Fake News Detection

Introduction:

The world is filled with a lot of news generated everyday. News contains very crucial information, which has significant effect on public, government, economy of a country. In these modern days, fake news are very common, which lead to bad impact on social health. So, identifying fake news is very important. Manual checking is not suitable because of lots of manpower and cost.

This project is aimed to build a ML model which uses statistical analysis and modern machine learning approaches to predict the fake news using the text and title of the news.

Dataset:

train.csv: A full training dataset with the following attributes:

id: unique id for a news article.

title: the title of a news article.

author: author of the news article.

text: the text of the article; could be incomplete.

label: a label that marks the article as potentially unreliable.

Where 1: unreliable and 0: reliable

Objectives:

- Build a model which predicts the fake news using text and title of the news.
- To make a black list of words which are more likely to cause fake news.
- To determine whether there is difference in lentgh of news (count of words) between real and fake news.

Scope Of Study:

Fake news is very common in these days and it is very problematic. It will cause very serious situations and mislead people. And it is not easy to check whether the news is fake or not. Machine learning approach to this will solve the problem easily. Without much cost, energy and time, ML models can be used to detect fake news more accurately.

Exploratory Data Analysis

In [1]:

```
#import libraries:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from scipy.sparse import hstack
from sklearn.metrics import accuracy_score,confusion_matrix, log_loss, f1_score
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
data=pd.read_csv("D:\\collage project\\FAKE NEWS\\train.csv")
data
```

Out[2]:

	id	title	author	text	label
0	0	House Dem Aide: We Didn’t Even See Comey’s Let...	Darrell Lucus	House Dem Aide: We Didn’t Even See Comey’s Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1
...	...	...	...	...	...
20795	20795	Rapper T.I.: Trump a ’Poster Child For White S...	Jerome Hudson	Rapper T. I. unloaded on black celebrities who...	0
20796	20796	N.F.L. Playoffs: Schedule, Matchups and Odds -...	Benjamin Hoffman	When the Green Bay Packers lost to the Washing...	0
20797	20797	Macy’s Is Said to Receive Takeover Approach by...	Michael J. de la Merced and Rachel Abrams	The Macy’s of today grew from the union of sev...	0
20798	20798	NATO, Russia To Hold Parallel Exercises In Bal...	Alex Ansary	NATO, Russia To Hold Parallel Exercises In Bal...	1
20799	20799	What Keeps the F-35 Alive	David Swanson	David Swanson is an author, activist, journa...	1

20800 rows × 5 columns

In [3]:
```
data.head()
```

Out[3]:

	id	title	author	text	label
0	0	House Dem Aide: We Didn’t Even See Comey’s Let...	Darrell Lucus	House Dem Aide: We Didn’t Even See Comey’s Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0

	id	title	author	text	label
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1

In [4]:
```
# Shape of the Data:
data.shape
```

Out[4]:
```
(20800, 5)
```

This Dataset contains 5 variables and 20800 observations.

In [5]:
```
# information About the Dataset:

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20800 entries, 0 to 20799
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      20800 non-null  int64 
 1   title   20242 non-null  object
 2   author  18843 non-null  object
 3   text    20761 non-null  object
 4   label   20800 non-null  int64 
dtypes: int64(2), object(3)
memory usage: 812.6+ KB
```

In [6]:
```
# Data types:
data.dtypes
```

Out[6]:
```
id         int64
title     object
author    object
text      object
label      int64
dtype: object
```

In [7]:
```
data.describe()
```

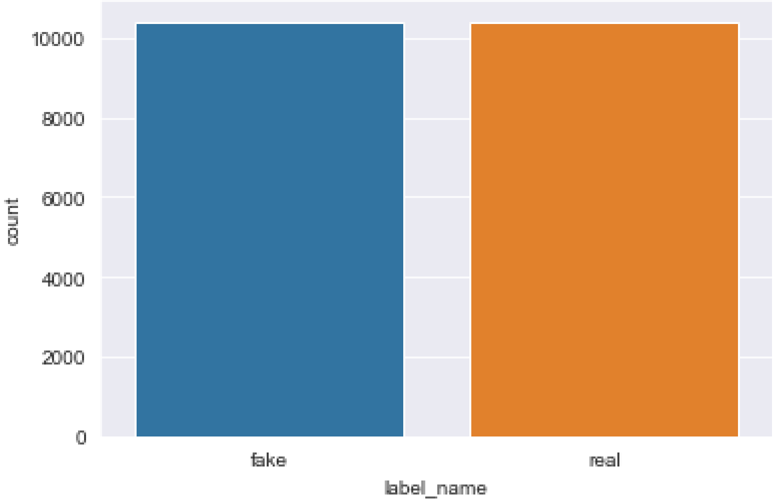
Out[7]:

	id	label
count	20800.000000	20800.000000
mean	10399.500000	0.500625
std	6004.587135	0.500012
min	0.000000	0.000000
25%	5199.750000	0.000000
50%	10399.500000	1.000000
75%	15599.250000	1.000000
max	20799.000000	1.000000

Check the imbalance

In [8]:
```
# check whether our data is balanced :
data["label_name"]=np.where(data["label"]==1,"fake","real")
sns.set_style('darkgrid')
sns.countplot(data['label_name'])
```

Out[8]:
```
<AxesSubplot:xlabel='label_name', ylabel='count'>
```



Here 0 represent Fake News and 1 represent Real news

We can conclude from the plot that the data is balanced.

This is important because, without knowing the data imbalance, we cannot consider the appropriate metric and model.

Choosing the performance metric:

Accuracy:

Accuracy can be used since the data is balanced. And it is simple to understand and interprete.

Data Cleaning:

In [9]:
```
data.drop("label_name",axis=1,inplace=True)
```

In [10]:
```
# Find Null Values:
null=data.isnull().sum()
null
```

Out[10]:
```
id           0
title      558
author    1957
text        39
label        0
dtype: int64
```

Title and Author and Text columns having lot of null values so fill null values by imputation method.

In [11]:
```
# Columns Name:
data.columns
```

Out[11]:
```
Index(['id', 'title', 'author', 'text', 'label'], dtype='object')
```

Data Preprocessing:

In [12]:

```
#calculating total null values:

total_rows =data.shape[0]
total_null =null.sum()

# percent of data that is null:

print("Percentage of null value instances present is : ",round((total_null/total_row
```

```
Percentage of null value instances present is :  12.279 %
```

In [13]:

```
# Impute missing values in dish liked with "unknown", so that "unknown" also conside
data['author'] = data['author'].fillna("unknown")
data['author']
```

Out[13]:

```
0                                    Darrell Lucus
1                                  Daniel J. Flynn
2                               Consortiumnews.com
3                                  Jessica Purkiss
4                                   Howard Portnoy
                           ...                    
20795                               Jerome Hudson
20796                           Benjamin Hoffman
20797     Michael J. de la Merced and Rachel Abrams
20798                                 Alex Ansary
20799                               David Swanson
Name: author, Length: 20800, dtype: object
```

In [14]:

```
#calculating total null values:

total_rows = data.shape[0]
total_null = null.sum()

# percent of data that is null:

print("Percentage of null value instances present is : ",round((total_null/total_row
```

```
Percentage of null value instances present is :  12.279 %
```

In [15]:

```
# Remove all other Null Values:
data.dropna(inplace=True)
```

In [16]:

```
data.isnull().sum()
```

Out[16]:

```
id        0
title     0
author    0
text      0
label     0
dtype: int64
```

So there is no null values now.

Text Cleaning:

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039

def decontracted(phrase):
    """This function will converts shorthend form to full form of english words"""
    # 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", " 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
```

In [18]:

```
from tqdm import tqdm
def preprocess_text(text_data):
    """This function will clean the text data"""
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)

        # removing \\r
        sent = sent.replace('\\r', ' ')

        #remove NaN text
        sent = sent.replace('NaN', ' ')

        #
        sent = sent.replace('\\n', ' ')

        #
        sent = sent.replace('\\"',  ' ')

        #substituting number with space
        sent = re.sub(r'[0-9]+',  ' ',sent)


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

        sent = ' '.join([w for w in sent.split() if len(w)>=3])
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
        # https://gist.github.com/sebleier/554280

    return preprocessed_text
```

In [19]:

```
def categorical_data(cat):

    """This function cleanse categorical text data"""

    preprocessed_cat = []

    for point in cat:

        lst_words = point.split()

        for i in range(len(lst_words)):
            splitted =lst_words[i].split()

            if len(splitted)>1:
                lst_words[i] = "_".join(splitted)
        point = "".join(lst_words).lower()

        preprocessed_cat.append(point)

    return preprocessed_cat
```

In [20]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not' because the

stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', '
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throu
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 't
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've",
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn'
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'm
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't","NaN",]
```

In [21]:

```
data['text'] = preprocess_text(data['text'].values)
```

```
100%|██████████████████████████████████████████████████████████████| 20
203/20203 [01:50<00:00, 183.41it/s]
```

In [22]:

```
data['title'] = preprocess_text(data['title'].values)
```

```
100%|█████████████████████████████████████████████████████████████| 202
03/20203 [00:02<00:00, 7716.12it/s]
```

In [23]:

```
data['author'] = categorical_data(data['author'].values)
```

In [24]:

```
data['text'][0]
```

Out[24]:

```
'house dem aide even see comey letter jason chaffetz tweeted darrell lucus october s
ubscribe jason chaffetz stump american fork utah image courtesy michael jolley avail
able creative commons license apologies keith olbermann doubt worst person world wee
k fbi director james comey according house democratic aide looks like also know seco
nd worst person well turns comey sent infamous letter announcing fbi looking emails
may related hillary clinton email server ranking democrats relevant committees hear
comey found via tweet one republican committee chairmen know comey notified republic
an chairmen democratic ranking members house intelligence judiciary oversight commit
tees agency reviewing emails recently discovered order see contained classified info
rmation not long letter went oversight committee chairman jason chaffetz set politic
al world ablaze tweet fbi dir informed fbi learned existence emails appear pertinent
investigation case reopened jason chaffetz jasoninthehouse october course know not c
ase comey actually saying reviewing emails light unrelated case know anthony weiner
sexting teenager apparently little things facts matter chaffetz utah republican alre
ady vowed initiate raft investigations hillary wins least two years worth possibly e
ntire term worth apparently chaffetz thought fbi already work resulting tweet briefl
y roiled nation cooler heads realized dud according senior house democratic aide mis
reading letter may least chaffetz sins aide told shareblue boss democrats even know
comey letter time found checked twitter democratic ranking members relevant committe
es receive comey letter republican chairmen fact democratic ranking members receive
chairman oversight government reform committee jason chaffetz tweeted made public le
t see got right fbi director tells chaffetz gop committee chairmen major development
potentially politically explosive investigation neither chaffetz nor colleagues cour
tesy let democratic counterparts know instead according aide made find twitter alrea
dy talk daily kos comey provided advance notice letter chaffetz republicans giving t
ime turn spin machine may make good theater nothing far even suggests case nothing f
```

```
ar suggests comey anything grossly incompetent tone deaf suggest however chaffetz ac
ting way makes dan burton darrell issa look like models responsibility bipartisanshi
p even decency notify ranking member elijah cummings something explosive trample bas
ic standards fairness know granted not likely chaffetz answer sits ridiculously repu
blican district anchored provo orem cook partisan voting index gave mitt romney puni
shing percent vote moreover republican house leadership given full support chaffetz 
planned fishing expedition mean turn hot lights textbook example house become republ
ican control also second worst person world darrell lucus darrell something graduate
university north carolina considers journalist old school attempt turn member religi
ous right college succeeded turning religious right worst nightmare charismatic chri
stian unapologetic liberal desire stand scared silence increased survived abusive th
ree year marriage may know daily kos christian dem follow twitter darrelllucus conne
ct facebook click buy darrell mello yello connect'
```

In [25]:

```
data['title'][0]
```

Out[25]:

```
'house dem aide even see comey letter jason chaffetz tweeted'
```

In [26]:

```
data['author'][0]
```

Out[26]:

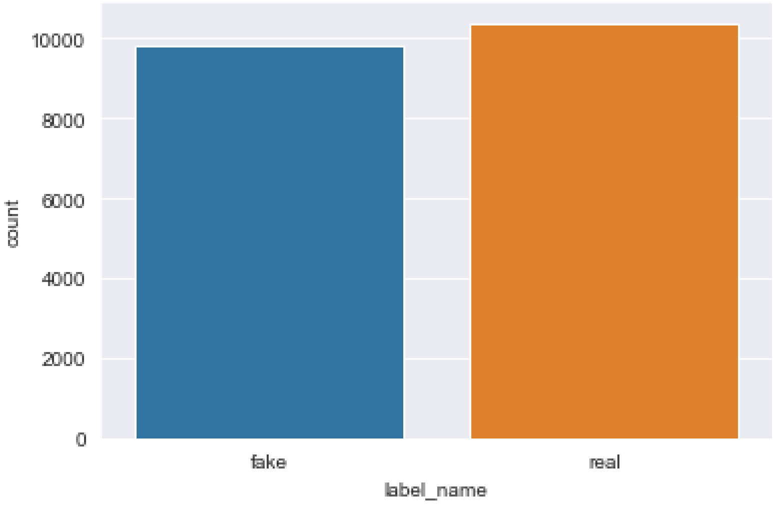
```
'darrelllucus'
```

In [27]:

```
# check whether our data is balanced :
data["label_name"]=np.where(data["label"]==1,"fake","real")
sns.set_style('darkgrid')
sns.countplot(data['label_name'])
```

Out[27]:

```
<AxesSubplot:xlabel='label_name', ylabel='count'>
```



In [28]:

```
data['label_name'].value_counts()
```

Out[28]:

```
real    10387
fake     9816
Name: label_name, dtype: int64
```

Model Splitting:

In [29]:

```
#train test splitting:
y = data['label']
x = data.drop(["label","id"], axis=1)
```

In [30]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=353)
```

In [31]:

```
x_train.shape
```

Out[31]:

```
(16162, 4)
```

In [32]:

```
x_test.shape
```

Out[32]:

```
(4041, 4)
```

In [33]:

```
y_train.shape
```

Out[33]:

```
(16162,)
```

In [34]:

```
y_test.shape
```

Out[34]:

```
(4041,)
```

BOW vectorizer:

In [35]:

```
vec = CountVectorizer()

#fitting countvectorizer using only train data:

vec.fit(x_train["text"].values)

#transforming to vector representation for train,test data:

x_train_text = vec.transform(x_train["text"].values)
x_test_text = vec.transform(x_test["text"].values)

print(x_train_text.shape)
print(x_test_text.shape)
```

```
(16162, 132829)
(4041, 132829)
```

In [36]:

```
vec = CountVectorizer()

#fitting countvectorizer using only train data:

vec.fit(x_train["title"].values)

#transforming to vector representation for train,test data:

x_train_title = vec.transform(x_train["title"].values)
x_test_title = vec.transform(x_test["title"].values)

print(x_train_title.shape)
print(x_test_title.shape)
```

```
(16162, 18539)
(4041, 18539)
```

In [37]:

```
# Use one hot encoding for this

vec = CountVectorizer()

#fitting countvectorizer using only train data:

vec.fit(x_train["author"].values)

#transforming to vector representation for train,test data:

x_train_author = vec.transform(x_train["author"].values)
x_test_author = vec.transform(x_test["author"].values)
```

```
print(x_train_author.shape)
print(x_test_author.shape)
```

```
(16162, 3668)
(4041, 3668)
```

In [38]:

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

In [39]:

```
#Concatenating all features
x_tr=hstack((x_train_text,x_train_title,x_train_author)).tocsr()
x_te=hstack((x_test_text,x_test_title,x_test_author)).tocsr()

print("FINAL DATA MATRIX SHAPE IS ........")
print(x_tr.shape,y_train.shape)
print(x_te.shape,y_test.shape)
print("*"*100)
```

```
FINAL DATA MATRIX SHAPE IS ........
(16162, 155036) (16162,)
(4041, 155036) (4041,)
************************************************************************************
****************
```

Data Analysis:

In [133…

```
# determine whether title contain more number of real or fake news.
fake_news_count = data[data.label == 1]['title'].value_counts()
real_news_count = data[data.label == 0]['title'].value_counts()
```

In [134…

```
fake_count = pd.DataFrame({
    'title':fake_news_count.index,
    'Fake':fake_news_count.values
})
real_count = pd.DataFrame({
    'title':real_news_count.index,
    'Real':real_news_count.values
})
```

In [135…

```
rf_count = pd.merge(real_count, fake_count, on='title', how='outer').fillna(0)
rf_count['Real'] = rf_count['Real'].astype(int)
rf_count['Fake'] = rf_count['Fake'].astype(int)
rf_count
```

Out[135…

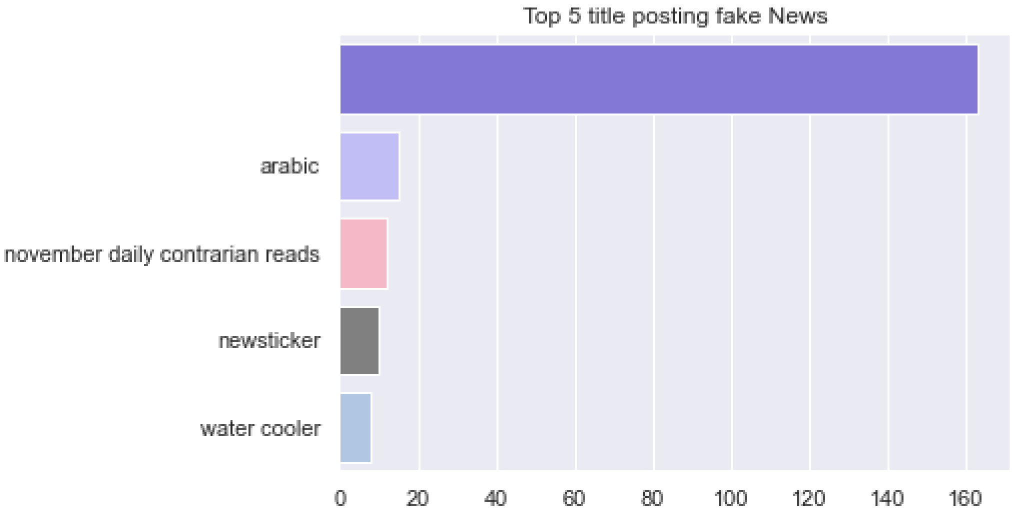
	title	Real	Fake
0	cook week new york times	4	0
1	great stories nothing politics new york times	4	0
2	right left partisan writing miss new york times	2	0
3	trump new york times	2	0
4	cook weekend new york times	2	0
...	...	...	...
19346	comments week comes trouble	0	1
19347	comment holocaust jewish population numbers je...	0	1
19348	hillary clinton private speech mentioned pales...	0	1
19349	lawsuits states accuse trump gop voter intimid...	0	1

	title	Real	Fake
19350	took less hours new email investigation story ...	0	1

19351 rows × 3 columns

In [145…

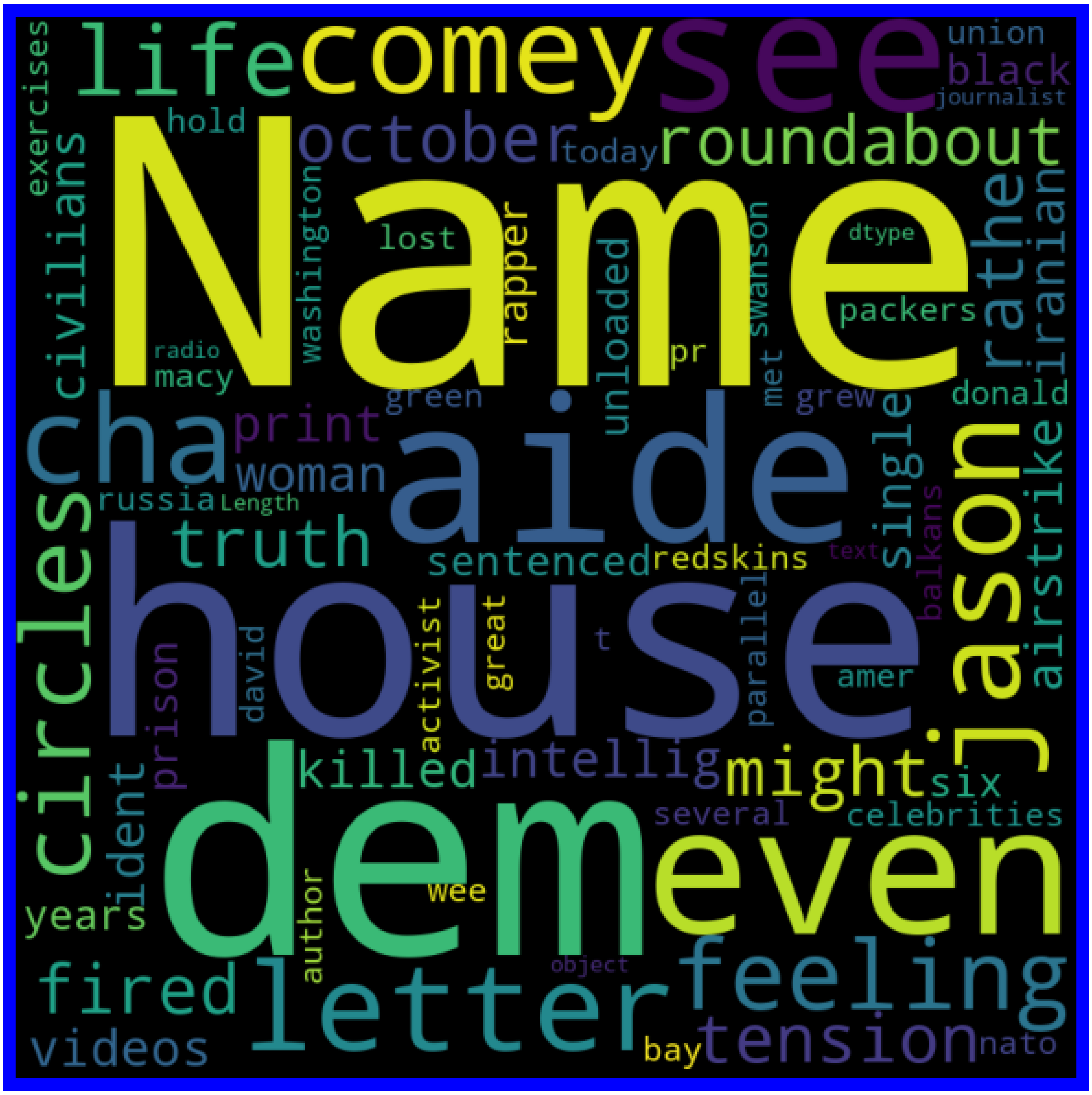
```
#Cheack top five Title contains Fake news:

sns.barplot(y=fake_news_count[:5].index, x=fake_news_count[:5].values,
            palette=['#7868e6', '#b8b5ff', '#ffaec0', 'grey', '#a7c5eb'])
sns.despine(bottom=True, left=True)
plt.title('Top 5 title posting fake News');
```



Here "Cook week new york times" and "Great stories nothing politics new york times" titles having more fake news frequency

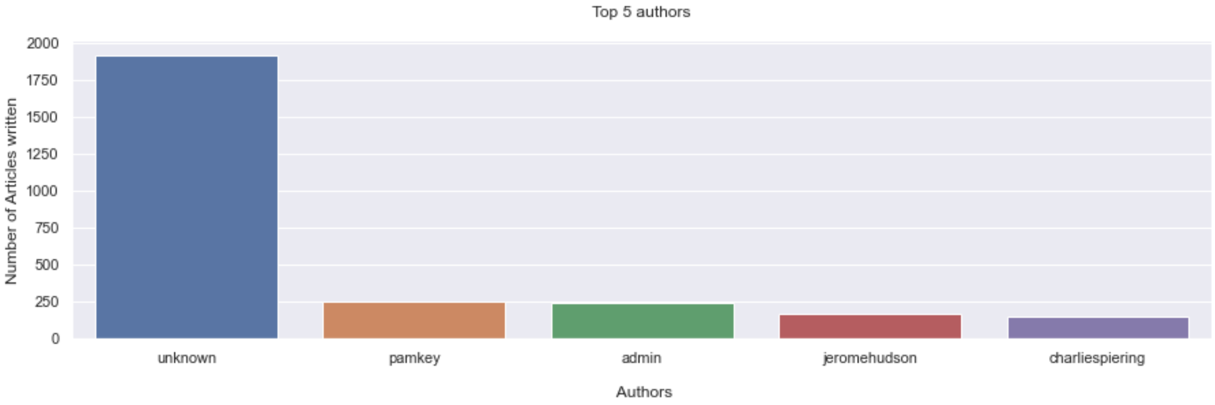
In [137…

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
stopwords = set(STOPWORDS)

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                stopwords = stopwords,
                min_font_size = 10).generate(str(data['text']))

# plot the WordCloud image
plt.figure(figsize = (8,8), facecolor = 'blue')
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0) ;
```



In [138…

```
d = data['author'].value_counts().sort_values(ascending=False).head(5)
d = pd.DataFrame(d)
d = d.reset_index() # dataframe with top 5 authors

# Plotting
sns.set()
plt.figure(figsize=(15,4))
sns.barplot(x='index', y='author', data=d)
plt.xlabel("\n Authors")
plt.ylabel("Number of Articles written")
plt.title("Top 5 authors\n")
plt.show()
```

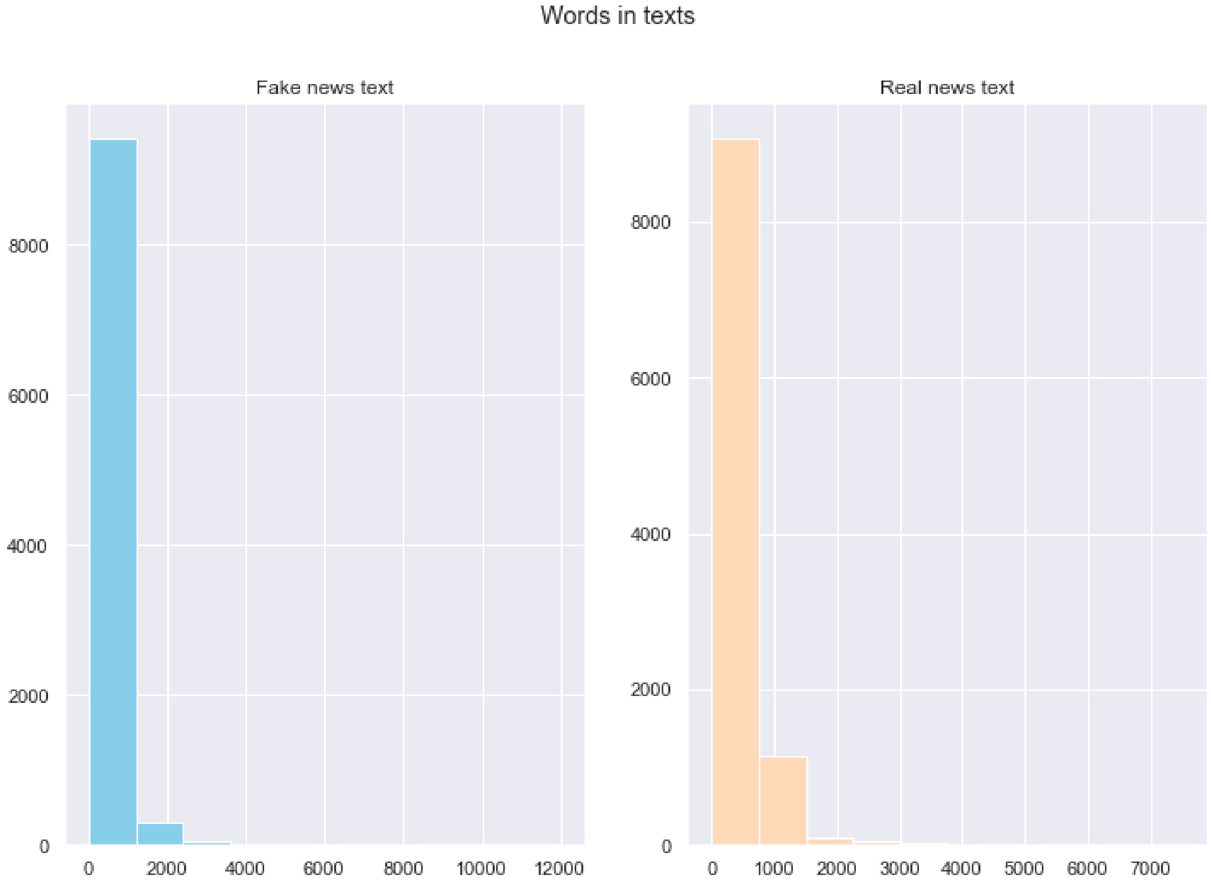


Here Pam key and admin authors are high.

There are lot of news which do not contain author names i.e. unknown

Analysis of count of words in text

```
In [139…
# Number of words in each text.

fig,(ax1,ax2)=plt.subplots(1,2,figsize=(12,8))
text_len=data[data['label']==1]['text'].str.split().map(lambda x: len(x))
ax1.hist(text_len,color='SkyBlue')
ax1.set_title('Fake news text')



text_len=data[data['label']==0]['text'].str.split().map(lambda x: len(x))
ax2.hist(text_len,color='PeachPuff')
ax2.set_title('Real news text')
fig.suptitle('Words in texts')
plt.show()
```



1000 words are most common in real news category while around 500 words are most common in fake news category.

By just observing this, we can say that real news contains more words than fake news. But this is not a statistical conclusion. We can carry out 2 sample t-test to compare this.

```
In [140…
data["length"] = [len(text) for text in data["text"]]

real_news_sample = data[data["label"]==0]["length"]
fake_news_sample = data[data["label"]==1]["length"]
```

```
In [141…
real_news_sample.mean()
```

```
Out[141… 3492.479541734861
```

```
In [142… fake_news_sample.mean()
```

```
Out[142… 2724.0915851670743
```

Two Sample T test:

```
In [143… import scipy.stats as stats
         t_stat, p_val = stats.ttest_ind(real_news_sample, fake_news_sample, equal_var=False)
```

```
In [144… stats.ttest_ind(real_news_sample, fake_news_sample, equal_var=False)
```

```
Out[144… Ttest_indResult(statistic=16.362499441450826, pvalue=9.461980060514652e-60)
```

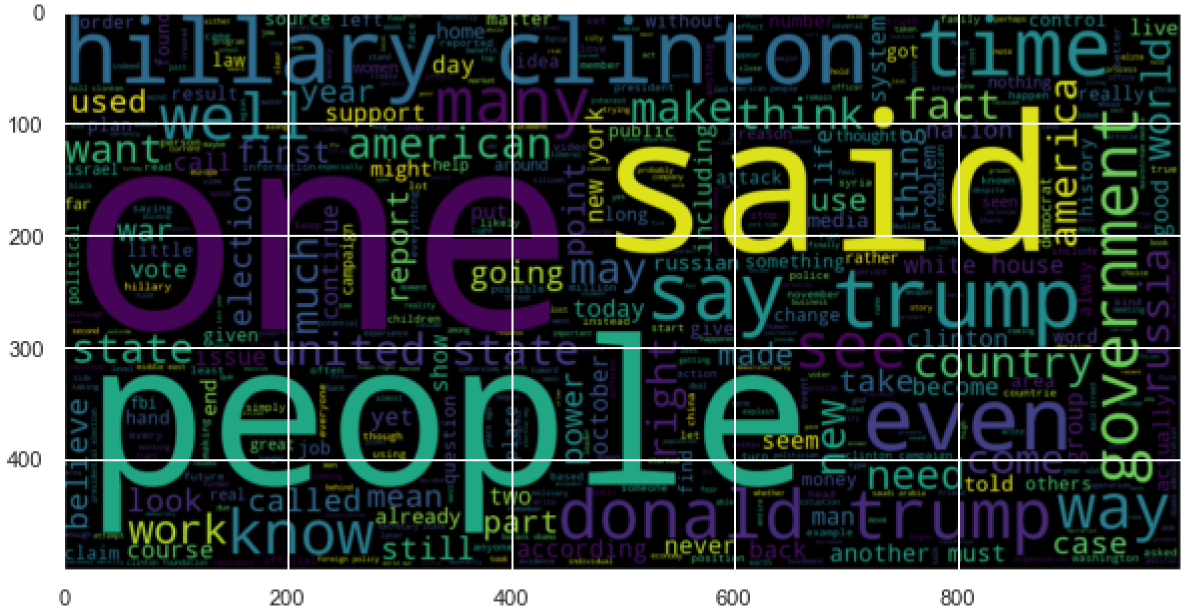
Conclusion:

P value is less than significant level. That is Pvalue < 0.05. Hence we reject H0. So by Two Sample t test there is significant difference between average length of fake news and real news.

WordCloud for Real News:

```
In [50]: from wordcloud import WordCloud,STOPWORDS
         plt.figure(figsize = (10,10))
         wc = WordCloud(max_words = 500 , width = 1000 , height = 500 , stopwords = STOPWORDS
         plt.imshow(wc , interpolation = 'bilinear')
```

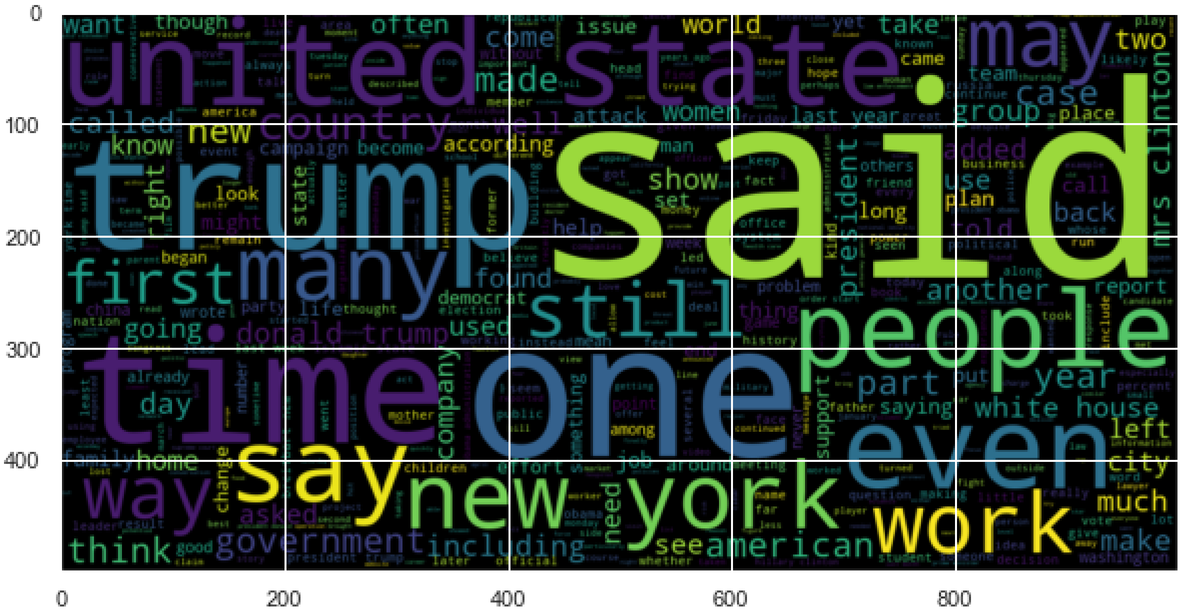
```
Out[50]: <matplotlib.image.AxesImage at 0x20dd8cd7bb0>
```



WordCloud for fake News

```
In [51]: plt.figure(figsize = (10,10))
         wc = WordCloud(max_words = 500 , width = 1000 , height = 500 , stopwords = STOPWORDS
         plt.imshow(wc , interpolation = 'bilinear')
```

```
Out[51]: <matplotlib.image.AxesImage at 0x20dd914a640>
```



Modelling the data

Logistic Regression

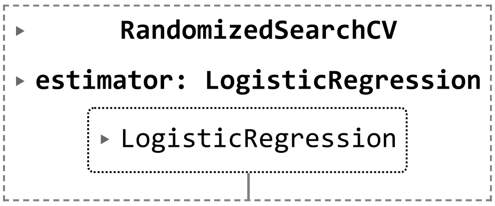
In [52]:
```
from sklearn.linear_model import LogisticRegression
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.Randomize
from sklearn.model_selection import RandomizedSearchCV
```

In [53]:
```
model = LogisticRegression()
```

Hyperparameter Tuning

In [54]:
```
hyperparameter = {"C" : [0.0001,0.001,0.01,0.1,1,10]}
search = RandomizedSearchCV(model, hyperparameter, scoring = "accuracy", random_stat
search.fit(x_tr,y_train)
```

Out[54]:


▸ RandomizedSearchCV

▸ estimator: LogisticRegression

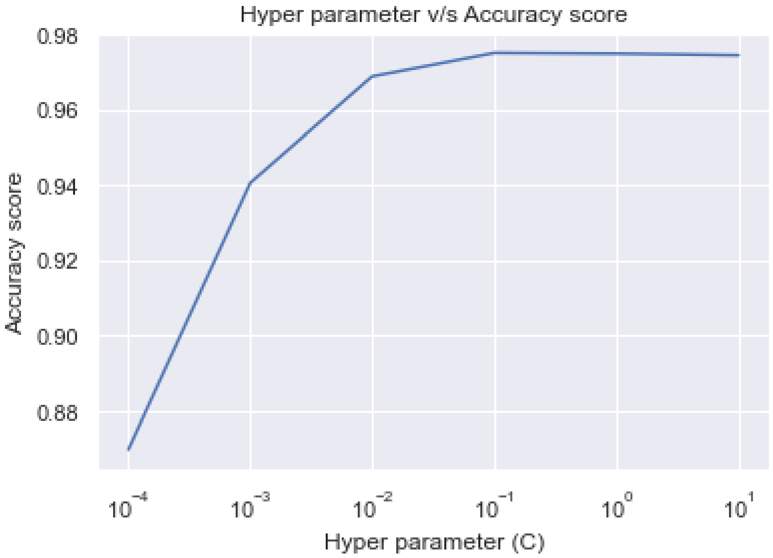
▸ LogisticRegression

In [55]:
```
search.cv_results_
```

Out[55]:
```
{'mean_fit_time': array([2.53019333, 3.93934231, 5.87677736, 8.67077656, 9.43222208,
        9.34314437]),
 'std_fit_time': array([0.05727277, 0.19091609, 0.26379319, 0.20598485, 0.1451983 ,
        0.07918666]),
 'mean_score_time': array([0.01060019, 0.0134315 , 0.01299248, 0.01466293, 0.0132841
6,
        0.01064591]),
 'std_score_time': array([0.00081468, 0.00207905, 0.00244764, 0.00359476, 0.0024457
2,
        0.00085658]),
 'param_C': masked_array(data=[0.0001, 0.001, 0.01, 0.1, 1, 10],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.0001},
  {'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 1},
  {'C': 10}],
 'split0_test_score': array([0.86545005, 0.93751933, 0.96814105, 0.97680173, 0.97680
173,
        0.97680173]),
 'split1_test_score': array([0.8731828 , 0.94401485, 0.97092484, 0.97742035, 0.97711
104,
        0.97649242]),
 'split2_test_score': array([0.86881188, 0.93997525, 0.96751238, 0.97431931, 0.97370
05 ,
        0.97215347]),
 'split3_test_score': array([0.87066832, 0.94306931, 0.96905941, 0.97462871, 0.97524
752,
        0.97462871]),
 'split4_test_score': array([0.87066832, 0.93842822, 0.96844059, 0.97215347, 0.97153
465,
        0.97215347]),
 'mean_test_score': array([0.86975627, 0.94060139, 0.96881565, 0.97506471, 0.9748790
9,
        0.97444596]),
 'std_test_score': array([0.00256282, 0.00254386, 0.00116639, 0.00188613, 0.0020682
5,
        0.00201406]),
 'rank_test_score': array([6, 5, 4, 1, 2, 3])}
```

In [56]:

```
plt.plot(hyperparameter["C"], search.cv_results_['mean_test_score'])
plt.xscale("log")
plt.xlabel("Hyper parameter (C)")
plt.ylabel("Accuracy score")
plt.title("Hyper parameter v/s Accuracy score")
plt.show()
```



For c=0.1 validation accuracy is high. so we can consider 0.1 is the best hyperparameter for our model.

In [57]:

```
search.best_params_
```

Out[57]:

```
{'C': 0.1}
```

Model with best hyperparameter

In [58]:

```
model = LogisticRegression(C = search.best_params_["C"])
model.fit(x_tr,y_train)
```

Out[58]:

```
▾  LogisticRegression
LogisticRegression(C=0.1)
```

In [59]:

```
coefs_df = pd.DataFrame()
coefs_df["coefs"] = model.coef_[0]
coefs_df["abs_coefs"] = abs(model.coef_[0])
coefs_df.sort_values(by="abs_coefs", inplace=True, ascending=False)
```

In [60]:

```
top_coefs = coefs_df.head(50)
top_coefs
```

Out[60]:

	coefs	abs_coefs
134846	-2.850448	2.850448
151284	-1.643217	1.643217
149518	-1.598940	1.598940
154865	1.429272	1.429272
151861	-1.096006	1.096006
14985	-1.039914	1.039914
143895	-1.010139	1.010139
42900	-1.002556	1.002556
82664	0.962555	0.962555
121452	-0.834175	0.834175
5131	0.795343	0.795343
151860	-0.772029	0.772029
81636	0.753945	0.753945
35637	0.639349	0.639349
80970	0.592978	0.592978
151862	-0.576345	0.576345
152002	-0.571661	0.571661
106180	0.556984	0.556984
110106	0.515807	0.515807
52671	0.511006	0.511006
22402	0.491886	0.491886
125800	0.482916	0.482916
153216	-0.479944	0.479944
71281	-0.457014	0.457014
105024	0.442244	0.442244
113843	-0.430602	0.430602
103119	-0.413177	0.413177
77387	-0.404610	0.404610

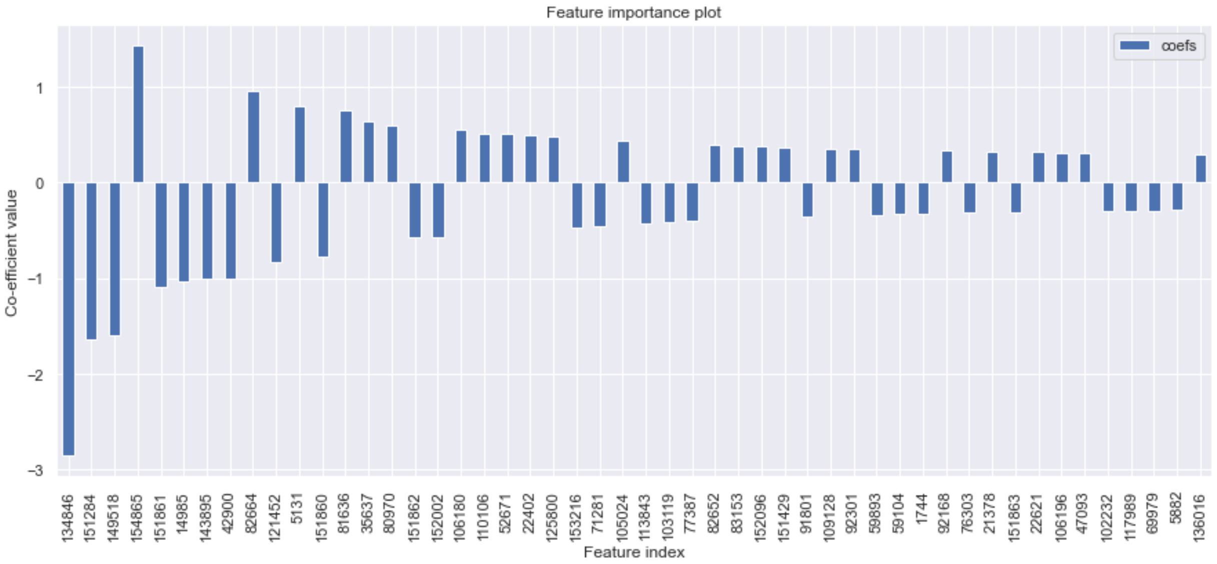
	coefs	abs_coefs
82652	0.399795	0.399795
83153	0.383349	0.383349
152096	0.380029	0.380029
151429	0.362534	0.362534
91801	-0.361924	0.361924
109128	0.348876	0.348876
92301	0.348701	0.348701
59893	-0.338336	0.338336
59104	-0.335731	0.335731
1744	-0.333319	0.333319
92168	0.333046	0.333046
76303	-0.319583	0.319583
21378	0.318092	0.318092
151863	-0.317486	0.317486
22621	0.316044	0.316044
106196	0.311680	0.311680
47093	0.307559	0.307559
102232	-0.303310	0.303310
117989	-0.297028	0.297028
69979	-0.293608	0.293608
5882	-0.292243	0.292243
136016	0.287326	0.287326

In [61]:

```
top_coefs["feature index"] = top_coefs.index
```

In [62]:

```
top_coefs[["coefs"]].plot(kind="bar",figsize=(15, 6))
plt.title("Feature importance plot")
plt.xlabel("Feature index")
plt.ylabel("Co-efficient value")
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

Model Evaluation

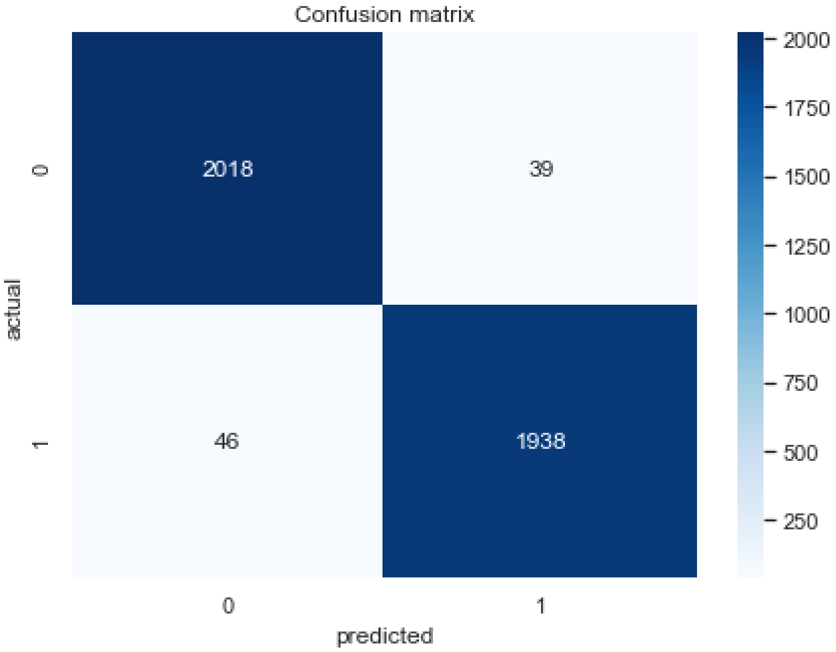
In [ ]:

In [63]:

```
y_test_pred = model.predict(x_te)
print("TEST PREDICTION: \n",y_test_pred[:3])
#predicting probability
y_test_pred_proba = model.predict_proba(x_te)
print("TEST PREDICTION PROBABILITY: \n",y_test_pred_proba[:3])


train_pred = model.predict(x_tr)
#predicting probability
train_pred_proba = model.predict_proba(x_tr)


accuracy = accuracy_score(y_test,y_test_pred)
print("Test log loss of the model is : ", np.round(accuracy*100,4), "%")
train_accuracy = accuracy_score(y_train,train_pred)
print("Train log loss of the model is : ", np.round(train_accuracy*100,4), "%")

print("="*50)

test_logloss = log_loss(y_test,y_test_pred_proba)
print("Test log loss of the model is : ", np.round(test_logloss,4))
train_logloss = log_loss(y_train,train_pred_proba)
print("Train log loss of the model is : ", np.round(train_logloss,4))

print("="*50)

test_f1 = f1_score(y_test,y_test_pred)
print("Test f1 score of the model is : ", np.round(test_f1*100,4))
train_f1 = f1_score(y_train,train_pred)
print("Train f1 score of the model is : ", np.round(train_f1*100,4))


cm1 = confusion_matrix(y_test,y_test_pred)
```

```
plt.figure(figsize=(7,5))
sns.heatmap(cm1, annot=True,fmt="d",cmap='Blues')
plt.xlabel("predicted")
plt.ylabel("actual")
plt.title("Confusion matrix")
plt.show()
```

```
TEST PREDICTION:
 [0 1 1]
TEST PREDICTION PROBABILITY:
 [[0.89573426 0.10426574]
 [0.01157584 0.98842416]
 [0.00109493 0.99890507]]
Test log loss of the model is :  97.8966 %
Train log loss of the model is :  99.9567 %
==================================================
Test log loss of the model is :  0.0746
Train log loss of the model is :  0.015
==================================================
Test f1 score of the model is :  97.8541
Train f1 score of the model is :  99.9553
```



In [ ]:

- 2018 data points which belongs to fake news are classified correctly.
- 46 data points which belongs to real news category are misclassified as fake news.
- 39 data points which belongs to fake news category are misclassified as real news.
- 1938 data points which belongs to real news are classified correctly.

KNN

In [64]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegr

from sklearn.neighbors import KNeighborsClassifier

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.Randomize
from sklearn.model_selection import RandomizedSearchCV
```

In [65]:
```
model = KNeighborsClassifier()
```

In [66]:
```
hyperparameter = {"n_neighbors" : [3,5,7,9,11,13,15,19,21]}
```

```
search = RandomizedSearchCV(model, hyperparameter, scoring = "accuracy", random_stat

search.fit(x_tr,y_train)
```

Out[66]:

```
▸        RandomizedSearchCV
▸ estimator: KNeighborsClassifier
      ▸ KNeighborsClassifier
```

In [67]:

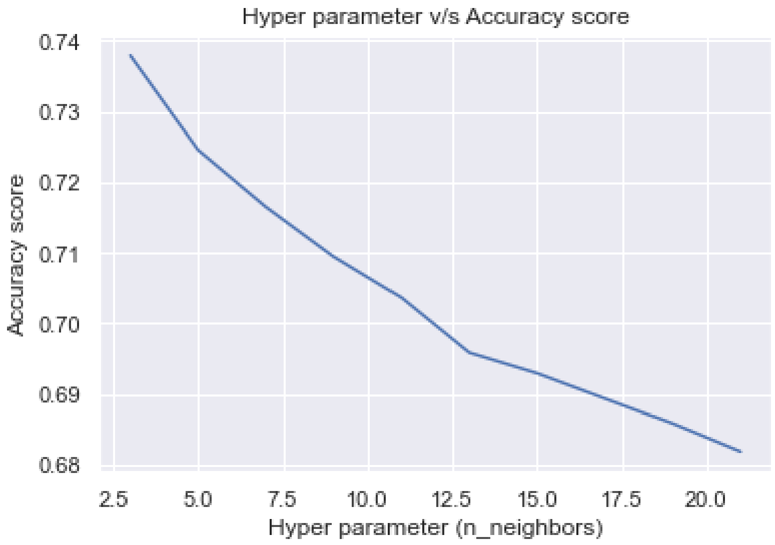
```
search.cv_results_
```

Out[67]:

```
{'mean_fit_time': array([0.05986347, 0.06069417, 0.06243091, 0.07088747, 0.06324778,
        0.06376486, 0.06740904, 0.0750968 , 0.06753979]),
 'std_fit_time': array([0.01237112, 0.00719888, 0.01192099, 0.00530294, 0.00661318,
        0.01034121, 0.00897797, 0.00966789, 0.00558381]),
 'mean_score_time': array([8.69746289, 9.42936983, 9.59136209, 9.806181  , 9.5899579
5,
        9.21916165, 9.58306046, 9.5725194 , 9.70794082]),
 'std_score_time': array([0.06309577, 0.18880388, 0.38274733, 0.5693801 , 0.1479986
9,
        0.18249445, 0.44665977, 0.26740583, 0.07985271]),
 'param_n_neighbors': masked_array(data=[3, 5, 7, 9, 11, 13, 15, 19, 21],
              mask=[False, False, False, False, False, False, False, False,
                    False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 3},
  {'n_neighbors': 5},
  {'n_neighbors': 7},
  {'n_neighbors': 9},
  {'n_neighbors': 11},
  {'n_neighbors': 13},
  {'n_neighbors': 15},
  {'n_neighbors': 19},
  {'n_neighbors': 21}],
 'split0_test_score': array([0.73368388, 0.7129601 , 0.70337148, 0.69873183, 0.69347
355,
        0.68543149, 0.68079183, 0.67336839, 0.6705846 ]),
 'split1_test_score': array([0.7370863 , 0.72038354, 0.71481596, 0.70832045, 0.70306
217,
        0.69347355, 0.69223631, 0.68481287, 0.67924528]),
 'split2_test_score': array([0.73019802, 0.72153465, 0.71441832, 0.70544554, 0.70266
089,
        0.69740099, 0.69337871, 0.68873762, 0.68626238]),
 'split3_test_score': array([0.73855198, 0.73174505, 0.72029703, 0.70946782, 0.70018
564,
        0.69461634, 0.6927599 , 0.68347772, 0.67790842]),
 'split4_test_score': array([0.75      , 0.73607673, 0.72957921, 0.72524752, 0.71875
,
        0.7082302 , 0.70544554, 0.69863861, 0.69523515]),
 'mean_test_score': array([0.73790404, 0.72454002, 0.7164964 , 0.70944263, 0.7036264
5,
        0.69583051, 0.69292246, 0.68580704, 0.68184716]),
 'std_test_score': array([0.00670214, 0.00831157, 0.00854413, 0.00873788, 0.0083053
3,
        0.00736733, 0.00780622, 0.00817767, 0.008341  ]),
 'rank_test_score': array([1, 2, 3, 4, 5, 6, 7, 8, 9])}
```

In [68]:

```
plt.plot(hyperparameter["n_neighbors"], search.cv_results_['mean_test_score'])
plt.xlabel("Hyper parameter (n_neighbors)")
plt.ylabel("Accuracy score")
plt.title("Hyper parameter v/s Accuracy score")
plt.show()
```



In [69]:
```
search.best_estimator_
```

Out[69]:
```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [70]:
```
search.best_params_
```

Out[70]:
```
{'n_neighbors': 3}
```

In [71]:
```
model = KNeighborsClassifier(n_neighbors = search.best_params_["n_neighbors"])
model.fit(x_tr,y_train)
```

Out[71]:
```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [ ]:

In [72]:
```
y_test_pred = model.predict(x_te)
```

In [ ]:

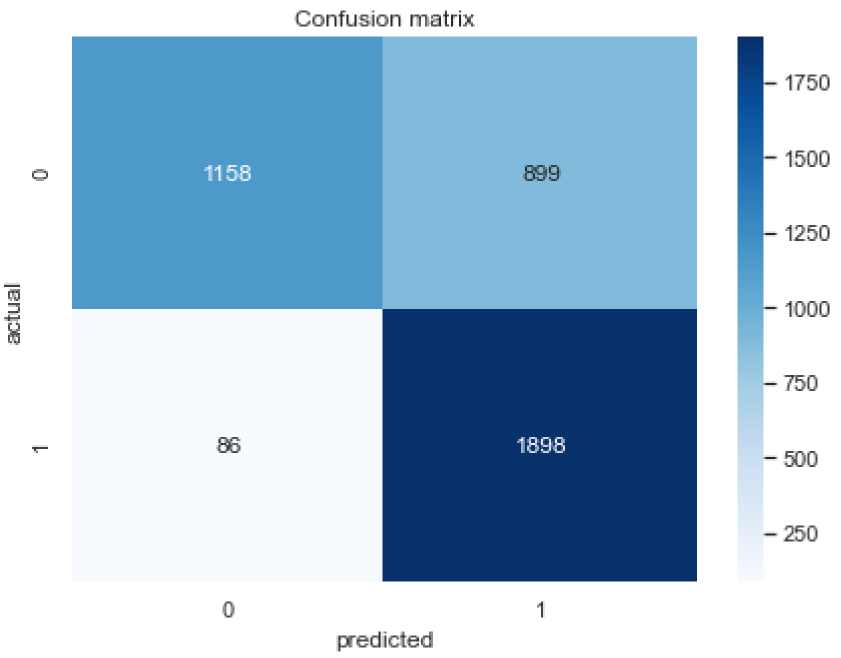
In [73]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [74]:
```
accuracy = accuracy_score(y_test,y_test_pred)

print("Test accuracy of the model is : ", np.round(accuracy*100,4), "%" )
train_pred = model.predict(x_tr)
train_accuracy = accuracy_score(y_train,train_pred)
print("Train accuracy of the model is : ", np.round(train_accuracy*100,4), "%")
```

```
Test accuracy of the model is :  75.6248 %
Train accuracy of the model is :  82.5393 %
```

In [75]:
```
cm1 = confusion_matrix(y_test,y_test_pred)
```

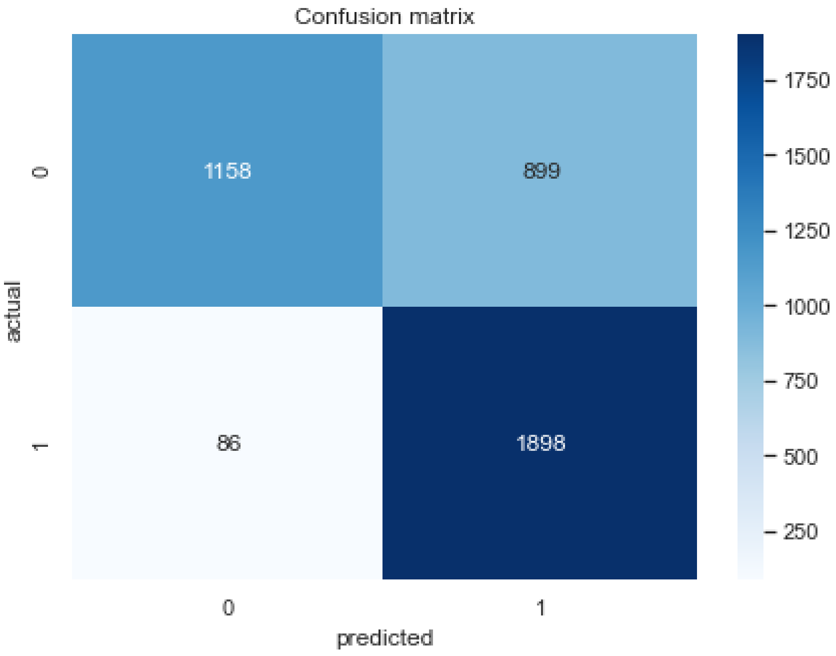
In [76]:
```
plt.figure(figsize=(7,5))

sns.heatmap(cm1, annot=True,fmt="d",cmap='Blues')
plt.xlabel("predicted")
plt.ylabel("actual")
plt.title("Confusion matrix")
plt.show()
```



In [77]:

```
y_test_pred = model.predict(x_te)
print("TEST PREDICTION: \n",y_test_pred[:3])
#predicting probability
y_test_pred_proba = model.predict_proba(x_te)
print("TEST PREDICTION PROBABILITY: \n",y_test_pred_proba[:3])


train_pred = model.predict(x_tr)
#predicting probability
train_pred_proba = model.predict_proba(x_tr)


accuracy = accuracy_score(y_test,y_test_pred)
print("Test log loss of the model is : ", np.round(accuracy*100,4), "%")
train_accuracy = accuracy_score(y_train,train_pred)
print("Train log loss of the model is : ", np.round(train_accuracy*100,4), "%")

print("="*50)

test_logloss = log_loss(y_test,y_test_pred_proba)
print("Test log loss of the model is : ", np.round(test_logloss,4))
train_logloss = log_loss(y_train,train_pred_proba)
print("Train log loss of the model is : ", np.round(train_logloss,4))

print("="*50)

test_f1 = f1_score(y_test,y_test_pred)
print("Test f1 score of the model is : ", np.round(test_f1*100,4))
train_f1 = f1_score(y_train,train_pred)
print("Train f1 score of the model is : ", np.round(train_f1*100,4))


cm1 = confusion_matrix(y_test,y_test_pred)
plt.figure(figsize=(7,5))
sns.heatmap(cm1, annot=True,fmt="d",cmap='Blues')
plt.xlabel("predicted")
plt.ylabel("actual")
plt.title("Confusion matrix")
plt.show()
```

```
TEST PREDICTION:
 [0 1 1]
TEST PREDICTION PROBABILITY:
 [[1. 0.]
 [0. 1.]
 [0. 1.]]
Test log loss of the model is :  75.6248 %
Train log loss of the model is :  82.5393 %
==================================================
Test log loss of the model is :  5.2119
Train log loss of the model is :  0.2579
==================================================
Test f1 score of the model is :  79.3976
Train f1 score of the model is :  84.4911
```



- 1158 data points which belongs to fake news are classified correctly.
- 86 data points which belongs to real news category are misclassified as fake news.
- 899 data points which belongs to fake news category are misclassified as real news.
- 1898 data points which belongs to real news are classified correctly.

In [ ]:

Decision Tree Classifier

In [90]:

```
from sklearn.tree import DecisionTreeClassifier
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.Randomize
from sklearn.model_selection import RandomizedSearchCV
```

In [117…

```
model = DecisionTreeClassifier()
```

In [118…

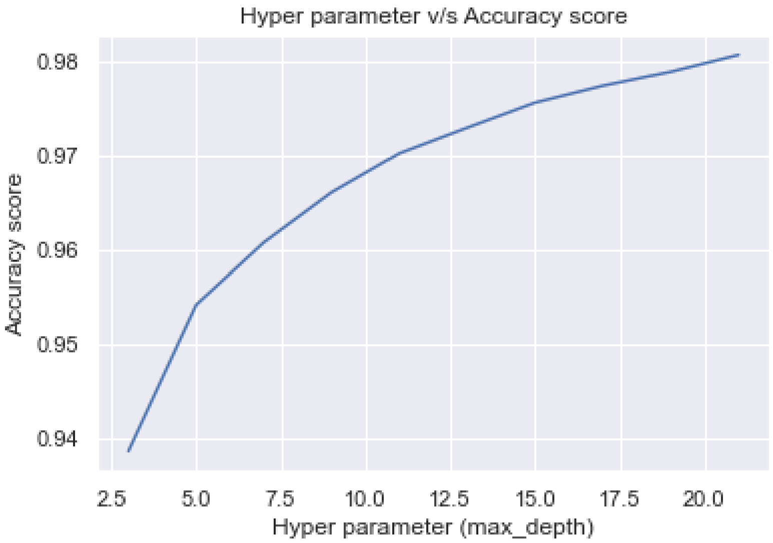
```
hyperparameter = {"max_depth" : [3,5,7,9,11,13,15,17,19,21]}
search = RandomizedSearchCV(model, hyperparameter, scoring = "accuracy", random_stat
search.fit(x_tr,y_train)
```

Out[118…

```
▸ RandomizedSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

In [119…

```
plt.plot(hyperparameter["max_depth"], search.cv_results_['mean_test_score'])
plt.xlabel("Hyper parameter (max_depth)")
plt.ylabel("Accuracy score")
plt.title("Hyper parameter v/s Accuracy score")
plt.show()
```



In [120…

```
search.cv_results_
```

Out[120…

```
{'mean_fit_time': array([ 9.63867116, 11.72168837, 13.53531256, 14.75545163, 15.4415
7877,
        16.33877888, 17.65191264, 18.12520823, 18.52821202, 19.46276317]),
 'std_fit_time': array([0.26887928, 0.19533027, 0.40742154, 0.26611401, 0.61157337,
        0.34445686, 0.55648399, 0.21417247, 0.41755096, 0.44823362]),
 'mean_score_time': array([0.02299848, 0.02217112, 0.02457943, 0.01999717, 0.0206461
4,
        0.02136021, 0.0216804 , 0.02044678, 0.02290483, 0.02054877]),
 'std_score_time': array([6.45042094e-03, 2.66942606e-03, 6.48007279e-03, 3.18948102
e-05,
        9.12627598e-04, 1.09920517e-03, 2.93394565e-03, 2.05235518e-03,
        1.80627660e-03, 2.77741039e-03]),
 'param_max_depth': masked_array(data=[3, 5, 7, 9, 11, 13, 15, 17, 19, 21],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 3},
  {'max_depth': 5},
  {'max_depth': 7},
  {'max_depth': 9},
  {'max_depth': 11},
  {'max_depth': 13},
  {'max_depth': 15},
  {'max_depth': 17},
  {'max_depth': 19},
  {'max_depth': 21}],
 'split0_test_score': array([0.93257037, 0.94772657, 0.95515002, 0.96102691, 0.96381
07 ,
        0.96628518, 0.97030622, 0.97401794, 0.9758738 , 0.97618311]),
 'split1_test_score': array([0.94370554, 0.96040829, 0.96628518, 0.97061553, 0.97525
518,
        0.97834828, 0.98020414, 0.98082277, 0.98360656, 0.98453449]),
 'split2_test_score': array([0.93595297, 0.95451733, 0.96008663, 0.96627475, 0.97029
703,
        0.97277228, 0.97462871, 0.97524752, 0.97617574, 0.98019802]),
 'split3_test_score': array([0.93997525, 0.95420792, 0.96039604, 0.96627475, 0.97029
703,
        0.9737005 , 0.97710396, 0.97834158, 0.9789604 , 0.98112624]),
 'split4_test_score': array([0.94059406, 0.95358911, 0.96163366, 0.96596535, 0.97122
525,
        0.97339109, 0.97555693, 0.97834158, 0.97957921, 0.98112624]),
 'mean_test_score': array([0.93855964, 0.95408984, 0.96071031, 0.96603146, 0.9701770
4,
        0.97289947, 0.97555999, 0.97735428, 0.97883914, 0.98063362]),
 'std_test_score': array([0.00388044, 0.00402148, 0.00355911, 0.00303914, 0.0036727
5,
        0.00385585, 0.00323794, 0.00243103, 0.0027992 , 0.00267204]),
 'rank_test_score': array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])}
```

In [121…

```
search.best_params_
```

Out[121…

```
{'max_depth': 21}
```

In [122…

```
model = DecisionTreeClassifier(max_depth = search.best_params_["max_depth"])
model.fit(x_tr,y_train)
```

Out[122…

```
▾       DecisionTreeClassifier
DecisionTreeClassifier(max_depth=21)
```

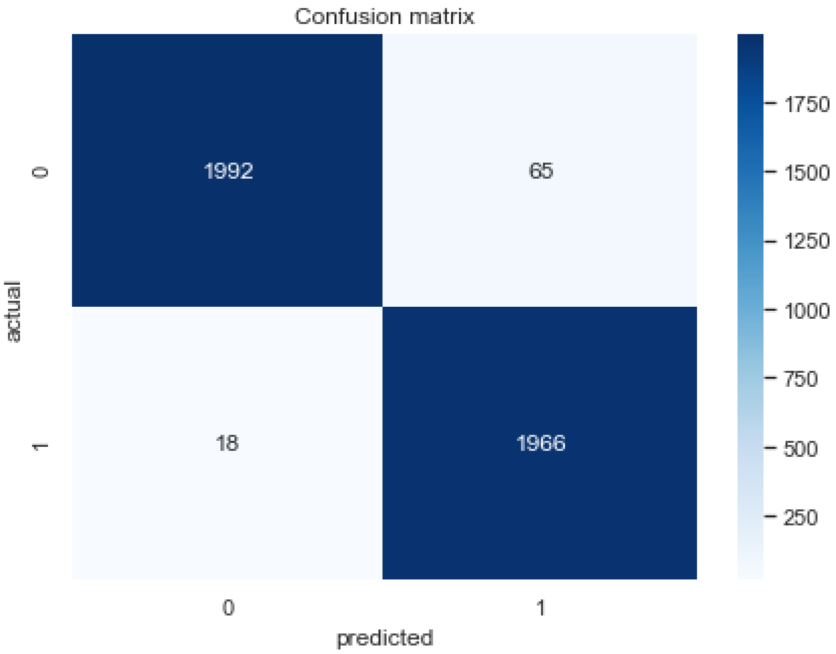
In [123…

```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [124…

```
y_test_pred = model.predict(x_te)
accuracy = accuracy_score(y_test,y_test_pred)
print("Test accuracy of the model is : ", np.round(accuracy*100,4), "%")
train_pred = model.predict(x_tr)
train_accuracy = accuracy_score(y_train,train_pred)
print("Train accuracy of the model is : ", np.round(train_accuracy*100,4), "%")
#===================================================================================
cm1 = confusion_matrix(y_test,y_test_pred)
plt.figure(figsize=(7,5))
sns.heatmap(cm1, annot=True,fmt="d",cmap='Blues')
plt.xlabel("predicted")
plt.ylabel("actual")
plt.title("Confusion matrix")
plt.show()
```

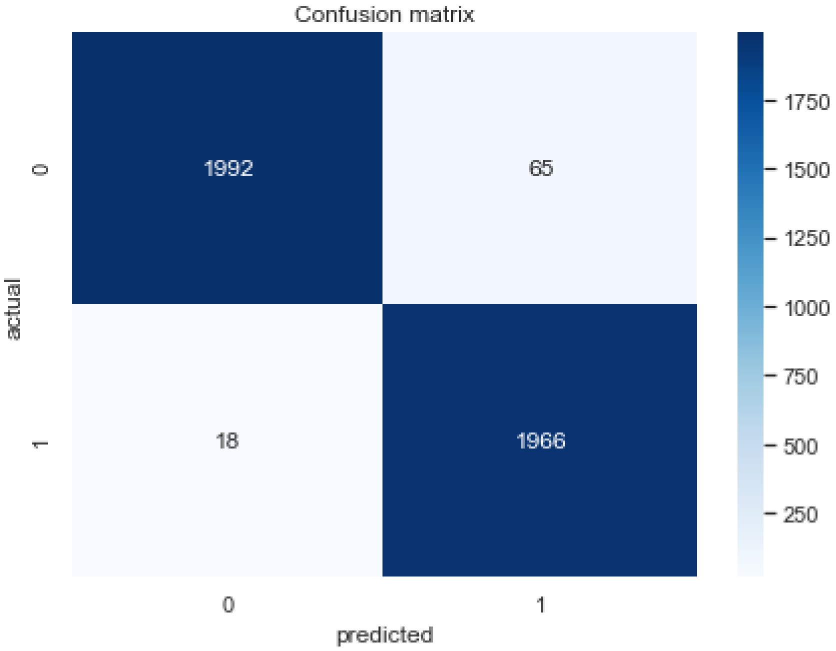
```
Test accuracy of the model is :  97.9461 %
Train accuracy of the model is :  98.744 %
```



In [125…

```
y_test_pred = model.predict(x_te)
print("TEST PREDICTION: \n",y_test_pred[:3])
#predicting probability
y_test_pred_proba = model.predict_proba(x_te)
print("TEST PREDICTION PROBABILITY: \n",y_test_pred_proba[:3])


train_pred = model.predict(x_tr)
#predicting probability
train_pred_proba = model.predict_proba(x_tr)


accuracy = accuracy_score(y_test,y_test_pred)
print("Test log loss of the model is : ", np.round(accuracy*100,4), "%")
train_accuracy = accuracy_score(y_train,train_pred)
print("Train log loss of the model is : ", np.round(train_accuracy*100,4), "%")

print("="*50)

test_logloss = log_loss(y_test,y_test_pred_proba)
print("Test log loss of the model is : ", np.round(test_logloss,4))
train_logloss = log_loss(y_train,train_pred_proba)
print("Train log loss of the model is : ", np.round(train_logloss,4))

print("="*50)

test_f1 = f1_score(y_test,y_test_pred)
print("Test f1 score of the model is : ", np.round(test_f1*100,4))
train_f1 = f1_score(y_train,train_pred)
print("Train f1 score of the model is : ", np.round(train_f1*100,4))


cm1 = confusion_matrix(y_test,y_test_pred)
plt.figure(figsize=(7,5))
sns.heatmap(cm1, annot=True,fmt="d",cmap='Blues')
plt.xlabel("predicted")
plt.ylabel("actual")
plt.title("Confusion matrix")
plt.show()
```

```
TEST PREDICTION:
 [1 1 1]
TEST PREDICTION PROBABILITY:
 [[0.02566047 0.97433953]
 [0.02566047 0.97433953]
 [0.02566047 0.97433953]]
Test log loss of the model is :  97.9461 %
Train log loss of the model is :  98.744 %
==================================================
Test log loss of the model is :  0.3016
Train log loss of the model is :  0.0584
==================================================
Test f1 score of the model is :  97.9328
Train f1 score of the model is :  98.7206
```



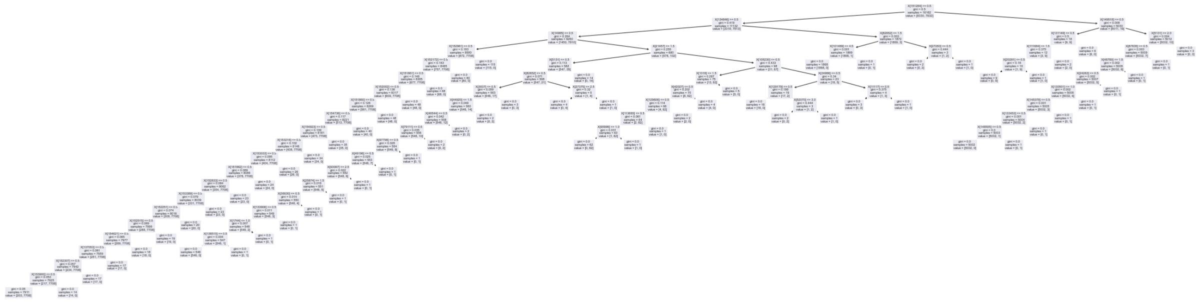
- 1996 data points which belongs to fake news are classified correctly.
- 17 data points which belongs to real news category are misclassified as fake news.
- 61 data points which belongs to fake news category are misclassified as real news.
- 1967 data points which belongs to real news are classified correctly.

In [126…

```
from sklearn import tree
```

In [127…

```
plt.figure(figsize=(40,10))
tree.plot_tree(model)
plt.show()
```



In [ ]:

In [169…

```
model.feature_importances_
```

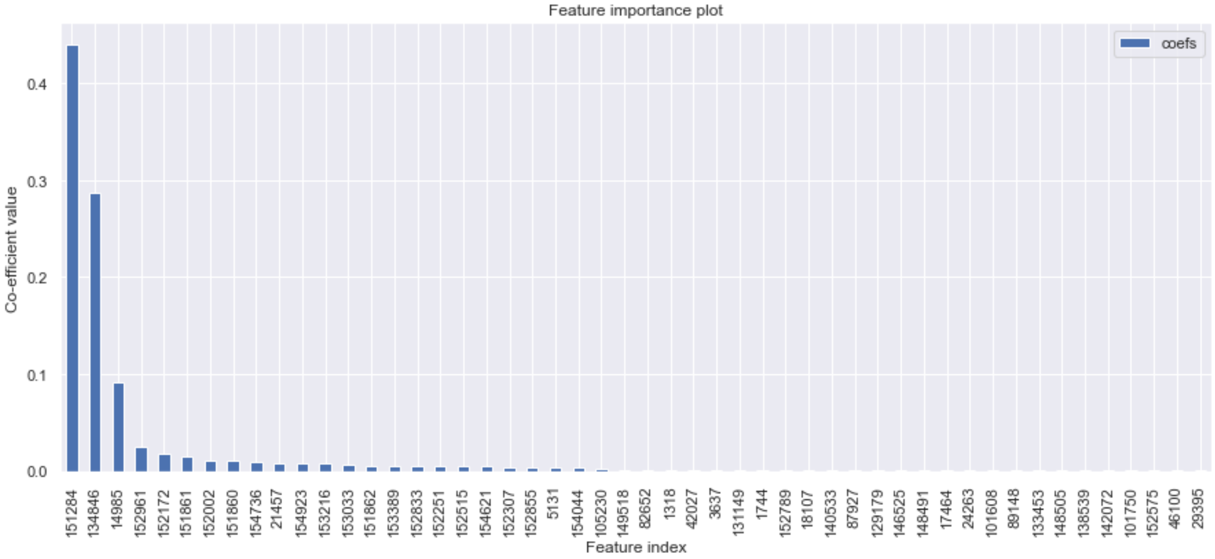
Out[169…

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

In [170…

```
coefs_df = pd.DataFrame()
coefs_df["coefs"] = model.feature_importances_
```

```
coefs_df["abs_coefs"] = abs(model.feature_importances_)
coefs_df.sort_values(by="abs_coefs", inplace=True, ascending=False)
top_coefs = coefs_df.head(50)
top_coefs
top_coefs["feature index"] = top_coefs.index
top_coefs[["coefs"]].plot(kind="bar",figsize=(15, 6))
plt.title("Feature importance plot")
plt.xlabel("Feature index")
plt.ylabel("Co-efficient value")
plt.show()
```



Random Forest Classifier

In [171…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
```

In [172…

```
model = RandomForestClassifier()
```

In [173…

```
hyperparameter = {"n_estimators" : [50,100,150,200]}
search = RandomizedSearchCV(model, hyperparameter, scoring = "accuracy", random_stat
search.fit(x_tr,y_train)
```

Out[173…

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [174…

```
search.cv_results_
```

Out[174…

```
{'mean_fit_time': array([132.15902462, 269.25143547, 401.6874825 , 491.48179626]),
 'std_fit_time': array([ 1.58724338,  3.5085762 ,  4.6406422 , 54.04581438]),
 'mean_score_time': array([0.70720143, 1.4028758 , 2.04618077, 2.38511906]),
 'std_score_time': array([0.02881821, 0.03492622, 0.074619  , 0.61941251]),
 'param_n_estimators': masked_array(data=[50, 100, 150, 200],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_estimators': 50},
  {'n_estimators': 100},
  {'n_estimators': 150},
  {'n_estimators': 200}],
 'split0_test_score': array([0.9396845 , 0.94834519, 0.95112898, 0.95607795]),
 'split1_test_score': array([0.94215899, 0.94989174, 0.94896381, 0.95484071]),
 'split2_test_score': array([0.95080446, 0.95420792, 0.95575495, 0.95730198]),
 'split3_test_score': array([0.93997525, 0.94461634, 0.94647277, 0.95482673]),
 'split4_test_score': array([0.93966584, 0.94647277, 0.95420792, 0.95420792]),
 'mean_test_score': array([0.94245781, 0.94870679, 0.95130569, 0.95545106]),
 'std_test_score': array([0.00427563, 0.0032717 , 0.0033791 , 0.00110694]),
 'rank_test_score': array([4, 3, 2, 1])}
```

In [175…

```
search.best_params_
```

Out[175…

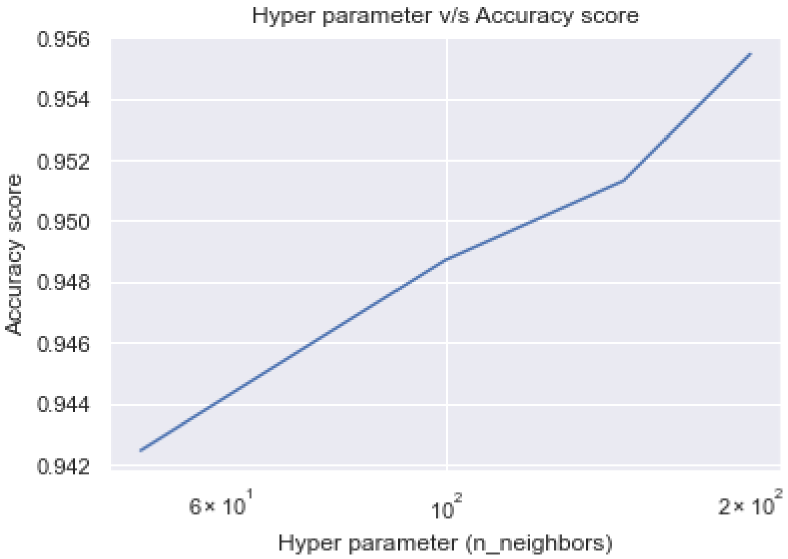
```
{'n_estimators': 200}
```

In [ ]:

```
search.best_estimator_
```

In [176…

```
plt.plot(hyperparameter["n_estimators"], search.cv_results_['mean_test_score'])
plt.xscale("log")
plt.xlabel("Hyper parameter (n_neighbors)")
plt.ylabel("Accuracy score")
plt.title("Hyper parameter v/s Accuracy score")
plt.show()
```



In [177…

```
model = RandomForestClassifier(n_estimators = search.best_params_["n_estimators"])
model.fit(x_tr,y_train)
```

Out[177…

```
▾          RandomForestClassifier

RandomForestClassifier(n_estimators=200)
```

In [ ]:

```
y_test_pred = model.predict(x_te)
accuracy = accuracy_score(y_test,y_test_pred)
print("Test accuracy of the model is : ", np.round(accuracy*100,4), "%")
train_pred = model.predict(x_tr)
train_accuracy = accuracy_score(y_train,train_pred)
print("Train accuracy of the model is : ", np.round(train_accuracy*100,4), "%")
#===================================================================================
cm2 = confusion_matrix(y_test,y_test_pred)
plt.figure(figsize=(7,5))
sns.heatmap(cm2, annot=True,fmt="d",cmap='Blues')
plt.xlabel("predicted")
plt.ylabel("actual")
plt.title("Confusion matrix")
plt.show()
```

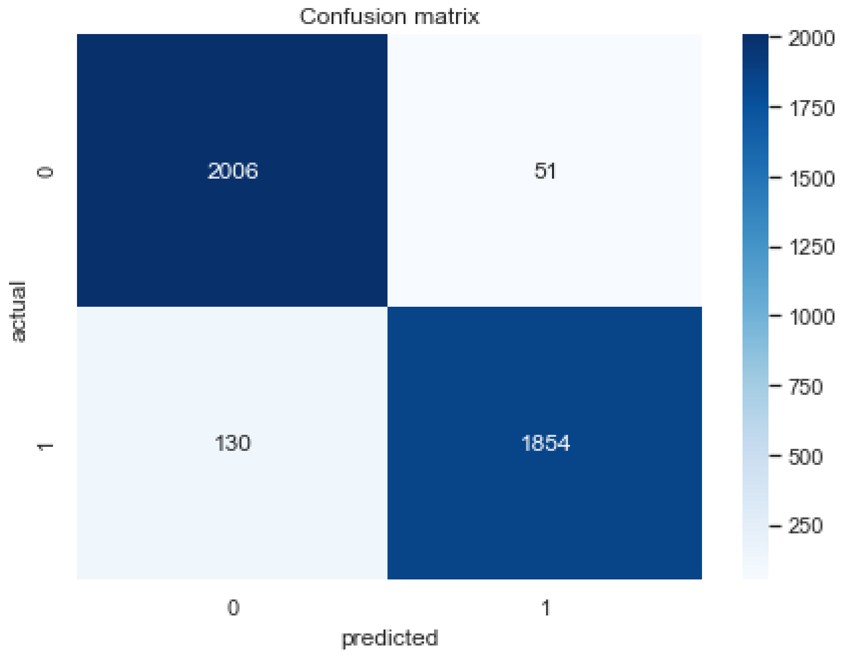
- 2009 data points which belongs to fake news are classified correctly.

- 134 data points which belongs to real news category are misclassified as fake news.
- 48 data points which belongs to fake news category are misclassified as real news.
- 1850 data points which belongs to real news are classified correctly.

In [178…

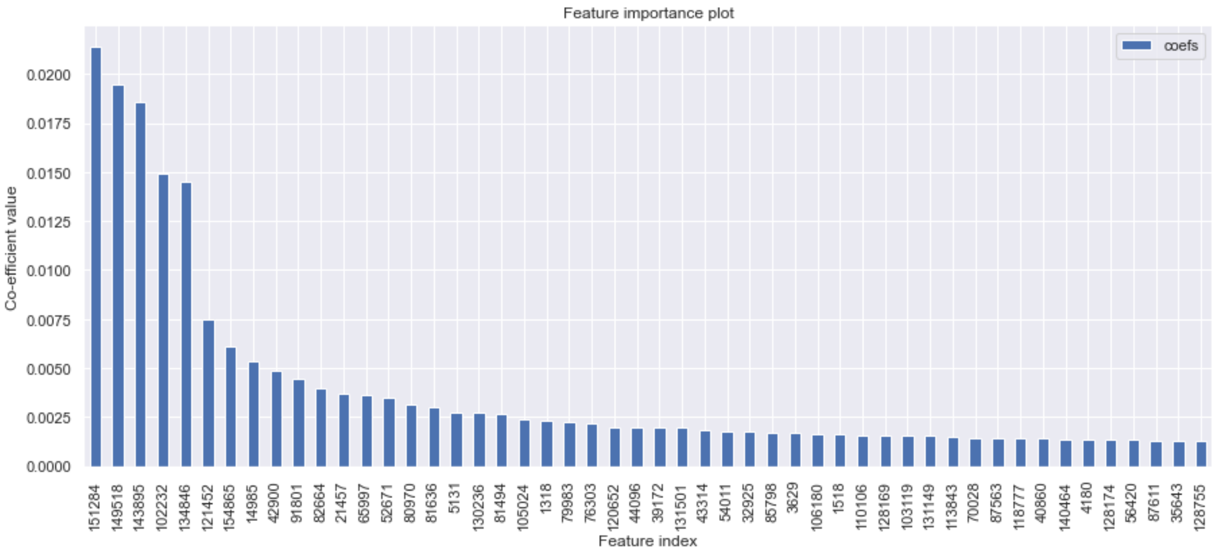
```
y_test_pred = model.predict(x_te)
print("TEST PREDICTION: \n",y_test_pred[:3])
#predicting probability
y_test_pred_proba = model.predict_proba(x_te)
print("TEST PREDICTION PROBABILITY: \n",y_test_pred_proba[:3])


train_pred = model.predict(x_tr)
#predicting probability
train_pred_proba = model.predict_proba(x_tr)


accuracy = accuracy_score(y_test,y_test_pred)
print("Test log loss of the model is : ", np.round(accuracy*100,4), "%")
train_accuracy = accuracy_score(y_train,train_pred)
print("Train log loss of the model is : ", np.round(train_accuracy*100,4), "%")

print("="*50)

test_logloss = log_loss(y_test,y_test_pred_proba)
print("Test log loss of the model is : ", np.round(test_logloss,4))
train_logloss = log_loss(y_train,train_pred_proba)
print("Train log loss of the model is : ", np.round(train_logloss,4))

print("="*50)

test_f1 = f1_score(y_test,y_test_pred)
print("Test f1 score of the model is : ", np.round(test_f1*100,4))
train_f1 = f1_score(y_train,train_pred)
print("Train f1 score of the model is : ", np.round(train_f1*100,4))


cm1 = confusion_matrix(y_test,y_test_pred)
plt.figure(figsize=(7,5))
sns.heatmap(cm1, annot=True,fmt="d",cmap='Blues')
plt.xlabel("predicted")
plt.ylabel("actual")
plt.title("Confusion matrix")
plt.show()
```

```
TEST PREDICTION:
 [0 1 1]
TEST PREDICTION PROBABILITY:
 [[0.585 0.415]
 [0.31  0.69 ]
 [0.165 0.835]]
Test log loss of the model is :  95.5209 %
Train log loss of the model is :  100.0 %
==================================================
Test log loss of the model is :  0.3061
Train log loss of the model is :  0.0974
==================================================
Test log loss of the model is :  95.3458
Train log loss of the model is :  100.0
```



In [179…

```
coefs_df = pd.DataFrame()
coefs_df["coefs"] = model.feature_importances_
coefs_df["abs_coefs"] = abs(model.feature_importances_)
coefs_df.sort_values(by="abs_coefs", inplace=True, ascending=False)
top_coefs = coefs_df.head(50)
top_coefs
top_coefs["feature index"] = top_coefs.index
top_coefs[["coefs"]].plot(kind="bar",figsize=(15, 6))
plt.title("Feature importance plot")
plt.xlabel("Feature index")
plt.ylabel("Co-efficient value")
plt.show()
```



In [187…

In [ ]:

In [ ]:

In [ ]: