National College of Ireland

Configuration Manual

MSc Research Project
MSCTOPAI

Arthur Ryan
Student ID: 23333138

School of Computing
National College of Ireland

Supervisor: Professor Sheresh Zahoor

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Arthur Ryan
Student ID:	23333138
Programme:	MSCAITOP **Year:** 2024
Module:	MSCAITOP
Lecturer:	Professor Sheresh Zahoor
Submission Due Date:	12/8/2024
Project Title:	Co-pilot widget for assisting the public in processing US presidential political candidate tweets from Twitter in 2024 US elections candidate choice through sarcasm and stance detection
Word Count:	608 **Page Count:** 72

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: *Arthur Ryan*

Date: 11/8/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Arthur Ryan
Student ID: 23333138

1 Introduction

This document summarises the specification of the software and hardware for the project 'Co-pilot widget for assisting the public in processing US presidential political candidate tweets from Twitter in 2024 US elections candidate choice through sarcasm and stance detection'

2 Environment

Python 3.0

3 Libraries

The following libraries are required for running the project code notebooks.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
!pip install gensim # Gensim is an open-source library for unsupervised
topic modeling and natural language processing
import nltk
nltk.download('punkt')
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
import nltk
import re
from nltk.corpus import stopwords
#!pip install nltk
import nltk
nltk.download("stopwords")
import seaborn as sns
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS

import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
```

For latency measurements:

```
import time
```

For Sarcasm and Stance:

```
import keras
#import tensorflow as tf
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding,
Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D,
Bidirectional
from keras.models import Sequential
from keras.metrics import F1Score
```

For Charting:

```
import io
import xml.etree.ElementTree as ET
import time

from matplotlib.lines import Line2D
from matplotlib.markers import MarkerStyle
from matplotlib.transforms import Affine2D
```

4 Software Specification

Google Colab: a web environment that uses Google Cloud
Gmail account: to access Google Colab

5 Hardware Specification

Hardware: Hewlard Packard (HP) Z440
Memory: 32GB RAM
Storage: 1TB SSD
Processor: 4 CPU each of 1.6 Ghz

6 Code Section

6.0 Code sources

Fake News

https://www.kaggle.com/code/paramarthasengupta/fake-news-detector-eda-prediction-99

Sarcasm

https://www.kaggle.com/datasets/deepnews/fakenews-reddit-comments/data

Stance

https://www.kaggle.com/datasets/arashnic/7-nlp-tasks-with-tweets

6.1 Truth-Fake News detection

Data Cleaning

- use of gensim library to preprocess data
- screen out stop words
- screen out words less than length 2

```
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 2 and token not in stop_words:
            result.append(token)

    return result
```

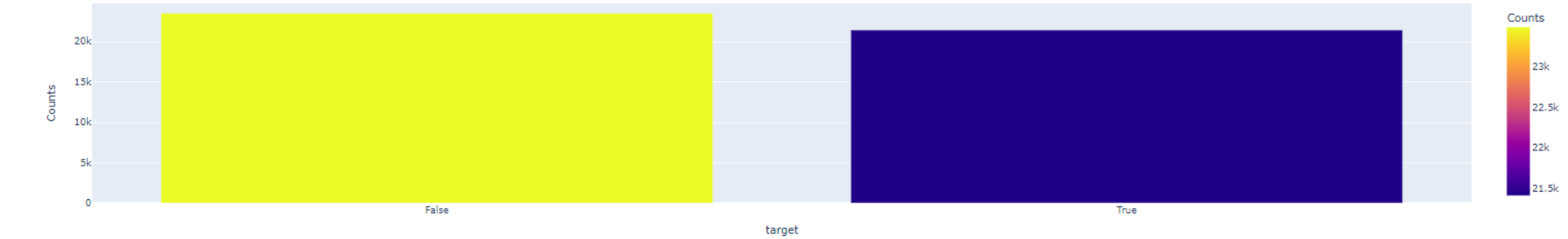
```
# Transforming the unmatching subjects to the same notation
df.subject=df.subject.replace({'politics':'PoliticsNews','politicsNews':'PoliticsNews'})
```

Additional data cleaning: transforming generic references to proper nouns

Exploratory Data Analysis (EDA)

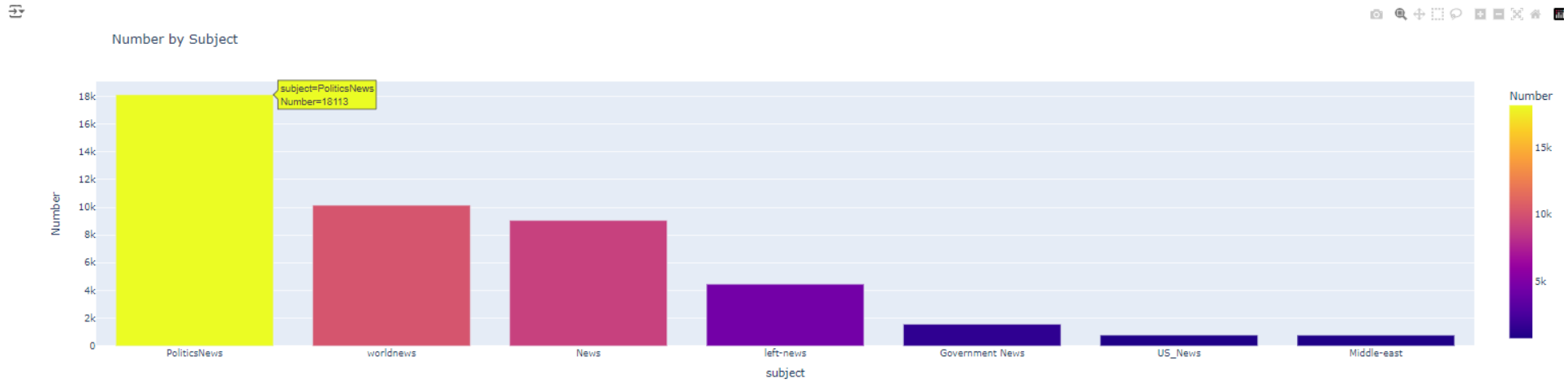
Histogram for balance of true v fake news

```
sub_tf_df=df.groupby('target').apply(lambda x:x['title'].count()).reset_index(name='Counts')
sub_tf_df.target.replace({0:'False',1:'True'},inplace=True)
fig = px.bar(sub_tf_df, x="target", y="Counts",
             color='Counts', barmode='group',
             height=400)
fig.show()
```

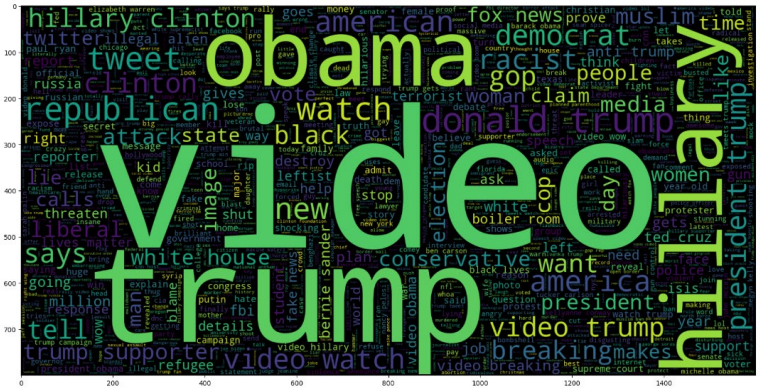


Historgam of subjects covered

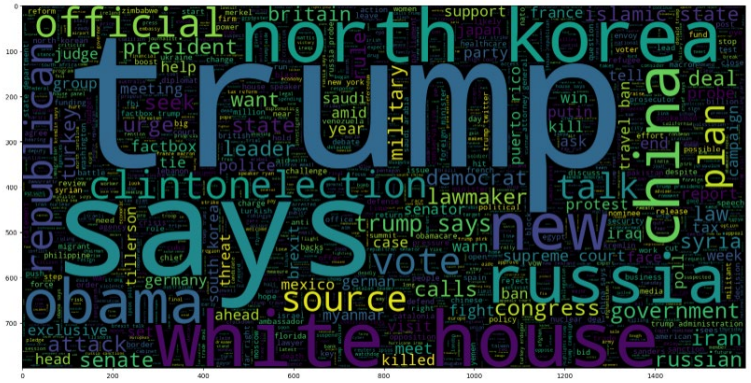
```
sub_check = df.groupby('subject').apply(lambda x:x['title'].count()).reset_index(name='Number') # .sort()
sub_check = sub_check.sort_values(by=['Number'], ascending=False)
fig=px.bar(sub_check, x='subject', y='Number', color='Number', title='Number by Subject')
fig.show()
```



Below is Word Cloud of True News words



Below is Word Cloud of Fake News words

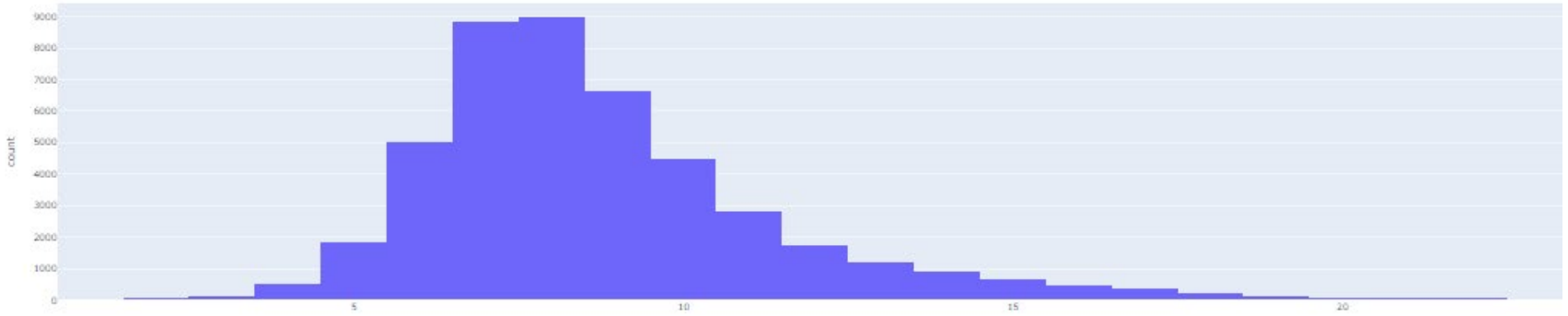


To understand dimensions of data being examined a graph of the title length was made

Measuring length of Titles

```
maxlen = -1
for doc in df.clean_joined_title:
    tokens = nltk.word_tokenize(doc)
    if(maxlen<len(tokens)):
        maxlen = len(tokens)
print("The maximum number of words in a title is =", maxlen)
fig = px.histogram(x = [len(nltk.word_tokenize(x)) for x in df.clean_joined_title], nbins = 50)
fig.show()
```

```
The maximum number of words in a title is = 34
```



Prediction with TF-iDF + Logistic Regression (only using title)

Using only the Title to predict

1st Prediction - based on Title(i.e., max 34 words)

```
[ ]  1 X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_title, df.target, test_size = 0.2,random_state=2)
     2 vec_train = CountVectorizer().fit(X_train)
     3 X_vec_train = vec_train.transform(X_train)
     4 X_vec_test = vec_train.transform(X_test)
```

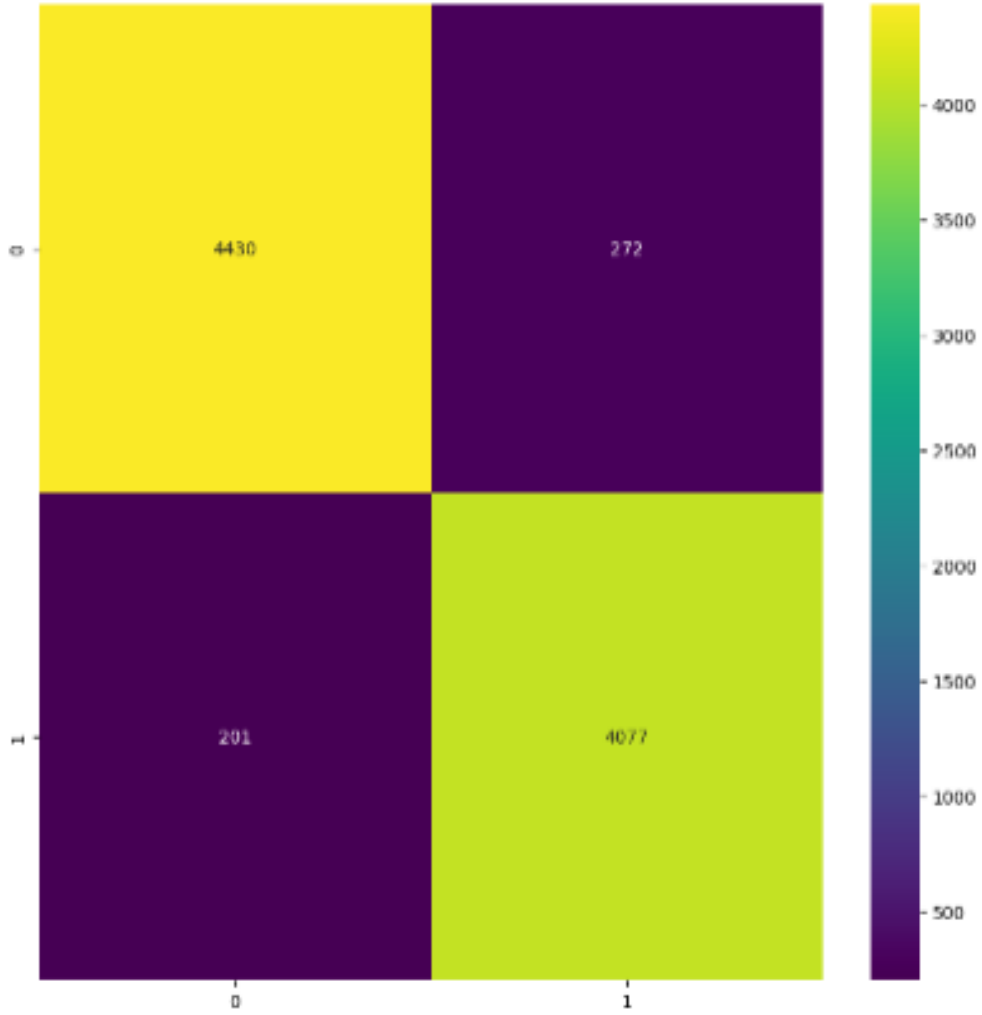
```
[ ]  1 model = LogisticRegression(C=2)
     2 model.fit(X_vec_train, y_train)
     3 predicted_value = model.predict(X_vec_test)
     4 accuracy_value = roc_auc_score(y_test, predicted_value)
     5 print("Accuracy score = ", accuracy_value)
     6 f1_score_value = f1_score(y_test, predicted_value, average='macro')
     7 print("F1_score = ", f1_score_value)
     8 precision_score_value = precision_score(y_test, predicted_value, average='macro')
     9 print("Precison score = ", precision_score_value)
```

```
Accuracy score =  0.9475838516986892
F1_score =  0.9472458762778595
Precison score =  0.9470268669868982
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning:
```

1st Confusion Matrix

```
[ ]  1 cm = confusion_matrix(list(y_test), predicted_value)
     2 plt.figure(figsize = (10, 10))
     3 sns.heatmap(cm, annot = True,fmt='g',cmap='viridis')
```

```
<Axes: >
```



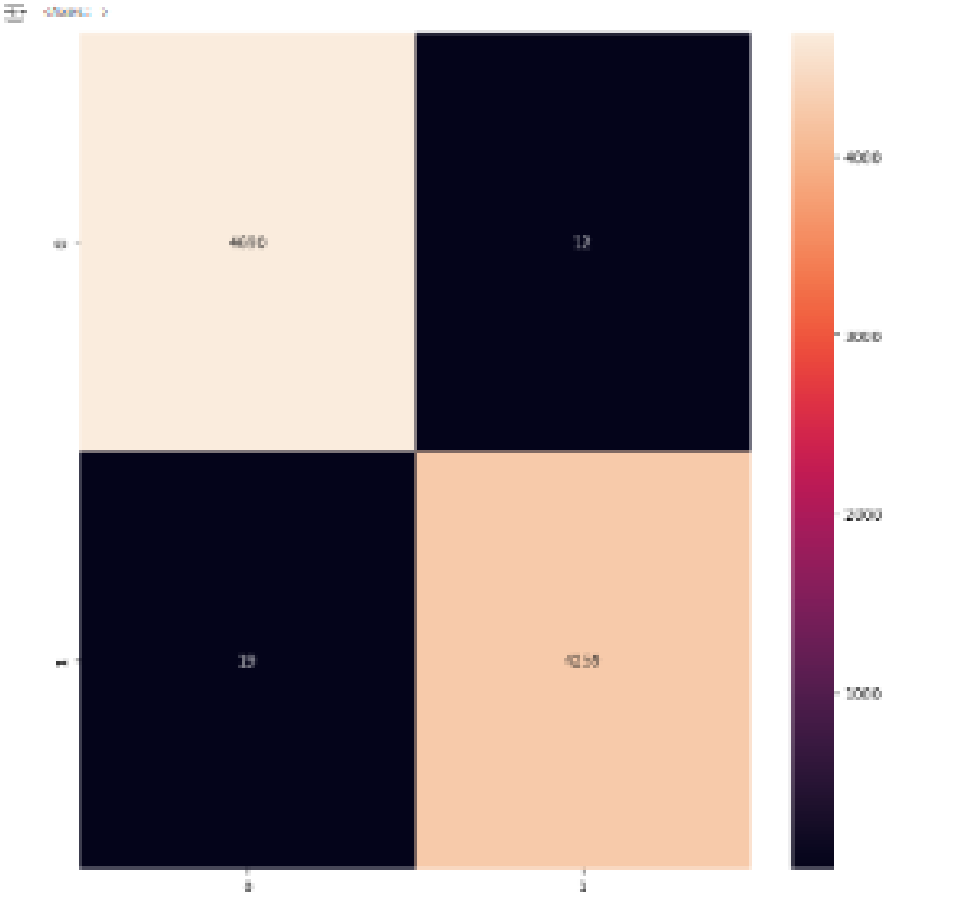
2nd Prediction based on Text (i.e., more words than just title)

```
#prediction based on Text 100-200 words vs only Title 8 words max

X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_text, df.target, test_size = 0.2,random_state=2)
vec_train = CountVectorizer().fit(X_train)
X_vec_train = vec_train.transform(X_train)
X_vec_test = vec_train.transform(X_test)
model = LogisticRegression(C=2.5)
model.fit(X_vec_train, y_train)
predicted_value = model.predict(X_vec_test)
accuracy_value = roc_auc_score(y_test, predicted_value)
print(accuracy_value)
print("Accuracy score = ", accuracy_value)
f1_score_value = f1_score(y_test, predicted_value, average='macro')
print("F1_score = ", f1_score_value)
precision_score_value = precision_score(y_test, predicted_value, average='macro')
print("Precison score = ", precision_score_value)
```

```
0.996503283394869
Accuracy score =  0.996503283394869
F1_score =  0.9965399136276033
Precison score =  0.996577763310009
```

2nd The Confusion Matrix

```
prediction = []
for i in range(len(predicted_value)):
    if predicted_value[i].item() > 0.5:
        prediction.append(1)
    else:
        prediction.append(0)
cm = confusion_matrix(list(y_test), prediction)
plt.figure(figsize = (10, 10))
sns.heatmap(cm, annot = True,fmt='g')
```



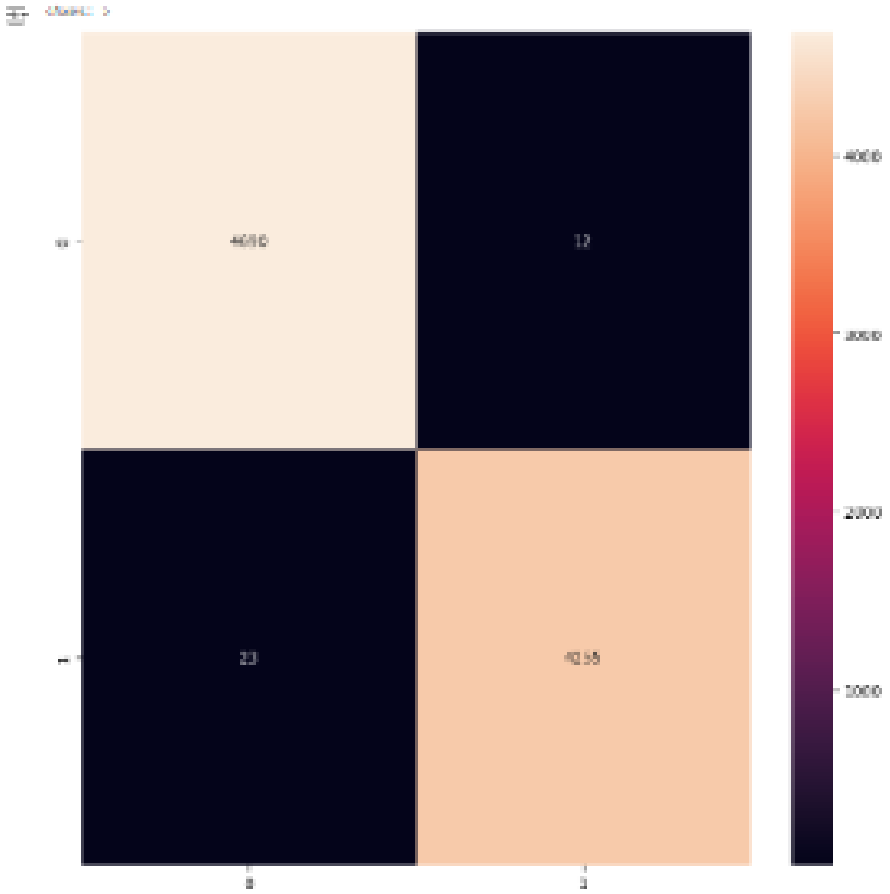
3rd - Prediction - Title and Text combined

```
[ ]  1 df['clean_final'] = df['original'].apply(preprocess)
     2 df['clean_joined_final']=df['clean_final'].apply(lambda x:" ".join(x))
```

```
     1 #prediction based on Text 100-200 words vs only Title 8 words max
     2
     3 X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_final, df.target, test_size = 0.2,random_state=2)
     4 vec_train = CountVectorizer().fit(X_train)
     5 X_vec_train = vec_train.transform(X_train)
     6 X_vec_test = vec_train.transform(X_test)
     7 model = LogisticRegression(C=2.5)
     8 model.fit(X_vec_train, y_train)
     9 predicted_value = model.predict(X_vec_test)
    10 accuracy_value = roc_auc_score(y_test, predicted_value)
    11 print(accuracy_value)
    12 print("Accuracy score = ", accuracy_value)
    13 f1_score_value = f1_score(y_test, predicted_value, average='macro')
    14 print("F1_score = ", f1_score_value)
    15 precision_score_value = precision_score(y_test, predicted_value, average='macro')
    16 print("Precison score = ", precision_score_value)
```

```
0.996035775213476
Accuracy score =  0.996035775213476
F1_score =  0.9960932826649207
Precison score =  0.9961538004445567
```

```
prediction = []
for i in range(len(predicted_value)):
    if predicted_value[i].item() > 0.5:
        prediction.append(1)
    else:
        prediction.append(0)
cm = confusion_matrix(list(y_test), prediction)
plt.figure(figsize = (30, 30))
sns.heatmap(cm, annot = True,fmt='g')
```



Above establishes baseline from historic tweet data

Build out data i.e., record predictions to feature columns in ‘memory’ / csv file to be passed to next metric notebook

Build out dataset

```
[ ]
# Build out Dataset - make predictions and assign to target column in dataframe

import numpy as np
np.set_printoptions(threshold=np.inf)

#X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_final, df.target, test_size = 0.2,random_state=0)
#X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_text, df.target, test_size = 0.2,random_state=0)

#vec_train = CountVectorizer().fit(X_train)
#X_vec_train = vec_train.transform(X_train)
#X_vec_test = vec_train.transform(X_test)

# ->> put in new tweets here ->>X_vec_test = vec_train.transform(X_test)

#model = LogisticRegression(C=3)
#model.fit(X_vec_train, y_train)

# Vectorising Text column (vs Title or Title and Text) to input to model previously traing on the static data inorder to make predictions on the new X data
X_vec_train = vec_train.transform(df_new['clean_joined_text'])
predicted_value = model.predict(X_vec_test) # it is these predicted values that are to have the graphics to be created from
#print(predicted_value.reshape(-1))

df_new['predicted_true_fake'] = 0
```

```
df_new['predicted_true_fake'] = 0

# for i in range(len(df_new)):
#   if predicted_value[i] == 1:
#     #df_new['target'].iloc[i] = 1
#     df_new.loc[i, 'predicted_true_fake'] = 1
#   # elif predicted_value.iloc[i] == 0:
#   #   df_new['Fake'] = 0

i = 0
for p in predicted_value:
  if p == 1:
    #df_new['target'].iloc[i] = 1
    df_new.loc[i, 'predicted_true_fake'] = 1
    i += 1
  # elif predicted_value.iloc[i] == 0:
  #   df_new['Fake'] = 0


print(df_new['predicted_true_fake'])

#df_new.info()
#df_new = df_new[0,1,2,3,4]
#df2 = df1[['A', 'C']].copy()
df_new = df_new[['id', 'content', 'clean_text', 'clean_joined_text', 'predicted_true_fake']].copy()
#df.drop(df.columns[[5:]], axis=1, inplace=True)
df_new.info()

# ADD section for EDA again here

# NOT relevant as not in training mode but prediction # accuracy_value = roc_auc_score(y_test, predicted_value)
# NOT relevant as not in training mode but prediction # print(accuracy_value)
```

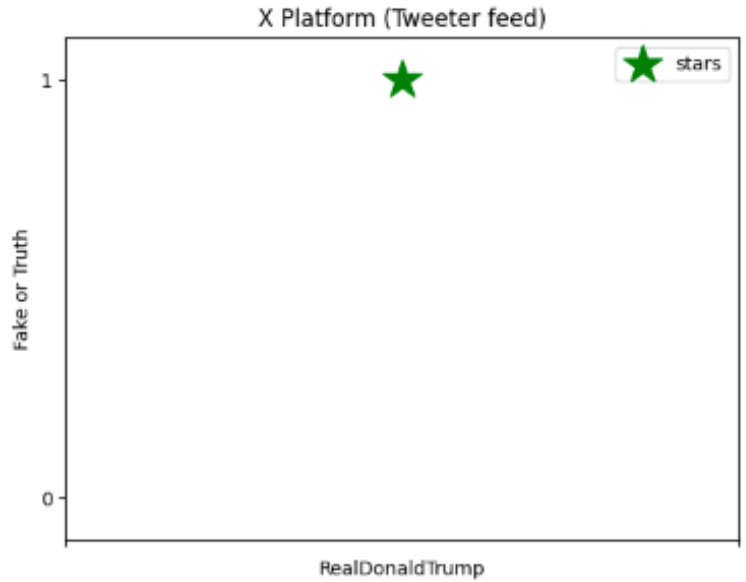
```
0      1
```

Sample graphic to represent prediction of Fake v True Tweet

```python
import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter

# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = df_new['predicted_true_fake'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]


if y == 0:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "rasberry", color= "red",
              marker= "X", s=450)

if y == 1:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "stars", color= "green",
              marker= "*", s=450)


# x-axis label
plt.xlabel('RealDonaldTrump')
# frequency label
plt.ylabel('Fake or Truth')
# plot title
plt.title('X Platform (Tweeter feed)')
# showing legend
plt.legend()
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
```

```
plt.title('X Platform (Tweeter feed)')
# showing legend
plt.legend()
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
plt.xticks([-1, 1])
plt.yticks([0, 1])

# function to show the plot
plt.show()
```

```
<ipython-input-48-a158904e4e60>:30: UserWarning:

Attempting to set identical low and high xlims makes transformation singular; automatically expanding.
```



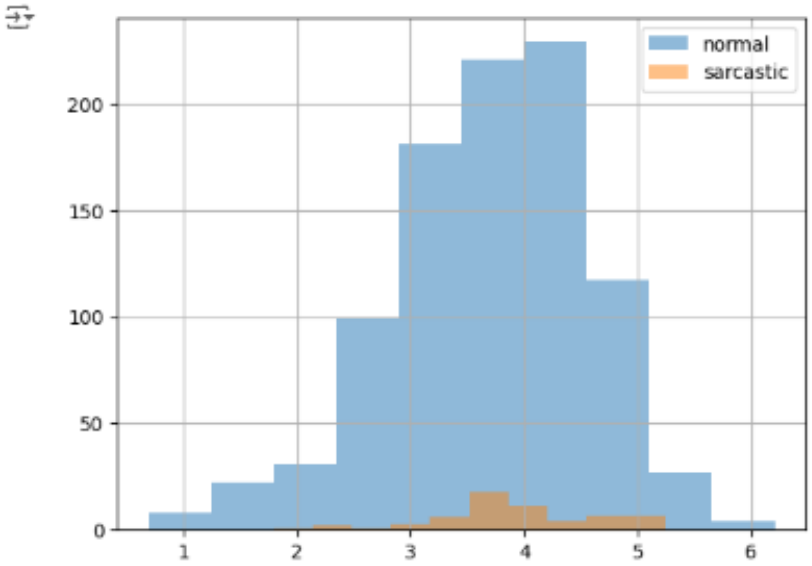
Create memory for metrics i.e., save a file and add feature 1, 2(in next notebook), 3(in next notebook after notebook 2)

```
# save prediction values to csv file to be shared with other metrics to follow i.e., stance and mis/disinformation

df_new.to_csv('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', encoding="utf-8")

# #df_new.save_csv("data_to_graph")

# import csv

# # field names
# fields = ['Author', 'Text', 'clean_text', 'clean_joined_text',  'target']  # ['Name', 'Branch', 'Year', 'CGPA']

# # data rows of csv file
# rows = df_new # [ ['Nikhil', 'COE', '2', '9.0'],
#         #  ['Sanchit', 'COE', '2', '9.1'],
#         #  ['Aditya', 'IT', '2', '9.3'],
#         #  ['Sagar', 'SE', '1', '9.5'],
#         #  ['Prateek', 'MCE', '3', '7.8'],
#         #  ['Sahil', 'EP', '2', '9.1']]

# # name of csv file
# filename = "/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv"

# # writing to csv file
# with open(filename, 'w') as csvfile:
#     # creating a csv writer object
#     csvwriter = csv.writer(csvfile)

#     # writing the fields
#     csvwriter.writerow(fields)

#     # writing the data rows
#     csvwriter.writerows(rows


# df_new.info()
# df_new.predicted_true_fake
```

6.2 Sarcasm

Exploratory data analysis (EDA)

```
[ ]  1 # histograph of sarcastic vs normal
     2
     3 train_df.loc[train_df['label'] == 0, 'comment'].str.len().apply(np.log1p).hist(label='normal', alpha=.5)
     4 train_df.loc[train_df['label'] == 1, 'comment'].str.len().apply(np.log1p).hist(label='sarcastic', alpha=.5)
     5 plt.legend();
```



Analysis of Data

All reddit comments of any size

```
     1 sub_df = train_df.groupby('subreddit')['label'].agg([np.size, np.mean, np.sum, np.median, np.std, np.min, np.max])
     2 sub_df.sort_values(by='sum', ascending=False).head(10)
```

subreddit	size	mean	sum	median	std	min	max
politics	68	0.102941	7	0.0	0.306141	0	1
The_Donald	50	0.100000	5	0.0	0.303046	0	1
leagueoflegends	14	0.285714	4	0.0	0.468807	0	1
wow	5	0.400000	2	0.0	0.547723	0	1
nfl	24	0.083333	2	0.0	0.282330	0	1
oddlysatisfying	3	0.666667	2	1.0	0.577350	0	1
nba	8	0.250000	2	0.0	0.462910	0	1
AskReddit	100	0.020000	2	0.0	0.140705	0	1
CFBOffTopic	1	1.000000	1	1.0	NaN	1	1
supergirlTV	1	1.000000	1	1.0	NaN	1	1

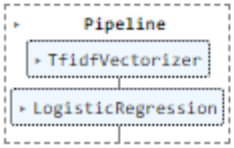
Create pipeline and train the TF-iDF + Logistic Regression model

Training the model

```
# build bigrams, put a limit on maximal number of features
# and minimal word frequency
tf_idf = TfidfVectorizer(ngram_range=(1, 2), max_features=50000, min_df=2)
# multinomial logistic regression a.k.a softmax classifier
logit = LogisticRegression(C=1, n_jobs=4, solver='lbfgs',
                           random_state=42, verbose=1)
# sklearn's pipeline
tfidf_logit_pipeline = Pipeline([('tf_idf', tf_idf),
                                 ('logit', logit)])
```

```
%%time
tfidf_logit_pipeline.fit(train_texts, y_train)
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
CPU times: user 115 ms, sys: 113 ms, total: 228 ms
Wall time: 2.34 s
```



Visualisations Word Clouds

```
# wordcloud

from wordcloud import WordCloud, STOPWORDS
```

```
wordcloud = WordCloud(background_color='black', stopwords = STOPWORDS,
                      max_words = 200, max_font_size = 100,
                      random_state = 17, width=800, height=400)
```

Prediction score from TF-iDF + Logistic Regression model

Reference base line Saracasm accuracy

```
[ ]  1 accuracy_value = accuracy_score(y_valid, valid_pred)
     2 print("Accuracy score = ", accuracy_value)
     3 f1_score_value = f1_score(y_valid, valid_pred, average='macro')
     4 print("F1_score = ", f1_score_value)
     5 precision_score_value = precision_score(y_valid, valid_pred, average='macro')
     6 print("Precison score = ", precision_score_value)
```

```
Accuracy score =  0.932
F1_score =  0.48240165631469983
Precison score =  0.466
```

Point TF-iDF + Logistic Regression model trained model at new data

Predict Sarcasm with Logistic regression based on new X(twitter) texts

```
  1 #ORIGINAL predicted_value = tfidf_logit_pipeline.predict(df_new_texts['Text'])
  2
  3 predicted_value = tfidf_logit_pipeline.predict(df_new_texts['content'].values.astype('U'))
```

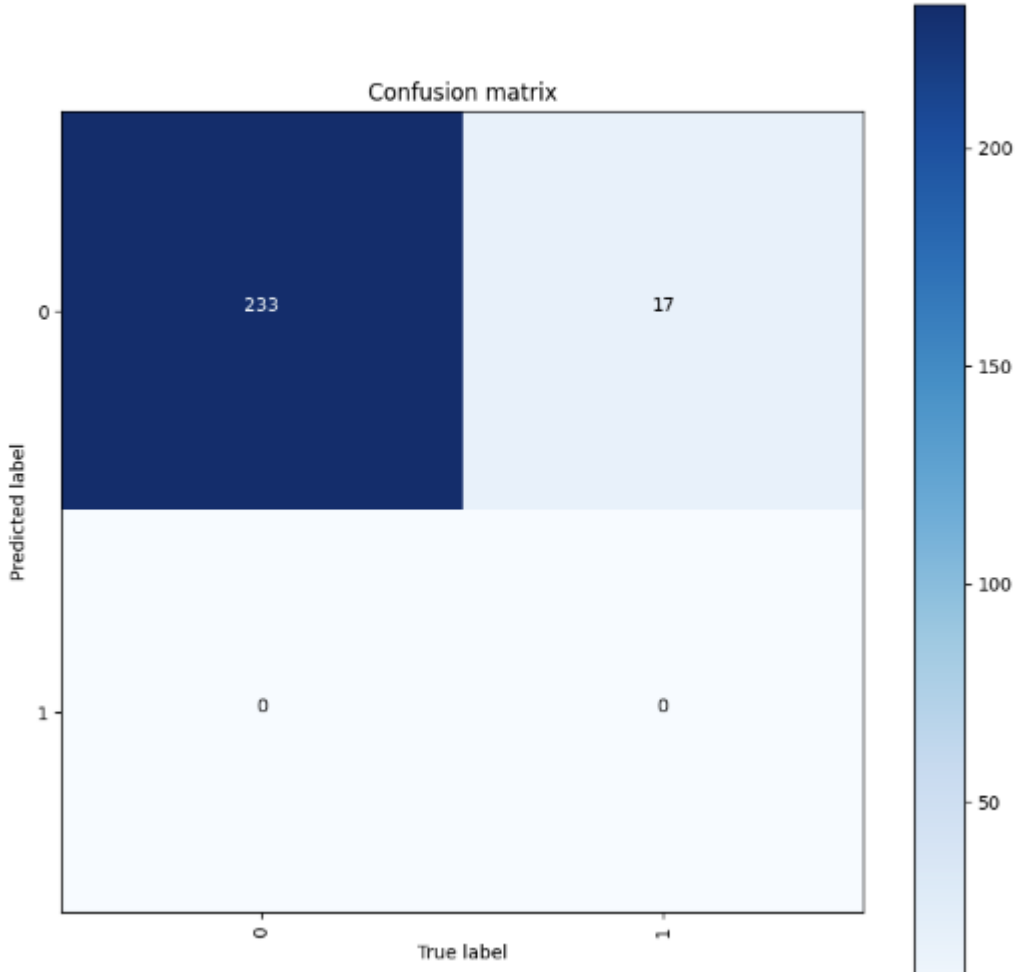
```
[ ]  1 # NEW SECTION
     2
     3 df_new_texts['predicted_sarcasm'] = 0
     4
     5 for i in range(len(predicted_value)):
     6   if predicted_value[i] == 1:
     7     df_new_texts.loc[i, ('predicted_sarcasm')] = 1
     8   # elif predicted_value.iloc[i] == 0:
     9   #   df_new['Fake'] = 0
    10
    11 print(df_new_texts['predicted_sarcasm'])
```

Confusion Matrix

```
[ ]
def plot_confusion_matrix(actual, predicted, classes,
                          normalize=False,
                          title='Confusion matrix', figsize=(7,7),
                          cmap=plt.cm.Blues, path_to_save_fig=None):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    import itertools
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(actual, predicted).T
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=figsize)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

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

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

    if path_to_save_fig:
        plt.savefig(path_to_save_fig, dpi=300, bbox_inches='tight')
```

Sarcasm - Confusion Matrix

```
[ ]  1 plot_confusion_matrix(y_valid, valid_pred,
     2                       tfidf_logit_pipeline.named_steps['logit'].classes_, figsize=(8, 8))
```



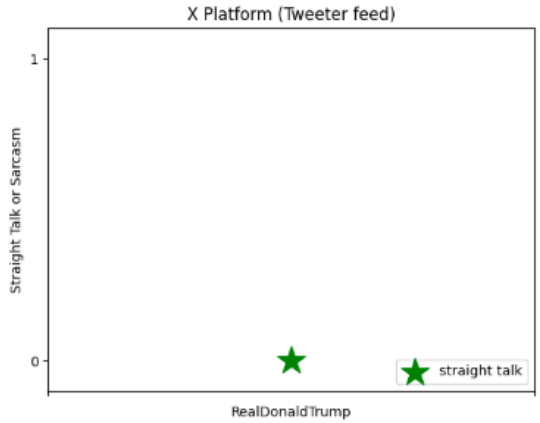
Sample graphic to represent prediction of Sarcasm or Not/Straight Talk

```
[ ]
# TO BE POINTED AT THE NEW DATA #
# TO BE POINTED AT THE NEW DATA #
# TO BE POINTED AT THE NEW DATA # - originally form Truth Fake module

import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter

# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = df_new_texts['predicted_sarcasm'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]


if y == 1:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "rasberry talk", color= "red",
              marker= "X", s=450)
  plt.legend(loc='upper right')

if y == 0:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "straight talk", color= "green",
              marker= "*", s=450)
  plt.legend(loc='lower right')


# x-axis label
plt.xlabel('RealDonaldTrump')
# frequency label
plt.ylabel('Straight Talk or Sarcasm')
# plot title
```

a

```
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
plt.xticks([-1, 1])
plt.yticks([0, 1])

# function to show the plot
plt.show()
```

```
<ipython-input-31-9e2db9c6fb92>:36: UserWarning: Attempting to set identical low and high xlims makes transformation singular; automatically expanding.
  plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
```



Improving the prediction performance: Build and use a LSTM model

Model Improvement above tfidf+logit regression 70% accurate -> trying here LSTM to get higher accuracy

```
[ ]
# LSTM

# Text pre-processing
# import tensorflow as tf
# from tensorflow.keras.preprocessing.text import Tokenizer
# from tensorflow.keras.preprocessing.sequence import pad_sequences
# from tensorflow.keras.callbacks import EarlyStopping
# # Modeling
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional


# #!pip install keras
# import keras.preprocessing.text Tokenizer
# from keras.preprocessing.sequence import pad_sequences
# from keras.layers import Embedding, LSTM, Dense
# from keras.models import Sequential

import keras
#import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional
from keras.models import Sequential


# train_texts, valid_texts, y_train, y_valid = \
#         train_test_split(train_df['comment'], train_df['label'], random_state=42)
```

```
26
27 # train_texts, valid_texts, y_train, y_valid = \
28 #         train_test_split(train_df['comment'], train_df['label'], random_state=42)
29
30 # The input text, example could be list of sentences
31 texts = train_texts[:] # [...]
32
33 # The labels corresponding to the input text
34 labels = y_train[:] # train_df['label'] # train_texts['Author'] # [...]
35
36 # Hyperparameters
37 max_words = 10000 # max number of words to use in the vocabulary
38 max_len = 200 # max length of each text (in terms of number of words)
39 embedding_dim = 200 # dimension of word embeddings
40 lstm_units = 64 # number of units in the LSTM layer
41 num_classes = len(set(labels)) # 2 # len(y_train) # y_train.unique() # 2 # len(set(labels)) # number of classes
42
43 # Tokenize the texts and create a vocabulary
44 tokenizer = Tokenizer(num_words=max_words)
45 tokenizer.fit_on_texts(texts.values.astype('U'))
46 sequences = tokenizer.texts_to_sequences(texts.values.astype('U'))
47
48 # Pad the sequences so they all have the same length
49 x = pad_sequences(sequences, maxlen=max_len)
50
51 # Create one-hot encoded labels
52 y = keras.utils.to_categorical(labels,
53                                num_classes)
54
55 # Build the model
56 model = Sequential()
57 model.add(Embedding(max_words, embedding_dim, input_length=max_len))
58 model.add(LSTM(lstm_units))
59 model.add(Dense(num_classes, activation='softmax')) # 'relu'' ; "rmsprop"
60
61 # optimizer='adam', loss='mse'
62
63
64 # Compile the model
65 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # 'binary_crossentropy'
```

The LSTM model is trained over 10 epochs (shorten quantity of data to be reproducible in a code demonstration i.e., full dataset takes 10 mintues just to do this section)

```
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # 'binary_crossentropy'

# train_texts, valid_texts, y_train, y_valid

# Train the model
model.fit(x, y, batch_size=32, epochs=10)
```

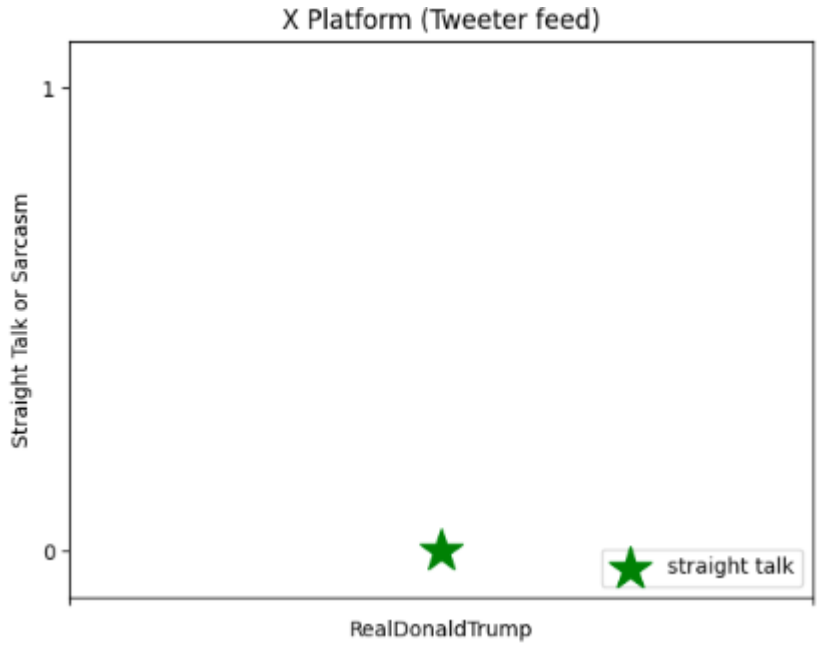
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Epoch 1/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 9s 225ms/step - accuracy: 0.8997 - loss: 0.4441
Epoch 2/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 12s 307ms/step - accuracy: 0.9435 - loss: 0.2157
Epoch 3/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 10s 303ms/step - accuracy: 0.9414 - loss: 0.2068
Epoch 4/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 7s 292ms/step - accuracy: 0.9411 - loss: 0.1798
Epoch 5/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 225ms/step - accuracy: 0.9645 - loss: 0.0946
Epoch 6/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 10s 440ms/step - accuracy: 0.9825 - loss: 0.0444
Epoch 7/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 13s 533ms/step - accuracy: 0.9960 - loss: 0.0230
Epoch 8/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 14s 262ms/step - accuracy: 0.9976 - loss: 0.0091
Epoch 9/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 9s 210ms/step - accuracy: 0.9978 - loss: 0.0069
Epoch 10/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 7s 309ms/step - accuracy: 1.0000 - loss: 0.0036
<keras.src.callbacks.history.History at 0x7bffe6bc5e10>
```

Sample graphic to represent prediction of Sarcasm or Not/Straight Talk

```
[ ]
# TO BE POINTED AT THE NEW DATA #
# TO BE POINTED AT THE NEW DATA #
# TO BE POINTED AT THE NEW DATA # - originally form Truth Fake module

import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter


plt.subplot(1, 2, 1)  # row 1, column 2, count 1
# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = df_new_texts['predicted_true_fake'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]


if y == 0:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "rasberry", color= "red",
              marker= "X", s=450)

if y == 1:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "stars", color= "green",
              marker= "*", s=450)


# x-axis label
plt.xlabel('RealDonaldTrump')
# frequency label
plt.ylabel('Fake or Truth')
# plot title
plt.title('X Platform (Twitter feed)')
# showing legend
plt.legend()
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
```

```
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
plt.xticks([-1, 1])
plt.yticks([0, 1])


plt.subplot(1, 2, 2)  # row 1, column 2, count 1
# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values
#y = df_new_texts['target'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
#y = int(prediction[0]) # [2,4,5,7,6,8,9,11,12,12]
y = prediction[0] # [2,4,5,7,6,8,9,11,12,12]

if y >= 0.99:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "rasberry talk", color= "red",
              marker= "X", s=450)
  plt.legend(loc='upper right')

if y == 0:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "straight talk", color= "green",
              marker= "*", s=450)
  plt.legend(loc='lower right')

# x-axis label
plt.xlabel('RealDonaldTrump')
# frequency label
plt.ylabel('Straight Talk or Sarcasm')
# plot title
plt.title('X Platform (Twitter feed)')
# showing legend
#plt.legend()
#plt.legend(['Legend'], loc='upper left')
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
```

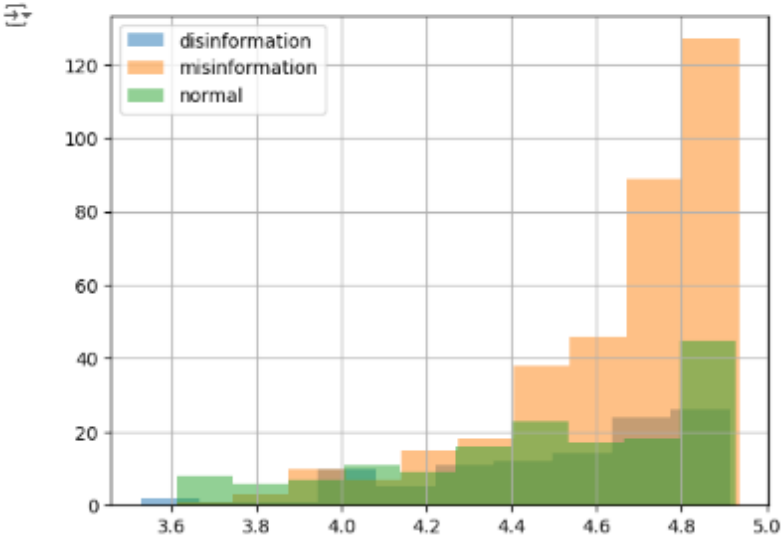
```
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
plt.xticks([-1, 1])
plt.yticks([0, 1])

# function to show the plot
#plt.show()

# function to show the plot
plt.show()
```


X Platform (Tweeter feed)
1
0
Straight Talk or Sarcasm
straight talk
RealDonaldTrump

6.3 Stance

EDA / Exploratory data analysis

```
[ ]  1 # Needs adjusting for three categories of labels 0,1,2 / true-neutral, misinformation(mistakes), disinformation(lies)
     2
     3 train_df.loc[train_df['label'] == 2, 'tweet'].str.len().apply(np.log1p).hist(label='disinformation', alpha=.5)
     4 train_df.loc[train_df['label'] == 1, 'tweet'].str.len().apply(np.log1p).hist(label='misinformation', alpha=.5)
     5 train_df.loc[train_df['label'] == 0, 'tweet'].str.len().apply(np.log1p).hist(label='normal', alpha=.5)
     6 plt.legend();
```



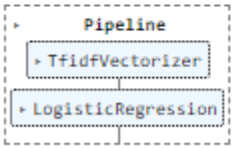
Build TF-iDF model (base model)

```
# build bigrams, put a limit on maximal number of features
# and minimal word frequency
tf_idf = TfidfVectorizer(ngram_range=(1, 2), max_features=50000, min_df=2)
# multinomial logistic regression a.k.a softmax classifier
logit = LogisticRegression(C=1, n_jobs=4, solver='lbfgs',
                           random_state=42, verbose=1)
# sklearn's pipeline
tfidf_logit_pipeline = Pipeline([('tf_idf', tf_idf),
                                 ('logit', logit)])
```

Build pipeline

```
%%time
tfidf_logit_pipeline.fit(train_texts, y_train)
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
CPU times: user 122 ms, sys: 79 ms, total: 201 ms
Wall time: 4.59 s
```



Base model: TF-iDF scores (accuracy + f1 + precision + recall)

```
accuracy_value = accuracy_score(y_valid, valid_pred)
print("Accuracy score = ", accuracy_value)
f1_score_value = f1_score(y_valid, valid_pred, average='macro')
print("F1_score = ", f1_score_value)
precision_score_value = precision_score(y_valid, valid_pred, average='macro')
print("Precison score = ", precision_score_value)
recall_score_value = recall_score(y_valid, valid_pred, average='macro')
print("Recall score = ", recall_score_value)

```

```
Accuracy score =  0.6064516129032258
F1_score =  0.37997052627772204
Precison score =  0.5466183574879228
```

New Section - Prediction on Pulled from X / Twitter tweets

Build out the memory / file from previous two notebooks that added the Truth-Fake, then Sarcasm features/mettrics

```
[ ]
# NEW SECTION

df_new_texts['predicted_misinformation'] = 0
df_new_texts['predicted_disinformation'] = 0

for i in range(len(predicted_value)):
  if predicted_value[i] == 1:
    df_new_texts.loc[i, 'predicted_misinformation'] = 1
  elif predicted_value[i] == 2:
    df_new_texts.loc[i, 'predicted_disinformation'] = 1

df_new_texts = df_new_texts[['id',  'content',  'clean_text', 'clean_joined_text',  'predicted_true_fake',  'predicted_sarcasm',  'predicted_misinformation', 'predicted_disinformation']]

#print(df_new_texts['predicted_disinformation'])
```

Model

```
def plot_confusion_matrix(actual, predicted, classes,
                          normalize=False,
                          title='Confusion matrix', figsize=(7,7),
                          cmap=plt.cm.Blues, path_to_save_fig=None):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    import itertools
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(actual, predicted).T
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=figsize)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

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

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

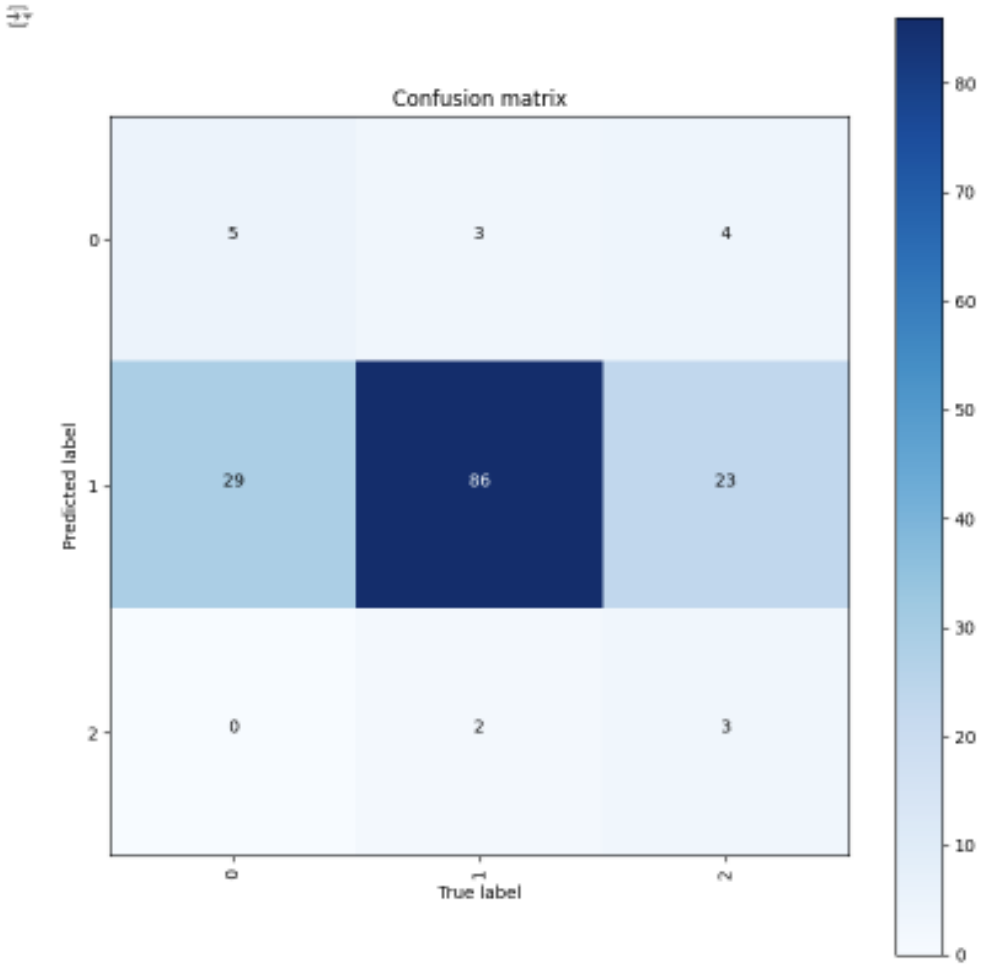
```
32    plt.xlabel('True label')
33
34    if path_to_save_fig:
35        plt.savefig(path_to_save_fig, dpi=300, bbox_inches='tight')
```

Read in memory file of previous notebooks features, Truth-Fake, Sarcasm then write the new feature Stance to the memory file

```python
import csv

# Step 1: Read the existing CSV file and store its content
csv_file_path = 'data_to_graph.csv'
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
    reader = csv.reader(file)
    data = list(reader)

# Step 2: Define the values for the new column
new_values = valid_pred # df_new_texts['predicted_sarcasm'] # ['New York', 'Los Angeles', 'Chicago', 'San Francisco']

# Step 3: Add the new column header to the first row of the data
data[0].append('predicted_stance')

# Step 4: Add the new column values to the remaining rows of the data
for i in range(1, 154): # len(data)):
    data[i].append(new_values[i - 1])

# Step 5: Write the updated data back to the CSV file
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

Confusion matrix

```
[ ]  1 plot_confusion_matrix(y_valid, valid_pred,
     2                       tfidf_logit_pipeline.named_steps['logit'].classes_, figsize=(8, 8))
```



Side-by-Side graphic: three metrics (True v Fake news + Sarcasm + Stance) visualised in graph version 1 i.e., the side-by-side graph

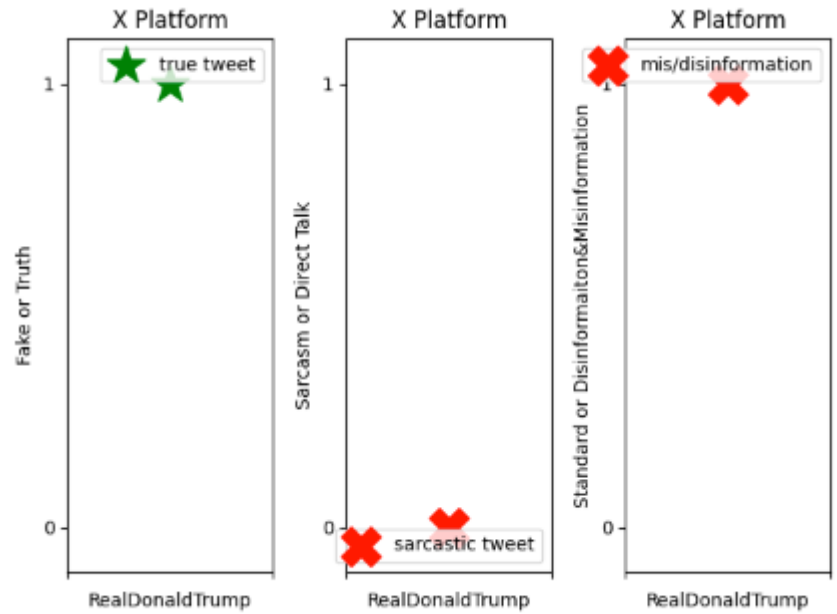
Sample graphic to represent 3 metrics i) True vs Fake ; ii) Straight vs Sarcasm ; iii) Misinformation vs Disinformation

```
# TO BE POINTED AT THE NEW DATA #
# TO BE POINTED AT THE NEW DATA #
# TO BE POINTED AT THE NEW DATA # - originally form Truth Fake module

import time
import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter

# measure User ; Sys ; and Wall time for evaluation
#%%time
start_time = time.process_time()
# Code to measure

plt.subplot(1, 3, 1)  # row 1, column 2, count 1
# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values

df_new_texts = pd.read_csv('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv')
y = df_new_texts['predicted_true_fake'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]

if y == 0:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "fake tweet", color= "red",
              marker= "X", s=450)
  plt.legend(loc='lower left')
if y >= 0.99:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "true tweet", color= "green",
              marker= "*", s=450)
  plt.legend(loc='upper right')

# x-axis label
plt.xlabel('RealDonaldTrump')
# frequency label
plt.ylabel('Fake or Truth')
```

```
# frequency label
plt.ylabel('Fake or Truth')
# plot title
plt.title('X Platform')
# showing legend
#plt.legend()
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
plt.xticks([-1, 1])
plt.yticks([0, 1])


plt.subplot(1, 3, 2)  # row 1, column 2, count 1
# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values
#y = df_new_texts['target'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
#y = int(prediction[0]) # [2,4,5,7,6,8,9,11,12,12]
#y = prediction[0] # [2,4,5,7,6,8,9,11,12,12]
y = df_new_texts['predicted_sarcasm'].iloc[0]


if y >= 0.99:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "direct tweet", color= "green",
              marker= "*", s=450)
  plt.legend(loc='upper right')
if y == 0:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "sarcastic tweet", color= "red",
              marker= "X", s=450)
  plt.legend(loc='lower right')

# x-axis label
plt.xlabel('RealDonaldTrump')
```

```
# x-axis label
plt.xlabel('RealDonaldTrump')
# frequency label
plt.ylabel('Sarcasm or Direct Talk')
# plot title
plt.title('X Platform')
# showing legend
#plt.legend()
#plt.legend(['Legend'], loc='upper left')
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
plt.xticks([-1, 1])
plt.yticks([0, 1])

# function to show the plot
#plt.show()


plt.subplot(1, 3, 3)  # row 1, column 2, count 1
# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values

#y = predicted_value.iloc[0] # valid_pred # df_new_texts['predicted_true_fake'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
y = df_new_texts['predicted_stance'].iloc[0]

if y == 0:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "neutral tweet", color= "green",
              marker= "star", s=450)
  plt.legend(loc='lower left')
if y >= 0.99:
  # plotting points as a scatter plot
```

```
if y >= 0.99:
  # plotting points as a scatter plot
  plt.scatter(x, y, label= "mis/disinformation", color= "red",
              marker= "X", s=450)
  plt.legend(loc='upper right')

# x-axis label
plt.xlabel('RealDonaldTrump')
# frequency label
plt.ylabel('Standard or Disinformaiton&Misinformation')
# plot title
plt.title('X Platform')
# showing legend
#plt.legend()
# Setting the axis range
plt.xlim(0,0) # (-0.5, 0.5)  # X-axis range from 0 to 5
plt.ylim(-0.1, 1.1)
plt.gca().set_xticklabels([])

#plt.set_yticklabels([str(round(float(label), 0)) for target in y])
plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
plt.xticks([-1, 1])
plt.yticks([0, 1])


# function to show the plot
plt.tight_layout()
plt.show()

end_time = time.process_time()
print(f"CPU time used: {end_time - start_time} seconds")
```




```
CPU time used: 0.7297617659999993 seconds
```

New model (LSTM) to see can an improvement be made to the base model of Tf-iDF + Logistic Regression

->>NEW LSTM in Stance notebook<<- Model Improvement** above tfidf+logit regression 70% accurate -> trying here LSTM to get higher accuracy **bold text**

+ Code + Text

```
[ ]
# LSTM

# Text pre-processing
# import tensorflow as tf
# from tensorflow.keras.preprocessing.text import Tokenizer
# from tensorflow.keras.preprocessing.sequence import pad_sequences
# from tensorflow.keras.callbacks import EarlyStopping
# # Modeling
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional


# #!pip install keras
# import keras.preprocessing.text Tokenizer
# from keras.preprocessing.sequence import pad_sequences
# from keras.layers import Embedding, LSTM, Dense
# from keras.models import Sequential

import keras
#import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional
from keras.models import Sequential
from keras.metrics import F1Score # (average=None, threshold=None, name="f1_score", dtype=None)
```

```
from keras.metrics import F1Score # (average=None, threshold=None, name="f1_score", dtype=None)


# The input text, example could be list of sentences
texts = train_texts[:] # [...]

# The labels corresponding to the input text
labels = y_train[:] # train_df['label'] # train_texts['Author'] # [...]

# Hyperparameters
max_words = 10000 # max number of words to use in the vocabulary
max_len = 200 # max length of each text (in terms of number of words)
embedding_dim = 200 # dimension of word embeddings
lstm_units = 64 # number of units in the LSTM layer
num_classes = len(set(labels)) # 2 # len(y_train) # y_train.unique() # 2 # len(set(labels)) # number of classes

# Tokenize the texts and create a vocabulary
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts.values.astype('U'))
sequences = tokenizer.texts_to_sequences(texts.values.astype('U'))

# Pad the sequences so they all have the same length
x = pad_sequences(sequences, maxlen=max_len)

# Create one-hot encoded labels
y = keras.utils.to_categorical(labels,
                               num_classes)

# Build the model
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_len))
model.add(LSTM(lstm_units))
model.add(Dense(num_classes, activation='softmax')) # 'relu'' ; "rmsprop"

# optimizer='adam', loss='mse'
```

```
model.add(Dense(num_classes, activation='softmax')) # 'relu'' ; "rmsprop"

# optimizer='adam', loss='mse'


# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # 'binary_crossentropy'

# train_texts, valid_texts, y_train, y_valid

# Train the model
model.fit(x, y, batch_size=32, epochs=10)
```

Epoch training (data used is shortened to make the code run within the time limit of the demonstation)

```
Epoch 1/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 421ms/step - accuracy: 0.5086 - loss: 1.0644
Epoch 2/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 395ms/step - accuracy: 0.5684 - loss: 0.9416
Epoch 3/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 262ms/step - accuracy: 0.5929 - loss: 0.8251
Epoch 4/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 259ms/step - accuracy: 0.7658 - loss: 0.5960
Epoch 5/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 5s 367ms/step - accuracy: 0.8924 - loss: 0.3260
Epoch 6/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 5s 294ms/step - accuracy: 0.9796 - loss: 0.1402
Epoch 7/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 197ms/step - accuracy: 0.9893 - loss: 0.0688
Epoch 8/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 5s 202ms/step - accuracy: 0.9940 - loss: 0.0424
Epoch 9/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 273ms/step - accuracy: 0.9994 - loss: 0.0186
Epoch 10/10
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 197ms/step - accuracy: 0.9959 - loss: 0.0161
<keras.src.callbacks.history.History at 0x79d66adf25c0>
```

Point the new model (LSTM) at the new data

Using NEW data -> predict using NEW model LSTM previously trained on reference ground truth labelled stance data

```
[ ]
# Using NEW data -> predict using NEW model LSTM prviously trained on reference ground truth labelled stance data
# write Stance prediction results to shared dataframe

import numpy as np

#X = df_new_texts['Text']

# Tokenize the texts and create a vocabulary
tokenizer = Tokenizer(num_words=max_words)
#tokenizer.fit_on_texts(texts.values.astype('U'))
tokenizer.fit_on_texts(df_new_texts['content'].values.astype('U'))
# sequences = tokenizer.texts_to_sequences(texts.values.astype('U'))
sequences = tokenizer.texts_to_sequences(df_new_texts['content'].values.astype('U'))

# Pad the sequences so they all have the same length
X = pad_sequences(sequences, maxlen=max_len)

prediction = []
prediction = model.predict(X)

# Creating multi-dimension array
#array1 = [1, 2, 4, [5, [6, 7]]]

# Object Data type is accept all data-type
Data_type = int

# Now we fix the error
#np_array = numpy.array(prediction, dtype=Data_type)
# TEMP commented out
#prediction = np.array(prediction, dtype=Data_type)
#prediction = prediction.flatten('C')


df_new_texts['predicted_stance_neutral'] = 0
df_new_texts['predicted_stance_misinformation'] = 0
df_new_texts['predicted_stance_disinformation'] = 0
df_new_texts['predicted_stance_all_types'] = 0

```

```
df_new_texts['predicted_stance_disinformation'] = 0
df_new_texts['predicted_stance_all_types'] = 0


#for i in range(len(prediction)):
i = 0
for p in prediction[:, 0]:
  if p >= 0.50:
    #print(p)
    df_new_texts.loc[i, 'predicted_stance_neutral'] = 1
  i += 1

i = 0
for p in prediction[:, 1]:
  if p >= 0.50:
    df_new_texts.loc[i, 'predicted_stance_misinformation'] = 1
  i += 1

i = 0
for p in prediction[:, 2]:
  if p >= 0.50:
    df_new_texts.loc[i, 'predicted_stance_disinformation'] = 1
  i += 1

i = 0
#for p in prediction[:, 2]:
for i in range(len(df_new_texts)):
  if df_new_texts.loc[i, 'predicted_stance_neutral'] == 1 and df_new_texts.loc[i, 'predicted_stance_misinformation'] == 0 and df_new_texts.loc[i, 'predicted_stance_disinformation'] == 0:
    df_new_texts.loc[i, 'predicted_stance_all_types'] = 0
  if df_new_texts.loc[i, 'predicted_stance_neutral'] == 0 and df_new_texts.loc[i, 'predicted_stance_misinformation'] == 1 and df_new_texts.loc[i, 'predicted_stance_disinformation'] == 0:
    df_new_texts.loc[i, 'predicted_stance_all_types'] = 1
  if df_new_texts.loc[i, 'predicted_stance_neutral'] == 0 and df_new_texts.loc[i, 'predicted_stance_misinformation'] == 0 and df_new_texts.loc[i, 'predicted_stance_disinformation'] == 1:
    df_new_texts.loc[i, 'predicted_stance_all_types'] = 2
  i += 1
i += 1


df_new_texts = df_new_texts[['id', 'content', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_stance_neutral', 'predicted_stance_misinformation', 'predicted_stance_disinformation', 'predicted_stance_all_types']].copy()
```

```
#df_new_texts.drop(index='predicted_stance', axis=0, inplace=True)
print(df_new_texts)
df_new_texts.info()
df_new_texts.describe()
print(len(prediction))
#print(len( df_new_texts['predicted_stance']))
len(X)
```

Training new model and data output

```
304/304 ━━━━━━━━━━━━━━━━━━━━ 22s 73ms/step
                     id                                            content  \
0      965925223949357056  Thank you to @ foxandfriends for the great tim...
1      965928352614965248  "There is no serious person out there who woul...
2      965930611272712192  ....The President Obama quote just before elec...
3      965932714141650946  Republicans are now leading the Generic Poll, ...
4      965935035328155649  Matt Schlapp and CPAC are getting ready for an...
...                   ...                                                ...
9721  1273405198698975232  Joe Biden was a TOTAL FAILURE in Government. H...
9722  1273408026968457216  Will be interviewed on @ seanhannity tonight a...
9723  1273442195161387008                          pic.twitter.com/3lm1spbU8X
9724  1273442469066276864                          pic.twitter.com/vpCE5MadUz
9725  1273442528411385858                          pic.twitter.com/VL1c0BHW41
```

Write third (Stance) metric to the 'memory' / built up csv file

IMPORTANT - Only run ONCE

- write feaure column for neutral stance
- write feaure column for misinformation
- write feaure column for disinformation
- write feaure column for all stance types

```
# IMPORTANT - Only run ONCE
# save shared data frame with Saracasm prediction resutls added to the shared file

import csv

# # Step 1: Read the existing CSV file and store its content
# csv_file_path = 'data_to_graph.csv'
# with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
#     reader = csv.reader(file)
#     data = list(reader)

# # Step 2: Define the values for the new "City" column
# new_values = df_new_texts['predicted_sarcasm'] # ['New York', 'Los Angeles', 'Chicago', 'San Francisco']

# # Step 3: Add the new "City" column header to the first row of the data
# data[0].append('predicted_sarcasm')

# # Step 4: Add the new "City" column values to the remaining rows of the data
# for i in range(1, len(data)):
#     data[i].append(new_values[i - 1])

```

```
#     data[i].append(new_values[i - 1])

# # Step 5: Write the updated data back to the CSV file
# with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
#     writer = csv.writer(file)
#     writer.writerows(data)


# Step 1: Read the existing CSV file and store its content
csv_file_path = 'data_to_graph.csv'
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
    reader = csv.reader(file)
    data = list(reader)
#Step 2 , 3, 4 repeated
new_values = df_new_texts['predicted_stance_neutral']
data[0].append('predicted_stance_neutral')
#data = data[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm']].copy()
for i in range(1, len(data)):
    data[i].append(new_values[i - 1])
# Step 5: Write the updated data back to the CSV file
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

# Step 1: Read the existing CSV file and store its content
csv_file_path = 'data_to_graph.csv'
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
    #file = file[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_stance_neutral']].copy()
    reader = csv.reader(file)
    data = list(reader)
#Step 2, 3, 4 repeated
new_values = df_new_texts['predicted_stance_misinformation']
data[0].append('predicted_stance_misinformation')
for i in range(1, len(data)):
    data[i].append(new_values[i - 1])
# Step 5: Write the updated data back to the CSV file
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)
```

```
    writer.writerows(data)

# Step 1: Read the existing CSV file and store its content
csv_file_path = 'data_to_graph.csv'
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
    #file = file[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_stance_neutral', 'predicted_stance_misinformation']].copy()
    reader = csv.reader(file)
    data = list(reader)
#Step 2, 3, 4 repeated
new_values = df_new_texts['predicted_stance_disinformation']
data[0].append('predicted_stance_disinformation')
for i in range(1, len(data)):
    data[i].append(new_values[i - 1])
# Step 5: Write the updated data back to the CSV file
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)


# Step 1: Read the existing CSV file and store its content
csv_file_path = 'data_to_graph.csv'
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
    #file = file[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_stance_neutral', 'predicted_stance_misinformation', 'predicted_stance_disinformation']].copy()
    reader = csv.reader(file)
    data = list(reader)
#Step 2, 3, 4 repeated
new_values = df_new_texts['predicted_stance_all_types']
data[0].append('predicted_stance_all_types')
for i in range(1, len(data)):
    data[i].append(new_values[i - 1])
# Step 5: Write the updated data back to the CSV file
with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

# # Step 5: Write the updated data back to the CSV file
# with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
#     writer = csv.writer(file)
#     writer.writerows(data)
#       writer = csv.writer(file)
#       writer.writerows(data)


print('predicted_sarcasm = ', df_new_texts['predicted_sarcasm'].sum())
print('predicted_stance_neutral = ', df_new_texts['predicted_stance_neutral'].sum())
print('predicted_stance_misinformation = ', df_new_texts['predicted_stance_misinformation'].sum())
print('predicted_stance_disinformation = ', df_new_texts['predicted_stance_disinformation'].sum())
print('predicted_stance_all_types = ', df_new_texts['predicted_stance_all_types'].sum())
#print(df_new_texts['predicted_stance_all_types'].value_counts)
```

Output new version 2 of the visualisation i.e., the combined chart (of the three metrics True v Fake news, Sarcasm, Stance)

NEW combined chart

Plot version 2 chart - the combined features chart

```
[ ]
# NEW combined chart

# importing package
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from matplotlib.ticker import StrMethodFormatter

import io
import xml.etree.ElementTree as ET
import time

from matplotlib.lines import Line2D
from matplotlib.markers import MarkerStyle
from matplotlib.transforms import Affine2D

from google.colab import drive
drive.mount('/content/drive/')

# x-axis values
x = [0] # [1,2,3,4,5,6,7,8,9,10]
# y-axis values

# measure User ; Sys ; and Wall time for evaluation

start_time = time.process_time()
# Code to measure

df_new_texts = pd.read_csv('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv')


for i in range(10):
  y1 = df_new_texts['predicted_true_fake'].iloc[i] # [2,4,5,7,6,8,9,11,12,12]
  y2 = df_new_texts['predicted_sarcasm'].iloc[i]
  #y3 = df_new_texts['predicted_stance_neutral'].iloc[i]
  y3 = df_new_texts['predicted_stance_all_types'].iloc[i]

  # 'predicted_stance_misinformation'
  # 'predicted_stance_disinformation'
```

```
#y4 = df_new_texts['predicted_stance_misinformation'].iloc[0]
#y5 = df_new_texts['predicted_stance_disinformation'].iloc[0]

# create data
# x = [1,2,3,4,5]
# y = [3,3,3,3,3]

# text_style = dict(horizontalalignment='right', verticalalignment='center',
#                   fontsize=12, fontfamily='monospace')
# marker_style = dict(linestyle=':', color='0.8', markersize=10,
#                     markerfacecolor="tab:blue", markeredgecolor="tab:blue")

# y1 = [0]
# y2 = [0]
# y3 = [0]
# all good / all zeros / true news 0 / straight talk 0/ neutral stance 0
if y1 == [0] and y2 == [0] and y3 == [0]:
  #print('Test = if block runs')
  # plot lines
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(-3, -7, 'True News')
  x2 = [3,0]
  y2 = [5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(-3, -7, 'Direct Talk')
  x3 = [3,0]
  y3 = [7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance_Neutral", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
```

```
# y1 = [1]
# y2 = [1]
# y3 = [1]
# all up / all ones / all fake 1 / sarcasm 1/ all misinformation 1
if y1 == [1] and y2 == [1] and y3 == [1]:
  #print('Test = if block runs')
  # plot lines
  x1 = [-3,0]
  y1 = [-3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -3, 'Fake News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Sarcasm')
  x3 = [-3,0]
  y3 = [-7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Misinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -7, 'Misinformation')

# y1 = [1]
# y2 = [0]
# y3 = [1]
# all good / all zeros / true news 0 / straight talk 0/ neutral stance 0
if y1 == [1] and y2 == [0] and y3 == [1]:
  #print('Test = if block runs')
```

```
if y1 == [1] and y2 == [0] and y3 == [1]:
  #print('Test = if block runs')
  # plot lines
  x1 = [-3,0]
  y1 = [-3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -3, 'Fake News')
  x2 = [3,0]
  y2 = [5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 5, 'Direct Talk')
  x3 = [-3,0]
  y3 = [-7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance Misinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -7, 'Misinformation')

# y1 = [0]
# y2 = [0]
# y3 = [1]
# Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
if y1 == [0] and y2 == [0] and y3 == [1]:
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 3, 'True News')
  x2 = [3,0]
  y2 = [5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 5, 'Direct Talk')
```

```
  y3 = [-7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Misinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -7, 'Misinformation')

  plt.legend(loc='lower right')
  from matplotlib import pyplot as plt
  plt.savefig('/content/drive/My Drive/Colab_Notebooks/MSCAITOP/MSCAITOP_graphic_v2a.png', bbox_inches='tight')


# y1 = [0]
# y2 = [1]
# y3 = [1]
# passed testing
# Mixed / all fake 0/sarcasm 1/all misinformation&disinformation 1
if y1 == [0] and y2 == [1] and y3 == [1]:
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 3, 'True News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Sarcasm Talk')
  x3 = [-3,0]
  y3 = [-7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Misinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -7, 'Misinformation')
```

```
  plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -7, 'Misinformation')

# y1 = [0]
# y2 = [1]
# y3 = [0]
# Mixed / all fake 0/sarcasm 1/all misinformation&disinformation 0
if y1 == [0] and y2 == [1] and y3 == [0]:
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 3, 'True News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Sarcasm Talk')
  x3 = [3,0]
  y3 = [7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance_Neutral", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 7, 'Stance_Neutral')

# y1 = [1]
# y2 = [1]
# y3 = [0]
# Mixed / all fake 1/sarcasm 1/all misinformation&disinformation 0
if y1 == [1] and y2 == [1] and y3 == [0]:
  x1 = [-3,0]
  y1 = [-3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -3, 'Fake News')
  x2 = [-3,0]
```

```
  plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -3, 'Fake News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Saracasm Talk')
  x3 = [3,0]
  y3 = [7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance_Neutral", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 7, 'Stance_Neutral')

# y1 = [1]
# y2 = [1]
# y3 = [0]
# Mixed / all fake 1/sarcasm 1/all misinformation&disinformation 0
if y1 == [1] and y2 == [1] and y3 == [0]:
  x1 = [-3,0]
  y1 = [-3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -3, 'Fake News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Sarcasm Talk')
  x3 = [3,0]
  y3 = [7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance_Neurtral", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 7, 'Stance_Neutral')
```

```
    plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
    plt.text(3, 7, 'Stance_Neutral')

  # y1 = [1]
  # y2 = [0]
  # y3 = [0]
  #passed testing
  # Mixed / all fake 1/sarcasm 0/all misinformation&disinformation 0
  if y1 == [1] and y2 == [0] and y3 == [0]:
    x1 = [-3,0]
    y1 = [-3,0]
    #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
    plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
    plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
    plt.text(-3, -3, 'Fake News')
    x2 = [3,0]
    y2 = [5,0]
    #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
    plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
    plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
    plt.text(3, 5, 'Direct Talk')
    x3 = [3,0]
    y3 = [7,0]
    #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
    plt.plot(x3, y3, label = "Stance_Neurtral", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
    plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
    plt.text(3, 7, 'Stance_Neutral')



  plt.xlim(-4, 6)
  plt.ylim(-8, 9)
  plt.gca().set_xticklabels([])
  plt.gca().set_yticklabels([])

  # y1 = [0]
  # y2 = [0]
  # y3 = [1]
  # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
  if y1 == [0] and y2 == [0] and y3 == [1]:
```

```
plt.gca().set_yticklabels([])

# y1 = [0]
# y2 = [0]
# y3 = [1]
# Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
if y1 == [0] and y2 == [0] and y3 == [1]:
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 3, 'True News')
  x2 = [3,0]
  y2 = [5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 5, 'Direct Talk')
  x3 = [-3,0]
  y3 = [-7,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance Misinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -7, 'Stance Misinformation')


# *** NEW for 0,1,2 single feature for stance column ***
# y1 = [0]
# y2 = [0]
# y3 = [2]
# Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
if y1 == [0] and y2 == [0] and y3 == [2]:
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 3, 'True News')
  x2 = [3,0]
  y2 = [5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 5, 'Direct Talk')
```

```
  plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 5, 'Direct Talk')
  x3 = [-3,0]
  y3 = [-9,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -9, 'Stance Disinformation')

# y1 = [0]
# y2 = [1]
# y3 = [2]
# Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
if y1 == [0] and y2 == [1] and y3 == [2]:
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 3, 'True News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Sarcasm Talk')
  x3 = [-3,0]
  y3 = [-9,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -9, 'Stance Disinformation')

# y1 = [1]
# y2 = [1]
# y3 = [2]
# Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
if y1 == [1] and y2 == [1] and y3 == [2]:
  x1 = [-3,0]
```

```
  x1 = [-3,0]
  y1 = [-3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -3, 'Fake News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Sarcasm Talk')
  x3 = [-3,0]
  y3 = [-9,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -9, 'Stance Disinformation')

# y1 = [1]
# y2 = [0]
# y3 = [2]
# Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
if y1 == [1] and y2 == [0] and y3 == [2]:
  x1 = [-3,0]
  y1 = [-3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -3, 'Fake News')
  x2 = [3,0]
  y2 = [5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 5, 'Direct Talk')
  x3 = [-3,0]
```

```
  plt.text(3, 5, 'Direct Talk')
  x3 = [-3,0]
  y3 = [-9,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -9, 'Stance Disinformation')

# y1 = [0]
# y2 = [1]
# y3 = [2]
# Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
if y1 == [0] and y2 == [1] and y3 == [2]:
  x1 = [3,0]
  y1 = [3,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
  plt.text(3, 3, 'True News')
  x2 = [-3,0]
  y2 = [-5,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -5, 'Sarcasm Talk')
  x3 = [-3,0]
  y3 = [-9,0]
  #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style)  # linestyle="-", color='red'
  plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
  plt.text(-3, -9, 'Stance Disinformation')

plt.legend(loc='lower right')

from matplotlib import pyplot as plt
plt.savefig('/content/drive/My Drive/Colab_Notebooks/MSCAITOP/MSCAITOP_graphic_v2b.png', bbox_inches='tight')

plt.show()

# l1, = ax.plot([0.1, 0.5, 0.9], [0.1, 0.9, 0.5], "bo-",
```

```
440  plt.savefig('/content/drive/My Drive/Colab_Notebooks/MSCAITOP/MSCAITOP_graphic_v2b.png', bbox_inches='tight')
441
442  plt.show()
443
444  # l1, = ax.plot([0.1, 0.5, 0.9], [0.1, 0.9, 0.5], "bo-",
445  #               mec="b", lw=5, ms=10, label="Line 1")
446
447
448 end_time = time.process_time()
449
450 print(f"Total for batch of ", i + 1, " charts - the Total CPU time used:", {end_time - start_time}, "seconds")
451 print(f"Per chart CPU time used: {(end_time - start_time) / (i + 1)} seconds")
452
```

References

1 - https://www.kaggle.com/code/paramarthasengupta/fake-news-detector-eda-prediction-99

2 - https://www.kaggle.com/datasets/deepnews/fakenews-reddit-comments/data

3 - https://www.kaggle.com/datasets/arashnic/7-nlp-tasks-with-tweets