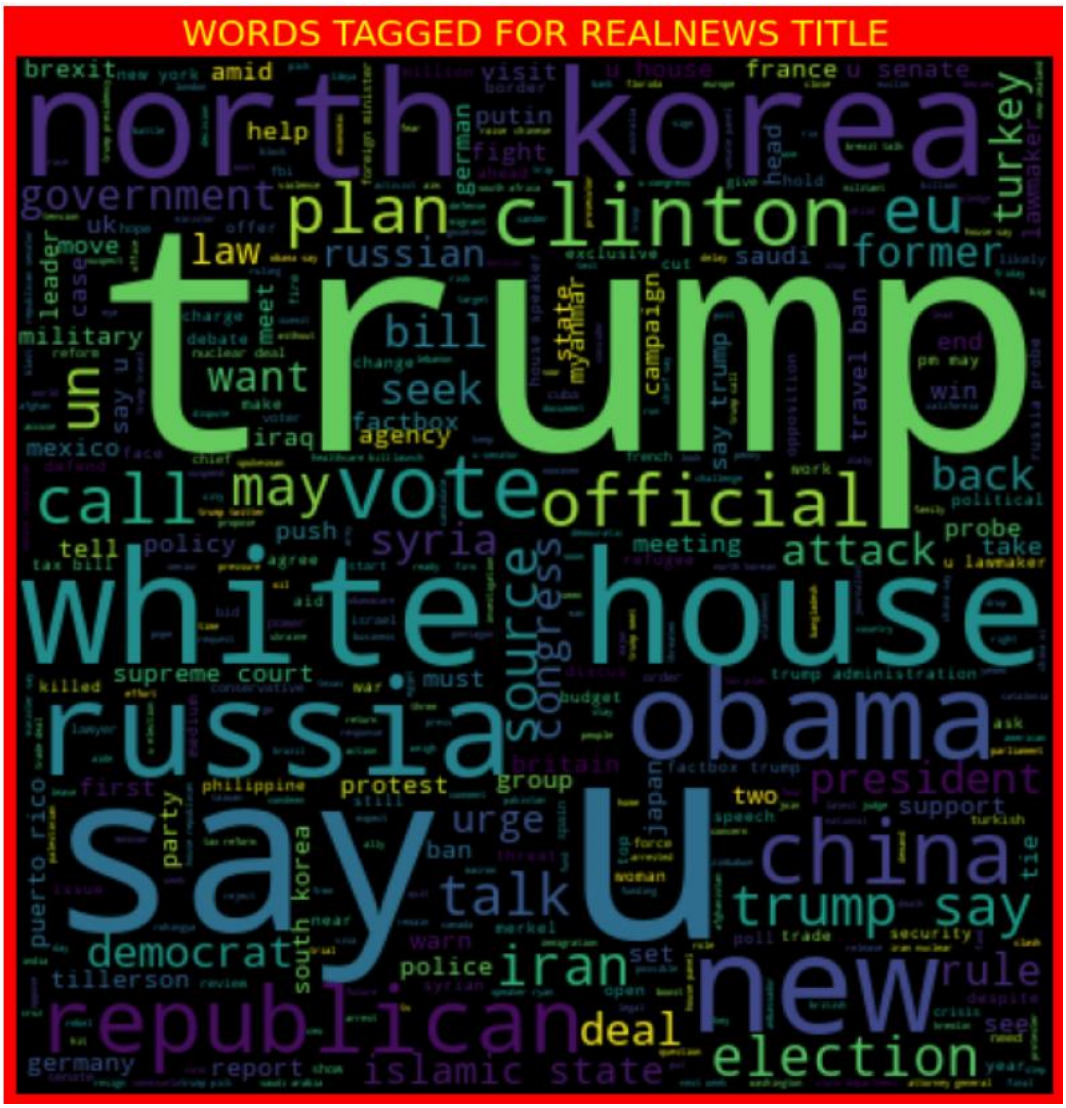


Key Observation: said, one, Donald trump, image, via etc are the most recurring words in the fake news text.

From here also we can observe that Obama, video, trump, Hillary, democrat etc are the most recurring words.

Now let's see what are the words we are frequent in Real/true news.

Key Observation: united states, Washington reuters, white house , said statement etc are the most common words in real news text. Let's see what title words are most common for real news.



Key observations: trump, white house, say, Russia, source, Obama, new etc are most common words found here.

- State the set of assumptions (if any) related to the problem under consideration

  We are assuming that the data provided to us is true. Also, in NLP problem we don't consider numerical columns, which is one of the reasons to drop date column from the dataset.

  We also assume that title and text column both has equal importance when we talk about model building to detect fake news.

- Hardware and Software Requirements and Tools Used:

  This machine learning project is a NLP problem, so we required all the NLP libraries along with basic python libraries.

1. Python
2. Pandas
3. Matplotlib.pyplot
4. Warnings
5. Numpy
6. Seaborn

   NLP Libraries
7. Re
8. String
9. Nltk
10. Stopwords
11. WordNetLemmatizer
12. Wordnet
13. Tqdm
14. Wordcloud
15. Tfidfvectorizer
16. Scikit

With there help we build an NLP classification project.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods):

  We use our vectorized data to build our final model. In our case Logistic Regression seems to be the best suited model.
  Although other model also performs equally well, but logistic regression considers to be best suited approach for our dataset with fractions of scores difference compared to others.

- Testing of Identified Approaches (Algorithms)

  Fake News Detection Project is basically a classification problem, so we follow following approaches with TFIDF vectorizer to get the best result.

  1. MultinomialNB
  2. Logistic Regression
  3. Random Forest classifier
  4. Decision tree classifier
  5. SVM
  6. XGBoost Classifier

- Run and Evaluate selected models

  For our dataset, we find Logistic Regression with TFIDF vectorization to be best suited approach. Till vectorization, we already define the approach. So, here we basically share the details of how we build our final model.

  We start with importing all the necessary library, then define a function to plot confusion matrix.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, f1_score, classification_report
```

```
# Function to plot the confusion matrix (code from https://scikit-learn.org/stable/auto_examples/model_se
from sklearn import metrics
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Then, we start with setting the parameter for Logistic Regression algorithm.

```
c_range=[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,10000]
for i in c_range:
    lr= LogisticRegression(C= i, max_iter=300)
    lr.fit(train_data_final_tfidf, train_output)

    train_prob= lr.predict_proba(train_data_final_tfidf)[:,1]
    train_AUC= roc_auc_score(train_output, train_prob)
    print("For  C=%f, train AUC =%f" % (i, train_AUC))

    cv_prob = lr.predict_proba(cv_data_final_tfidf)[:,1]
    cv_AUC = roc_auc_score(cv_output, cv_prob)
    print("For C =%f, cv AUC =%f" % (i, cv_AUC))

    train_scores = lr.predict(train_data_final_tfidf)
    train_f1 = f1_score(train_output, train_scores)
    print("For C =%f, train f1 score =%f" % (i, train_f1))

    cv_scores = lr.predict(cv_data_final_tfidf)
    cv_f1 = f1_score(cv_output, cv_scores)
    print("For C =%f, cv f1 score =%f" % (i, cv_f1))

    print("-"*50)
```

```
For  C=0.000100, train AUC =0.988425
For C =0.000100, cv AUC =0.988020
For C =0.000100, train f1 score =0.688869
For C =0.000100, cv f1 score =0.688836
--------------------------------------------------
For  C=0.001000, train AUC =0.989295
For C =0.001000, cv AUC =0.988877
For C =0.001000, train f1 score =0.828650
For C =0.001000, cv f1 score =0.828051
--------------------------------------------------
For  C=0.010000, train AUC =0.993825
For C =0.010000, cv AUC =0.993298
For C =0.010000, train f1 score =0.967956
For C =0.010000, cv f1 score =0.966957
--------------------------------------------------
For  C=0.100000, train AUC =0.998729
For C =0.100000, cv AUC =0.998309
For C =0.100000, train f1 score =0.986535
For C =0.100000, cv f1 score =0.985230
--------------------------------------------------
For  C=1.000000, train AUC =0.999955
For C =1.000000, cv AUC =0.999916
For C =1.000000, train f1 score =0.997686
For C =1.000000, cv f1 score =0.997156
--------------------------------------------------
For  C=10.000000, train AUC =1.000000
For C =10.000000, cv AUC =1.000000
For C =10.000000, train f1 score =0.999970
For C =10.000000, cv f1 score =1.000000
--------------------------------------------------
For  C=100.000000, train AUC =1.000000
For C =100.000000, cv AUC =1.000000
For C =100.000000, train f1 score =1.000000
For C =100.000000, cv f1 score =1.000000
```

We can observe that the value of C = 100 and onwards we are getting perfect 1 score for both AUC and F1. So, let's take C =100 for test data.

```
lr_clf= LogisticRegression(C= 100, max_iter=300)
lr_clf.fit(train_data_final_tfidf, train_output)

test_prob= lr_clf.predict_proba(test_data_final_tfidf)[:,1]
test_AUC_lr=roc_auc_score(test_output, test_prob)
print("For C=1, test AUC = %f" % (test_AUC_lr))

test_scores=lr_clf.predict(test_data_final_tfidf)
test_f1_lr= f1_score(test_output, test_scores)
print("For C=1, test f1 scores =%f" % (test_f1_lr))
```

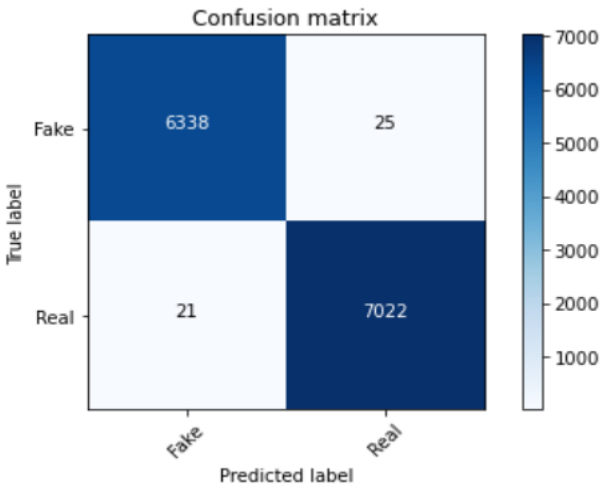
```
For C=1, test AUC = 0.999818
For C=1, test f1 scores =0.996735
```

```
# classification report
print("Classification Report For Logistic regression Model\n")
print(classification_report(test_output, test_scores))
```

```
Classification Report For Logistic regression Model

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6363
           1       1.00      1.00      1.00      7043

    accuracy                           1.00     13406
   macro avg       1.00      1.00      1.00     13406
weighted avg       1.00      1.00      1.00     13406
```

```
# confusion matrix
cm = metrics.confusion_matrix(test_output, test_scores)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

```
Confusion matrix, without normalization
```



Logistic regression seems to be the best fitted model so far with AUC score of 0.999 and f1 score of 0.995. Even the confusion matrix shows good result for this model.

- Key Metrics for success in solving problem under consideration

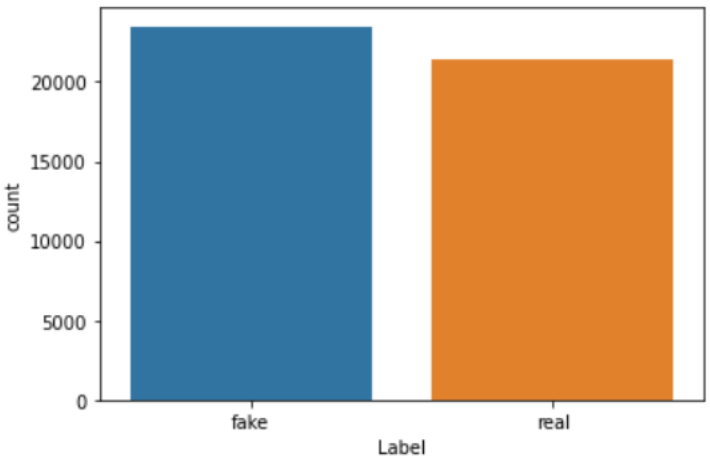
  For this model we consider roc_auc_score as our primary metrics and considering f1 score as secondary metrics.
  We also get classification report for each model to get a clear picture of how well the model is performing
  The confusion matrix is plus in the performance category.

- Visualizations

  Following are the graphs which we plot for EDA and visualization purpose.

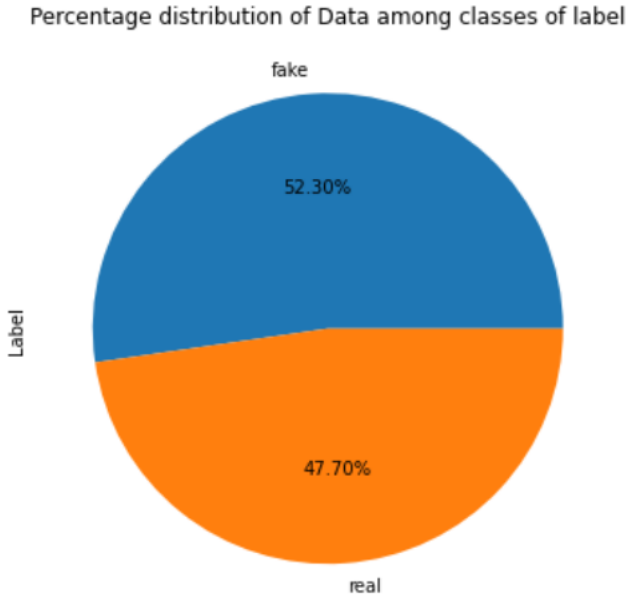
```
# let's start with label column.
sns.countplot(df_final['Label'])
print(df_final['Label'].value_counts())
```

```
fake    23481
real    21417
Name: Label, dtype: int64
```



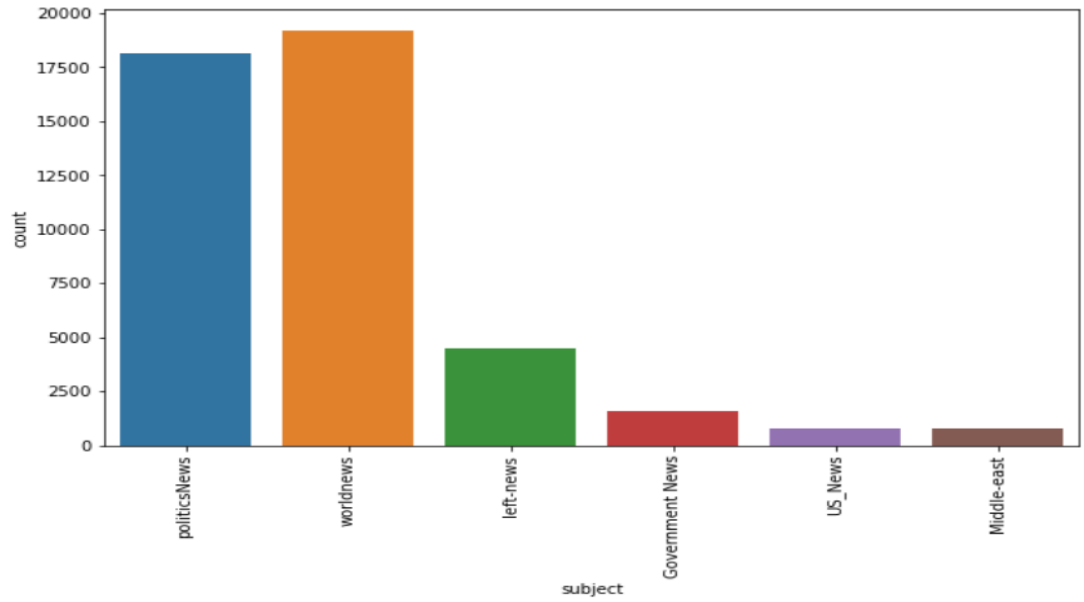
We can observe that both fake and real label in the dataset are nearly same, with some minor differences. It simply means our label classes are balanced.

```
plt.figure(figsize=(10,6))
(df_final['Label'].value_counts()*100.0 /len(df_final)).plot.pie(autopct='%.2f%%')
plt.title("Percentage distribution of Data among classes of label")
plt.show()
```



We can observe we got 47.70% of real news data and remaining 52.30% of fake news data.

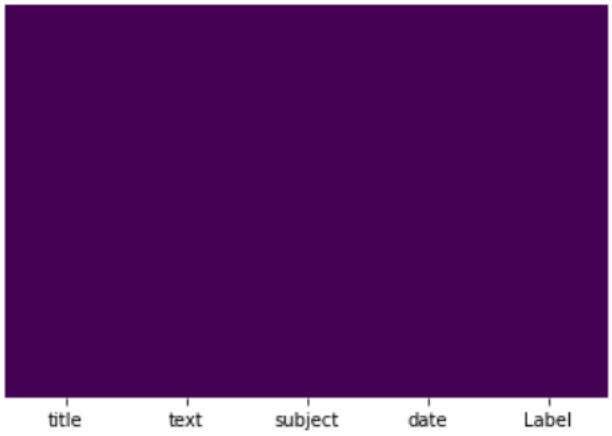
```
# subject distribution visualization
plt.figure(figsize= (10,6))
sns.countplot(df_final['subject'])
plt.xticks(rotation= 90)
plt.show()
```



From above plot we get to observe the subject wise distribution of dataset.

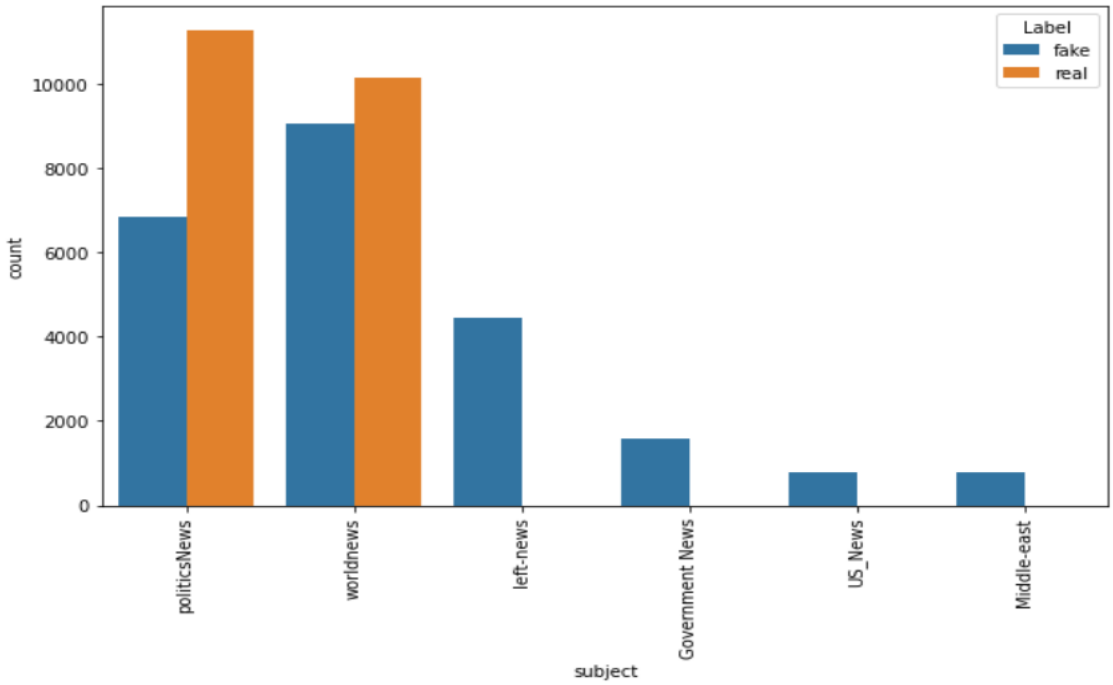
```
# Visualize null values in the dataset if any
sns.heatmap(df_final.isnull(),yticklabels=False , cbar= False, cmap= 'viridis')
```

```
<AxesSubplot:>
```



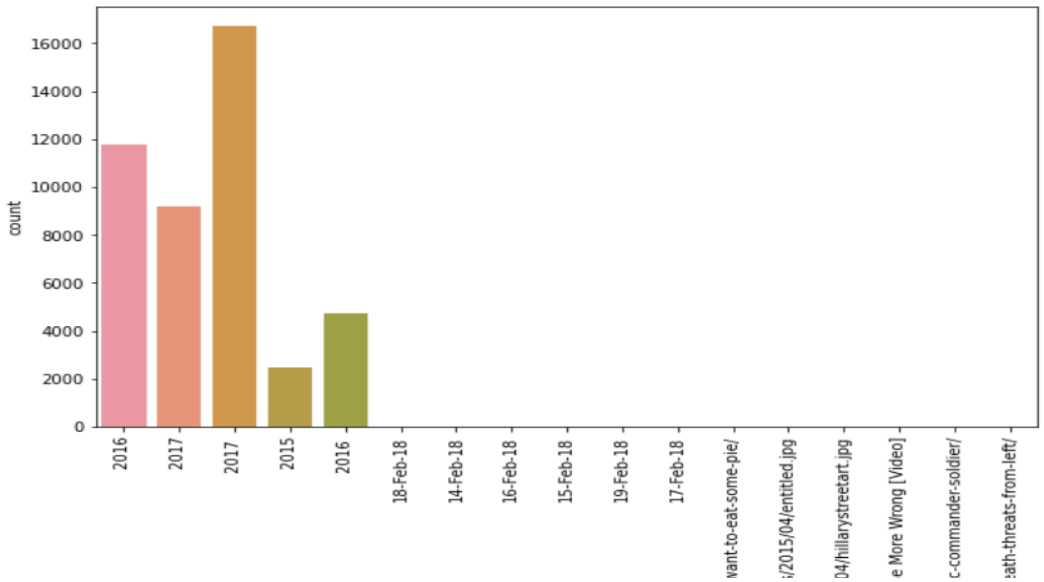
From above plot we observe that our dataset doesn't have any null values. If it has any null value then it will be visible with different colour on the plot.

```
plt.figure(figsize=(10,6))
sns.countplot(x='subject', hue='Label', data= df_final)
plt.xticks(rotation= 90)
plt.show()
```



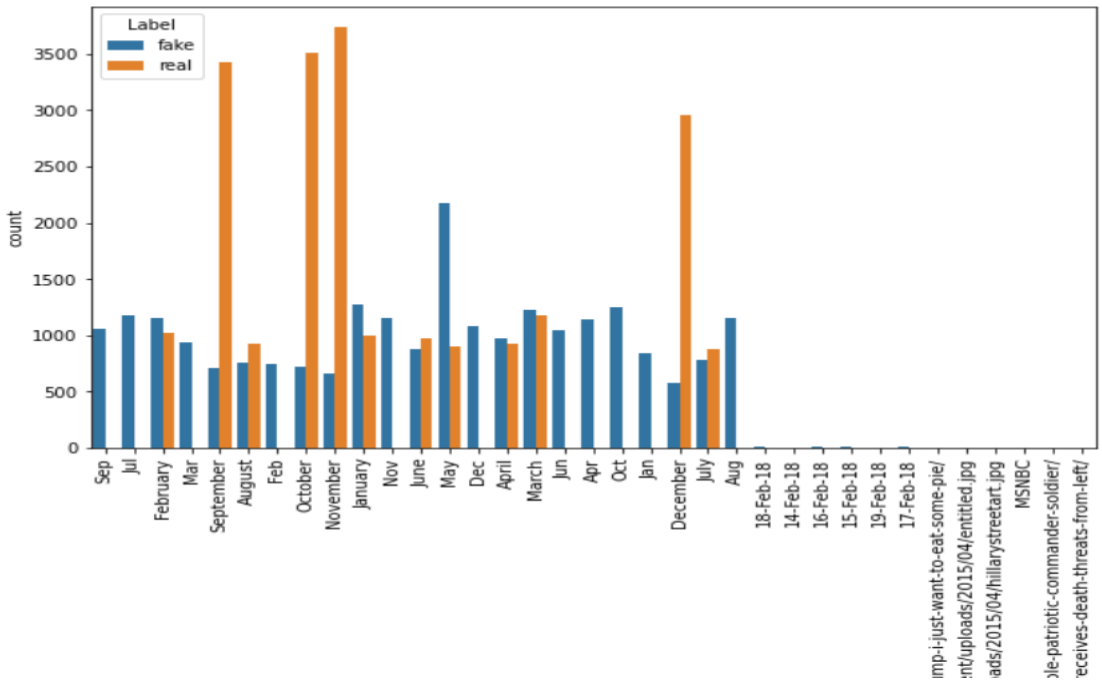
From above plot it is clear that the subjects for fake and real news are different, except worldnews and politicsNews. Interestingly realnews belongs to only two category worldnews and politicsNews. Rest all subject classes belongs to fake news.

```
plt.figure(figsize=(10,6))
sns.countplot('Year', data= df_final)
plt.xticks(rotation=90)
plt.show()
```



We tried to do feature extraction for date column just to see if there is any strong correlation between Label and date. Above plot show the data distribution on the basis of Year.

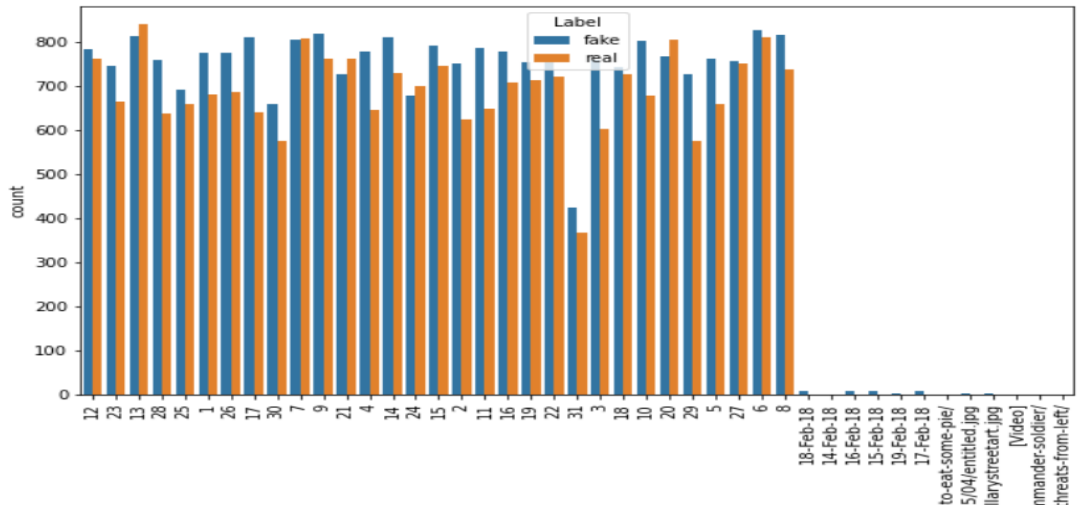
```
plt.figure(figsize=(10,6))
sns.countplot('Month',hue='Label', data= df_final)
plt.xticks(rotation= 90)
plt.show()
```



We observe that the date distribution throughout the dataset is not uniform, because of which we get some vague responses as well.

Also we notice that we got both abbreviation and full name for Months, which also unnecessarily vague the distribution pattern. This is similar situation for Year column as well.

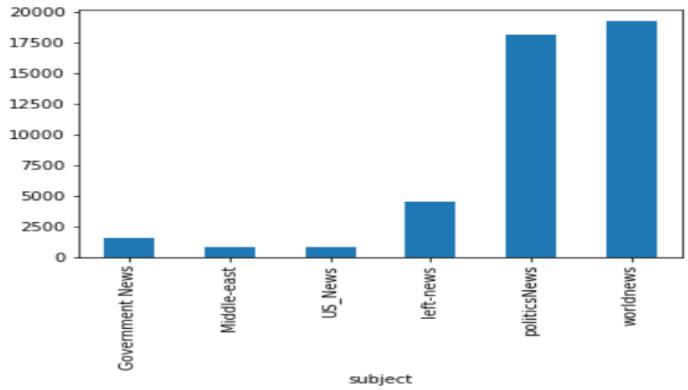
```
plt.figure(figsize=(10,6))
sns.countplot('Date',hue= 'Label', data= df_final)
plt.xticks(rotation =90)
plt.show()
```



We got similar observation for all the three extracted columns of date.
I also think that it won't show any strong connection with our target column, so it is better option to drop it here only.

```
# How many articles per subject?
print(df_final.groupby(['subject'])['text'].count())
df_final.groupby(['subject'])['text'].count().plot(kind="bar")
plt.show()
```

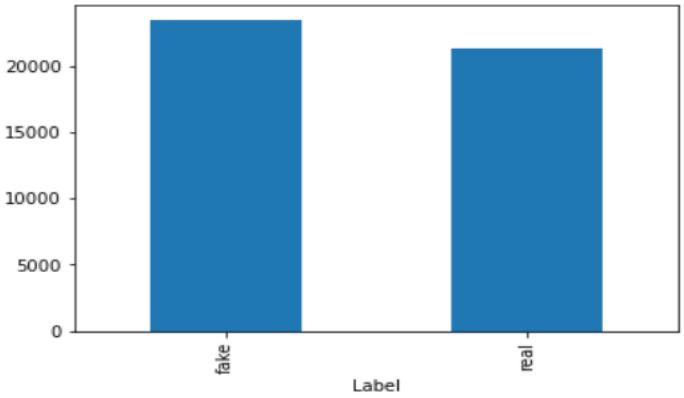
```
subject
Government News      1570
Middle-east           778
US_News               783
left-news            4459
politicsNews        18113
worldnews           19195
Name: text, dtype: int64
```



From above observation we can see that subject related to Middle east news are lowest while worldnews are highest.

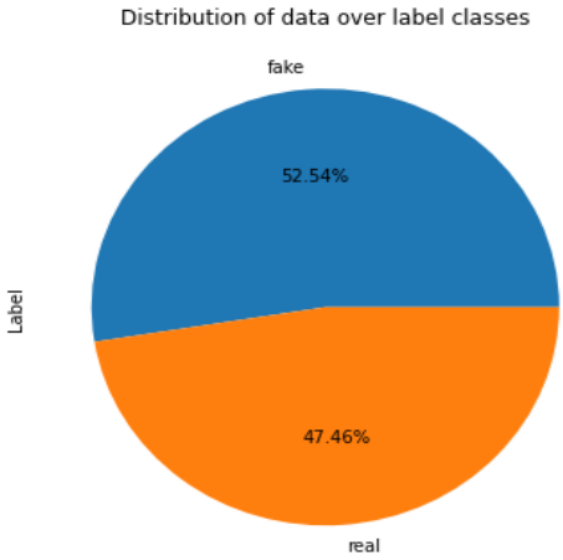
```
# How many fake and real articles?
print(df_final.groupby(['Label'])['text'].count())
df_final.groupby(['Label'])['text'].count().plot(kind="bar")
plt.show()
```

```
Label
fake    23481
real    21417
Name: text, dtype: int64
```



Above plot shows clear picture of text related to fake news and real news. We got text related to fake news are 23481 while text related to real news are 21417.

```
# chceking Label distribution after droping duplicates
plt.figure(figsize=(10,6))
(data['Label'].value_counts()*100.0 /len(data)).plot.pie(autopct='%.2f%%')
plt.title("Distribution of data over label classes")
plt.show()
```



From above observation we can see that even after dropping duplicates from the dataset, the %age distribution of real and fake news is more or less in the same line.

- Interpretation of the Results:
  1. Over dataset consist of two label classes, Fake and real, which simultaneously denotes Fake news and Real News.
  2. The distribution of data among the two classes of label are somewhat balanced, so we can easily take the same distribution for model building.
  3. We drop date column, as we don't find any definite relation between target column and date feature.
  4. Subject seems to be the topic of the news; we can label encode them if we want to use it further.
  5. We performed NLP pre processing techniques on our text and title column, where we tokenized them, remove all the stop words, punctuations, whitespace, hyperlinks, emails, special character etc and them lemmatize them in order to get pre-processed clean data.
  6. We perform TFIDF vectorization on our pre processed clean data, in order to vectorized it to machine understandable format.
  7. When we applied our classification algorithm on this pre-processed vectorized data, we can see that Logistic regression, SVC, XGBoost all three gives good results for our dataset.
  8. The deciding metrics for best algorithm is auc score and F1 score. And for Logistics regression we got F1 score of 0.996735 & ROC AUC score of 0.999818
  9. Confusion Matrix shows high classification of accuracy 25 out of 7021 are incorrect.
  10. Overall model fit is good.

Conclusion

- Key Findings and Conclusions of the Study:

  We split our dataset into three parts train, CV and test. This distribution helps to reduce data leakage, and also increase the efficiency of the model.

We convert pre-processed data into vectors by using TFIDF vectorizer. As we use tfidf vectorizer we don't need to scale the data before model building.
We use hstack in order to merge all the pre-processed vectorized columns.
Here is the summary of each algorithm we build in our project.

	Models	F1 Score	ROC AUC Score
4	Support Vector Machine (SVC)	0.996664	0.999844
1	Logistic Regression	0.996735	0.999818
5	XGBoost Classifier	0.997014	0.999771
2	RandomForestClassifier	0.993598	0.999697
3	DecisionTree Classifier	0.995815	0.995538
0	MultinomialNB	0.958664	0.991380

From here we can see that top 3 algorithms i.e., SVC, LogisticRegression and XGBoost Classifier are almost tying in their scores. However, if we consider both f1 score and ROC AUC score we can see that Logistic regression has edge in both. So, I think Logistic Regression seems to be best suited model for our dataset. We finally go ahead and save the best fitted Logistic regression model. As we already tune couple parameters while building the model.

- Learning Outcomes of the Study in respect of Data Science

  What's inside is more than just rows and columns. Make it easy for others to get started by describing how you acquired the data and what time period it represents, too.
  In our project we build a NLP model to detect fake news, here we took a fake and true dataset, the implemented some data cleaning function, then vectorized it using TFIDF vectorizer and finally build a Logistic Regression model with .99 of f1 score and .99 of roc auc score.
  This algorithm gives an edge to the business to detect the fake news before spreading it like water. And hence help in verification of news.

- Limitations of this work and Scope for Future Work

  Detecting fake news and stop them from spreading is important in many aspects. Our model helps in detecting it, but to built a system which helps in stopping the spread of such fake news a go to from here. Solving any problem can have various approaches, he we vectorized using tfidf, one can use word2vec in place and proceed from there to build a model, where he has to scale the model before building.

  NLP has a wide scope in many projects, and here we are just taking baby steps in that direction.
