

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)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

```
1 stop_words = stopwords.words('english')
2 stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
3 def preprocess(text):
4     result = []
5     for token in gensim.utils.simple_preprocess(text):
6         if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 2 and token not in stop_words:
7             result.append(token)
8
9     return result
```

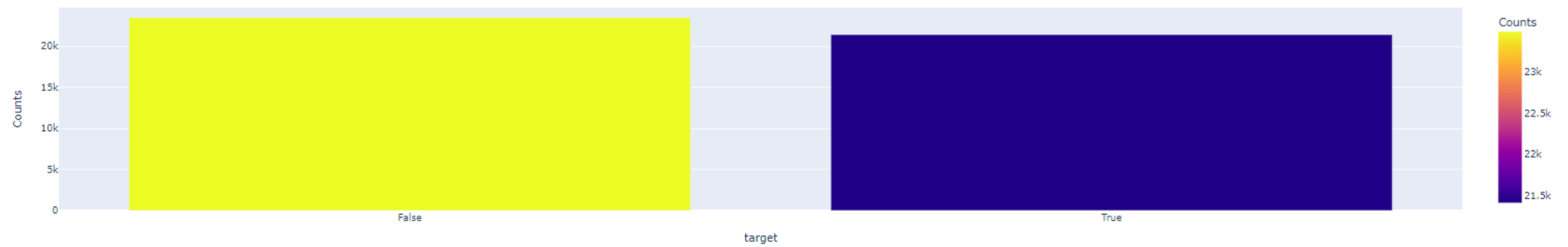
```
[ ] 1 # Transforming the unmatched subjects to the same notation
    2 df.subject=df.subject.replace({'politics':'PoliticsNews','politicsNews':'PoliticsNews'})
```

Additional data cleaning: transforming generic references to proper nouns

▼ Exploratory Data Analysis (EDA)

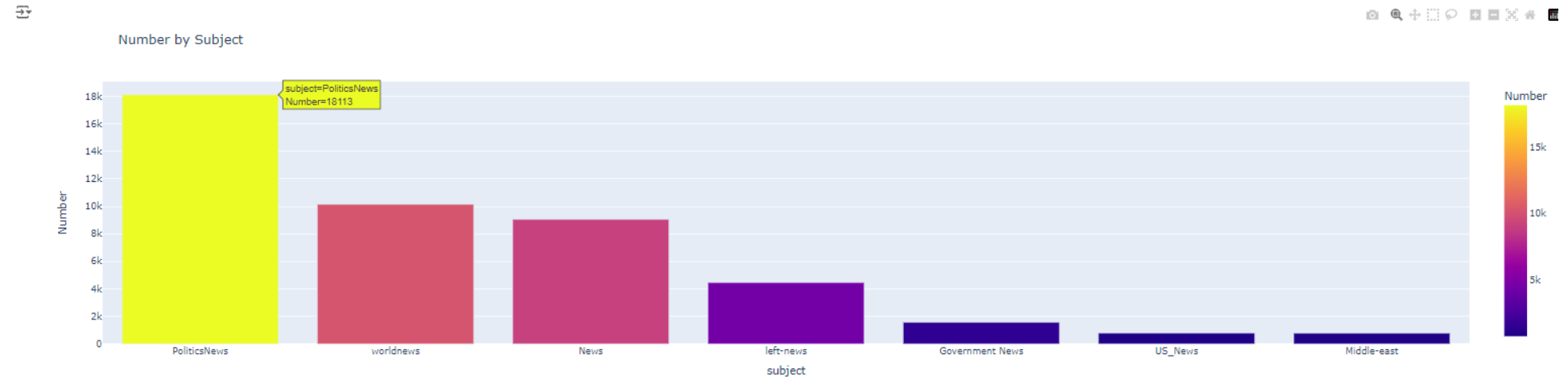
Histogram for balance of true v fake news

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

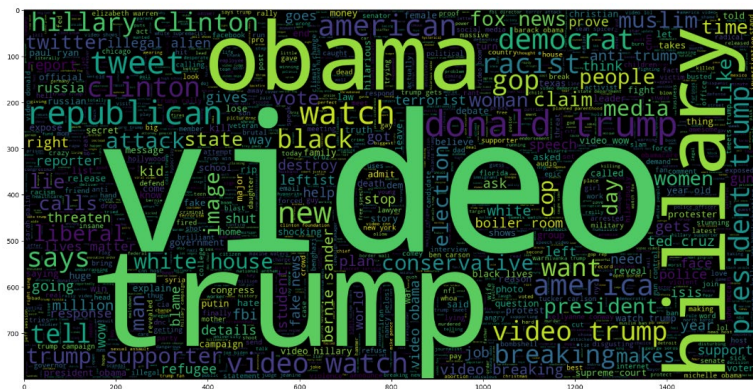


▼ Histogram of subjects covered

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



Below is Word Cloud of True News words



Below is Word Cloud of Fake News words

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("Precision score = ", precision_score_value)
```

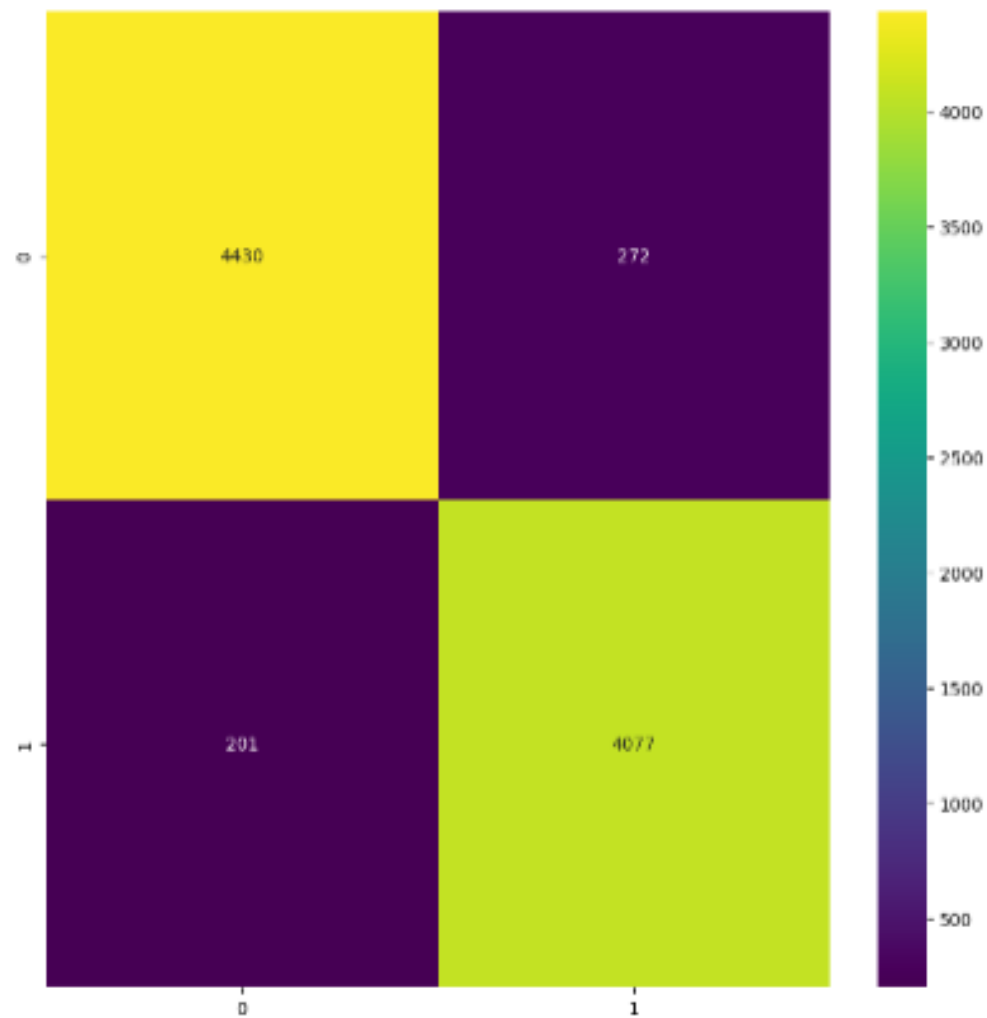
```
⇒ Accuracy score = 0.9475838516986892
   F1_score = 0.9472458762778595
   Precision score = 0.9470268669868982
   /usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning:
```

1st Confusion Matrix

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



<Axes: >



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

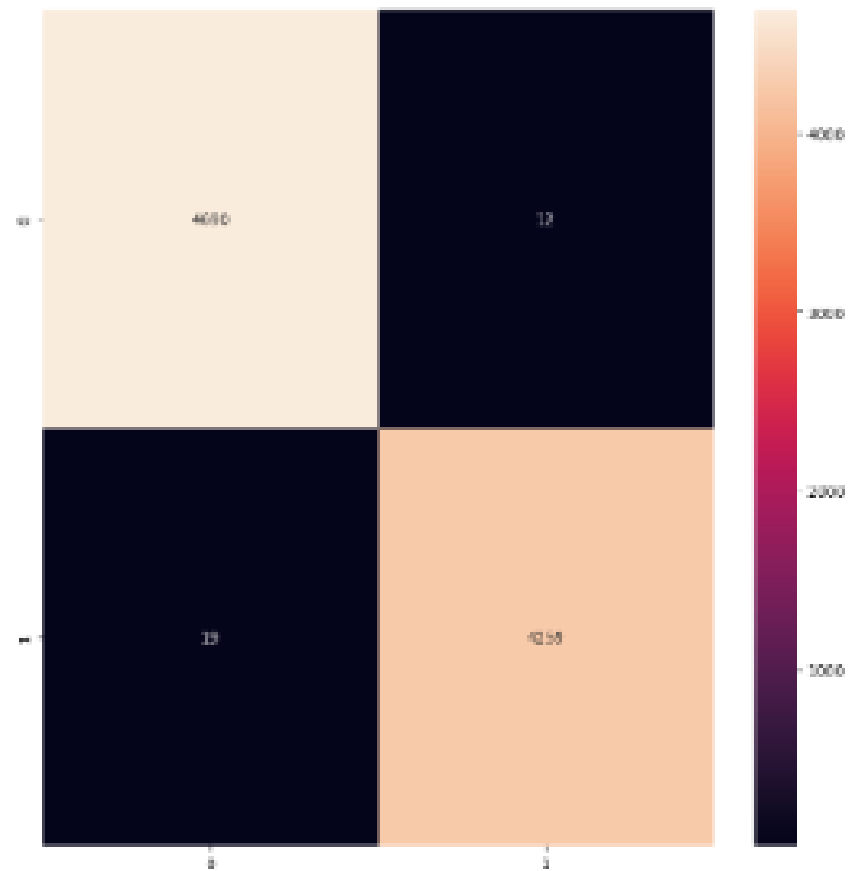
```
1 #prediction based on Text 100-200 words vs only Title 8 words max
2
3 X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_text, df.target, test_size = 0.2, random_state=2)
4 vec_train = CountVecorizer().fit(X_train)
5 X_vec_train = vec_train.transform(X_train)
6 X_vec_test = vec_train.transform(X_test)
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("Precision score = ", precision_score_value)
```

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

2nd The Confusion Matrix

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

<matplotlib.figure.Figure: 1000x1000>



✓ 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("Precision score = ", precision_score_value)
```

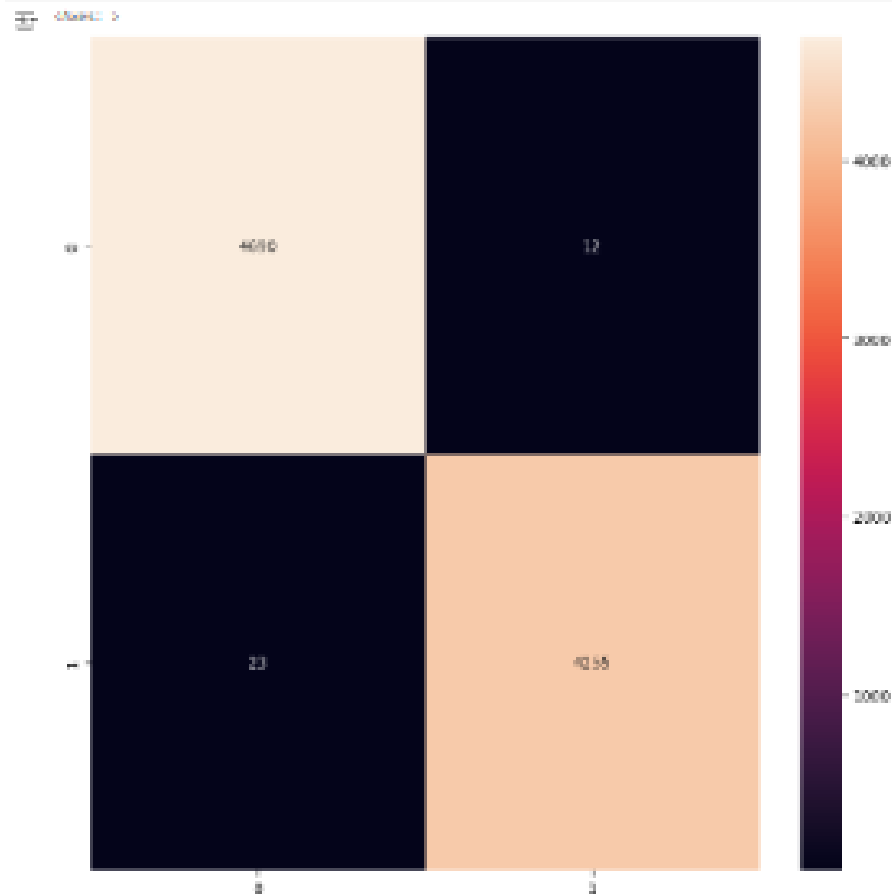
```
0.996035775213476
Accuracy score = 0.996035775213476
F1_score = 0.9960932826649207
Precision score = 0.9961538004445567
```



```

1 prediction = []
2 for i in range(len(predicted_value)):
3     if predicted_value[i].item() > 0.5:
4         prediction.append(1)
5     else:
6         prediction.append(0)
7 cm = confusion_matrix(list(y_test), prediction)
8 plt.figure(figsize = (10, 10))
9 plt.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

```
[ ] 1 # Build out Dataset - make predictions and assign to target column in dataframe
2
3 import numpy as np
4 np.set_printoptions(threshold=np.inf)
5
6 #X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_final, df.target, test_size = 0.2, random_state=0)
7 #X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_text, df.target, test_size = 0.2, random_state=0)
8
9 #vec_train = CountVectorizer().fit(X_train)
10 #X_vec_train = vec_train.transform(X_train)
11 #X_vec_test = vec_train.transform(X_test)
12
13 # -> put in new tweets here -> X_vec_test = vec_train.transform(X_test)
14
15 #model = LogisticRegression(C=3)
16 #model.fit(X_vec_train, y_train)
17
18 # Vectorising Text column (vs Title or Title and Text) to input to model previously training on the static data inorder to make predictions on the new X data
19 X_vec_train = vec_train.transform(df_new['clean_joined_text'])
20 predicted_value = model.predict(X_vec_test) # it is these predicted values that are to have the graphics to be created from
21 #print(predicted_value.reshape(-1))
22
23 df_new['predicted_true_fake'] = 0
24
```

```

23 df_new['predicted_true_fake'] = 0
24
25 # for i in range(len(df_new)):
26 #     if predicted_value[i] == 1:
27 #         #df_new['target'].iloc[i] = 1
28 #         df_new.loc[i, 'predicted_true_fake'] = 1
29 #     elif predicted_value.iloc[i] == 0:
30 #         # df_new['Fake'] = 0
31
32 i = 0
33 for p in predicted_value:
34     if p == 1:
35         #df_new['target'].iloc[i] = 1
36         df_new.loc[i, 'predicted_true_fake'] = 1
37         i += 1
38     # elif predicted_value.iloc[i] == 0:
39     #     df_new['Fake'] = 0
40
41
42 print(df_new['predicted_true_fake'])
43
44 #df_new.info()
45 #df_new = df_new[0,1,2,3,4]
46 #df2 = df1[['A', 'C']].copy()
47 df_new = df_new[['id', 'content', 'clean_text', 'clean_joined_text', 'predicted_true_fake']].copy()
48 #df.drop(df.columns[[5:]], axis=1, inplace=True)
49 df_new.info()
50
51 # ADD section for EDA again here
52
53 # NOT relevant as not in training mode but prediction # accuracy_value = roc_auc_score(y_test, predicted_value)
54 # NOT relevant as not in training mode but prediction # print(accuracy_value)

```

0 1

✓ Sample graphic to represent prediction of Fake v True Tweet

```
1 import matplotlib.pyplot as plt
2 from matplotlib.ticker import StrMethodFormatter
3
4 # x-axis values
5 x = [0] # [1,2,3,4,5,6,7,8,9,10]
6 # y-axis values
7 y = df_new['predicted_true_fake'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
8
9
10 if y == 0:
11     # plotting points as a scatter plot
12     plt.scatter(x, y, label= "rasberry", color= "red",
13               marker= "X", s=450)
14
15 if y == 1:
16     # plotting points as a scatter plot
17     plt.scatter(x, y, label= "stars", color= "green",
18               marker= "*", s=450)
19
20
21 # x-axis label
22 plt.xlabel('RealDonaldTrump')
23 # frequency label
24 plt.ylabel('Fake or Truth')
25 # plot title
26 plt.title('X Platform (Tweeter feed)')
27 # showing legend
28 plt.legend()
29 # Setting the axis range
30 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
31 plt.ylim(-0.1, 1.1)
32 plt.gca().set_xticklabels([])
33
34 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
```

```

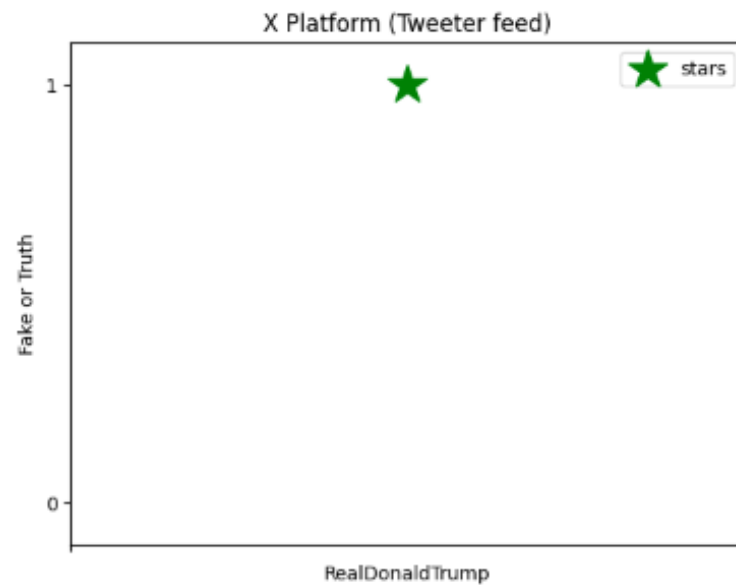
26 plt.title('X Platform (Tweeter feed)')
27 # showing legend
28 plt.legend()
29 # Setting the axis range
30 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
31 plt.ylim(-0.1, 1.1)
32 plt.gca().set_xticklabels([])
33
34 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
35 plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
36 plt.xticks([-1, 1])
37 plt.yticks([0, 1])
38
39 # function to show the plot
40 plt.show()

```



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

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



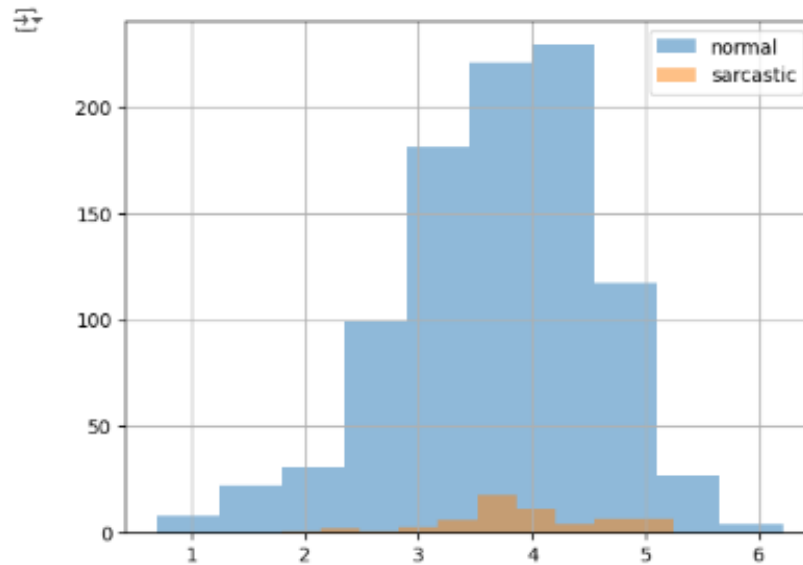
Create memory for metrics i.e., save a file and add feature 1, 2(in next notebook), 3(in next notebook after notebook 2)

```
1 # save prediction values to csv file to be shared with other metrics to follow i.e., stance and mis/disinformation
2
3 df_new.to_csv('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', encoding="utf-8")
4
5 # df_new.save_csv("data_to_graph")
6
7 # import csv
8
9 # # field names
10 # fields = ['Author', 'Text', 'clean_text', 'clean_joined_text', 'target'] # ['Name', 'Branch', 'Year', 'CGPA']
11
12 # # data rows of csv file
13 # rows = df_new # [ ['Nikhil', 'COE', '2', '9.0'],
14 #                  # ['Sanchit', 'COE', '2', '9.1'],
15 #                  # ['Aditya', 'IT', '2', '9.3'],
16 #                  # ['Sagar', 'SE', '1', '9.5'],
17 #                  # ['Prateek', 'MCE', '3', '7.8'],
18 #                  # ['Sahil', 'EP', '2', '9.1']]
19
20 # # name of csv file
21 # filename = "/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv"
22
23 # # writing to csv file
24 # with open(filename, 'w') as csvfile:
25 #     # creating a csv writer object
26 #     csvwriter = csv.writer(csvfile)
27
28 #     # writing the fields
29 #     csvwriter.writerow(fields)
30
31 #     # writing the data rows
32 #     csvwriter.writerows(rows)
33
34
35 # df_new.info()
36 # df_new.predicted_true_fake
```

6.2 Sarcasm

✓ Exploratory data analysis (EDA)

```
[ ] 1 # histogram 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)
```

	size	mean	sum	median	std	min	max
subreddit							
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

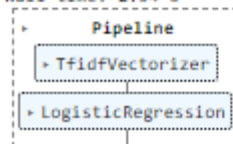
Create pipeline and train the TF-IDF + Logistic Regression model

✓ Training the model

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

```
1 %%time
2 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

```
[ ] 1 # wordcloud
2
3 from wordcloud import WordCloud, STOPWORDS
```

```
[ ] 1 wordcloud = WordCloud(background_color='black', stopwords = STOPWORDS,
2                             max_words = 200, max_font_size = 100,
3                             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("Precision score = ", precision_score_value)
```

```
→ Accuracy score = 0.932
   F1_score = 0.48240165631469983
   Precision 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'])
```

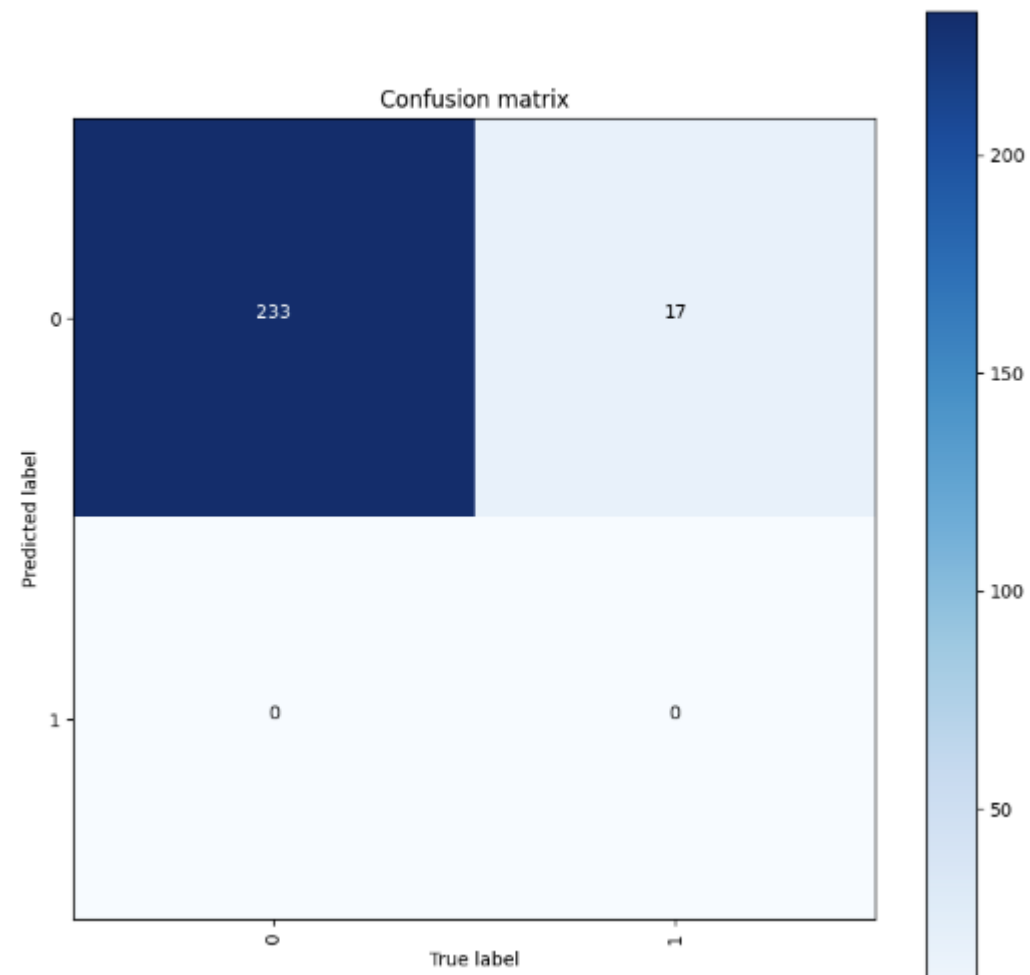
▼ Confusion Matrix

```
[ ] 1 def plot_confusion_matrix(actual, predicted, classes,
2     |                         normalize=False,
3     |                         title='Confusion matrix', figsize=(7,7),
4     |                         cmap=plt.cm.Blues, path_to_save_fig=None):
5     |
6     |     """
7     |     This function prints and plots the confusion matrix.
8     |     Normalization can be applied by setting 'normalize=True'.
9     |     """
10    |     import itertools
11    |     from sklearn.metrics import confusion_matrix
12    |     cm = confusion_matrix(actual, predicted).T
13    |     if normalize:
14    |         | cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
15    |
16    |     plt.figure(figsize=figsize)
17    |     plt.imshow(cm, interpolation='nearest', cmap=cmap)
18    |     plt.title(title)
19    |     plt.colorbar()
20    |     tick_marks = np.arange(len(classes))
21    |     plt.xticks(tick_marks, classes, rotation=90)
22    |     plt.yticks(tick_marks, classes)
23    |
24    |     fmt = '.2f' if normalize else 'd'
25    |     thresh = cm.max() / 2.
26    |     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
27    |         | plt.text(j, i, format(cm[i, j], fmt),
28    |         |         |         horizontalalignment="center",
29    |         |         |         color="white" if cm[i, j] > thresh else "black")
30    |
31    |     plt.tight_layout()
32    |     plt.ylabel('Predicted label')
33    |     plt.xlabel('True label')
34    |
35    |     if path_to_save_fig:
36    |         | 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))
```

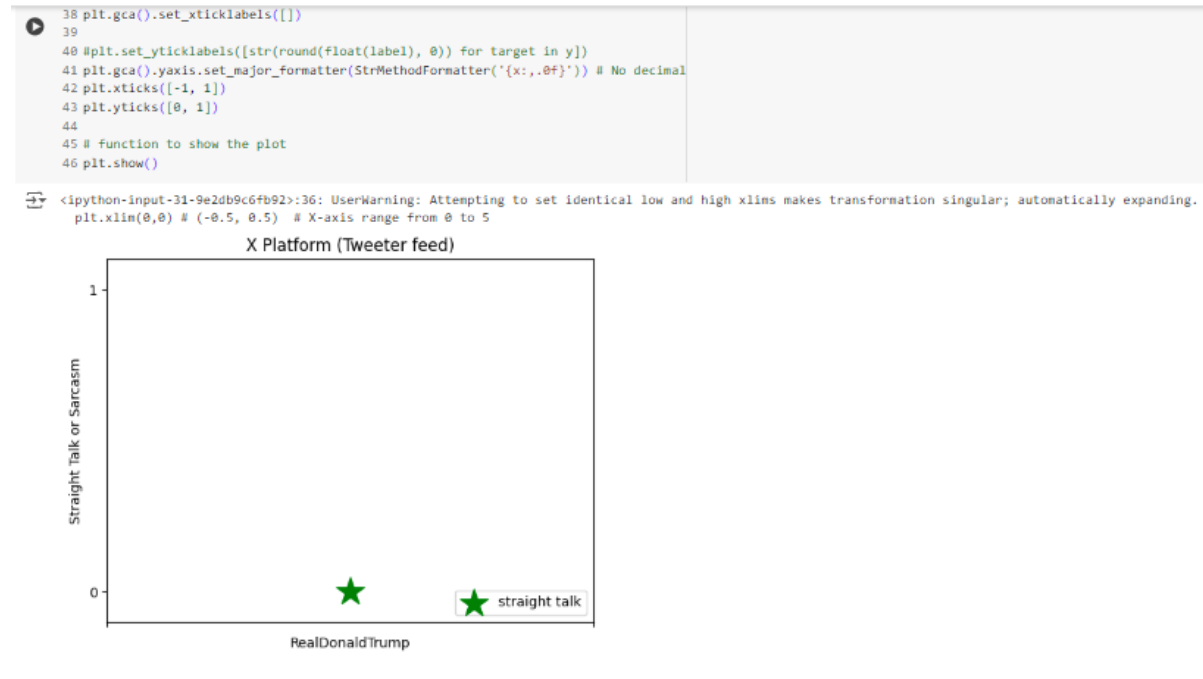
[]



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

```
[ ] 1 # TO BE POINTED AT THE NEW DATA #
    2 # TO BE POINTED AT THE NEW DATA #
    3 # TO BE POINTED AT THE NEW DATA # - originally from Truth Fake module
    4
    5 import matplotlib.pyplot as plt
    6 from matplotlib.ticker import StrMethodFormatter
    7
    8 # x-axis values
    9 x = [0] # [1,2,3,4,5,6,7,8,9,10]
   10 # y-axis values
   11 y = df_new_texts['predicted_sarcasm'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
   12
   13
   14 if y == 1:
   15     # plotting points as a scatter plot
   16     plt.scatter(x, y, label= "rasberry talk", color= "red",
   17                marker= "X", s=450)
   18     plt.legend(loc='upper right')
   19
   20 if y == 0:
   21     # plotting points as a scatter plot
   22     plt.scatter(x, y, label= "straight talk", color= "green",
   23                marker= "*", s=450)
   24     plt.legend(loc='lower right')
   25
   26
   27 # x-axis label
   28 plt.xlabel('RealDonaldTrump')
   29 # frequency label
   30 plt.ylabel('Straight Talk or Sarcasm')
   31 # plot title
```

a



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

```
[ ] 1 # LSTM
    2
    3 # Text pre-processing
    4 # import tensorflow as tf
    5 # from tensorflow.keras.preprocessing.text import Tokenizer
    6 # from tensorflow.keras.preprocessing.sequence import pad_sequences
    7 # from tensorflow.keras.callbacks import EarlyStopping
    8 # # Modeling
    9 # from tensorflow.keras.models import Sequential
    10 # from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional
    11
    12
    13 # !pip install keras
    14 # import keras.preprocessing.text Tokenizer
    15 # from keras.preprocessing.sequence import pad_sequences
    16 # from keras.layers import Embedding, LSTM, Dense
    17 # from keras.models import Sequential
    18
    19 import keras
    20 import tensorflow as tf
    21 from tensorflow.keras.preprocessing.text import Tokenizer
    22 from tensorflow.keras.preprocessing.sequence import pad_sequences
    23 from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional
    24 from keras.models import Sequential
    25
    26
    27 # train_texts, valid_texts, y_train, y_valid = \
    28 #     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)

```
64 # Compile the model
65 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # 'binary_crossentropy'
66
67 # train_texts, valid_texts, y_train, y_valid
68
69 # Train the model
70 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

```
[ ] 1 # TO BE POINTED AT THE NEW DATA #
    2 # TO BE POINTED AT THE NEW DATA #
    3 # TO BE POINTED AT THE NEW DATA # - originally from Truth Fake module
    4
    5 import matplotlib.pyplot as plt
    6 from matplotlib.ticker import StrMethodFormatter
    7
    8
    9 plt.subplot(1, 2, 1) # row 1, column 2, count 1
   10 # x-axis values
   11 x = [0] # [1,2,3,4,5,6,7,8,9,10]
   12 # y-axis values
   13 y = df_new_texts['predicted_true_fake'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
   14
   15
   16 if y == 0:
   17     # plotting points as a scatter plot
   18     plt.scatter(x, y, label= "rasberry", color= "red",
   19               | | | | marker= "X", s=450)
   20
   21 if y == 1:
   22     # plotting points as a scatter plot
   23     plt.scatter(x, y, label= "stars", color= "green",
   24               | | | | marker= "*", s=450)
   25
   26
   27 # x-axis label
   28 plt.xlabel('RealDonaldTrump')
   29 # frequency label
   30 plt.ylabel('Fake or Truth')
   31 # plot title
   32 plt.title('X Platform (Twitter feed)')
   33 # showing legend
   34 plt.legend()
   35 # Setting the axis range
   36 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
   37 plt.ylim(-0.1, 1.1)
   38 plt.gca().set_xticklabels([])
   39
   40 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
```

```

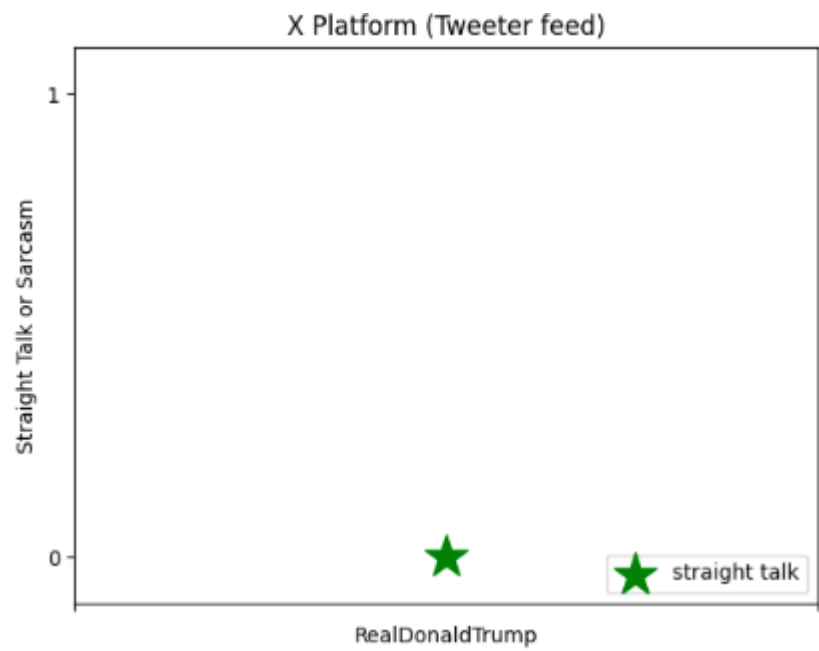
37 plt.ylim(-0.1, 1.1)
38 plt.gca().set_xticklabels([])
39
40 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
41 plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
42 plt.xticks([-1, 1])
43 plt.yticks([0, 1])
44
45
46 plt.subplot(1, 2, 2) # row 1, column 2, count 1
47 # x-axis values
48 x = [0] # [1,2,3,4,5,6,7,8,9,10]
49 # y-axis values
50 #y = df_new_texts['target'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
51 #y = int(prediction[0]) # [2,4,5,7,6,8,9,11,12,12]
52 y = prediction[0] # [2,4,5,7,6,8,9,11,12,12]
53
54 if y >= 0.99:
55     # plotting points as a scatter plot
56     plt.scatter(x, y, label= "rasberry talk", color= "red",
57                marker= "X", s=450)
58     plt.legend(loc='upper right')
59
60 if y == 0:
61     # plotting points as a scatter plot
62     plt.scatter(x, y, label= "straight talk", color= "green",
63                marker= "*", s=450)
64     plt.legend(loc='lower right')
65
66 # x-axis label
67 plt.xlabel('RealDonaldTrump')
68 # frequency label
69 plt.ylabel('Straight Talk or Sarcasm')
70 # plot title
71 plt.title('X Platform (Twitter feed)')
72 # showing legend
73 #plt.legend()
74 #plt.legend(['Legend'], loc='upper left')
75 # Setting the axis range
76 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
77 plt.ylim(-0.1, 1.1)

```

```

75 # Setting the axis range
76 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
77 plt.ylim(-0.1, 1.1)
78 plt.gca().set_xticklabels([])
79
80 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
81 plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
82 plt.xticks([-1, 1])
83 plt.yticks([0, 1])
84
85 # function to show the plot
86 #plt.show()
87
88 # function to show the plot
89 plt.show()

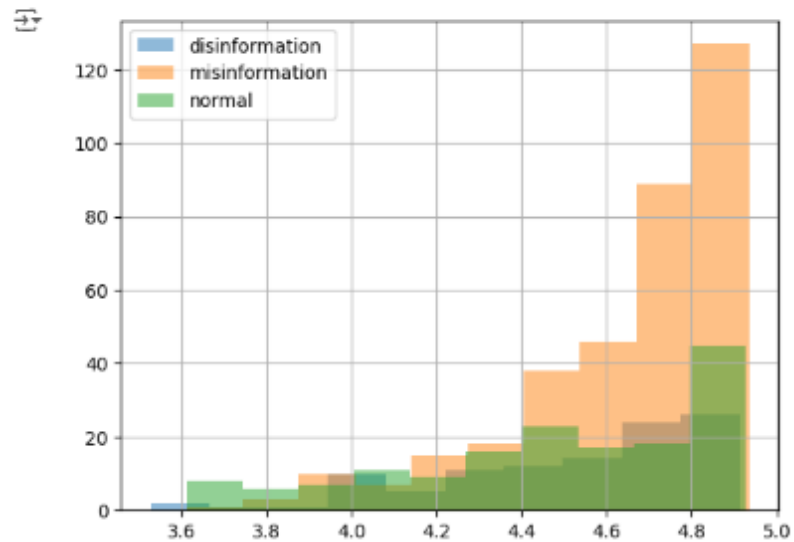
```



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)

```

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

```

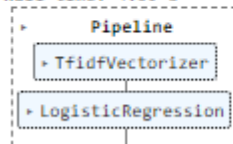
Build pipeline

```

1 %%time
2 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)

```

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("Precision score = ", precision_score_value)
7 recall_score_value = recall_score(y_valid, valid_pred, average='macro')
8 print("Recall score = ", recall_score_value)
9

```

Accuracy score = 0.6064516129032258
 F1_score = 0.3799705262772204
 Precision 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/metrics

```
[ ] 1 # NEW SECTION
    2
    3 df_new_texts['predicted_misinformation'] = 0
    4 df_new_texts['predicted_disinformation'] = 0
    5
    6 for i in range(len(predicted_value)):
    7     if predicted_value[i] == 1:
    8         df_new_texts.loc[i, 'predicted_misinformation'] = 1
    9     elif predicted_value[i] == 2:
   10         df_new_texts.loc[i, 'predicted_disinformation'] = 1
   11
   12 df_new_texts = df_new_texts[['id', 'content', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_misinformation', 'predicted_disinformation']]
   13
   14 #print(df_new_texts['predicted_disinformation'])
```


▼ Model

```
1 def plot_confusion_matrix(actual, predicted, classes,  
2                             normalize=False,  
3                             title='Confusion matrix', figsize=(7,7),  
4                             cmap=plt.cm.Blues, path_to_save_fig=None):  
5     """  
6     This function prints and plots the confusion matrix.  
7     Normalization can be applied by setting `normalize=True`.  
8     """  
9     import itertools  
10    from sklearn.metrics import confusion_matrix  
11    cm = confusion_matrix(actual, predicted).T  
12    if normalize:  
13        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
14  
15    plt.figure(figsize=figsize)  
16    plt.imshow(cm, interpolation='nearest', cmap=cmap)  
17    plt.title(title)  
18    plt.colorbar()  
19    tick_marks = np.arange(len(classes))  
20    plt.xticks(tick_marks, classes, rotation=90)  
21    plt.yticks(tick_marks, classes)  
22  
23    fmt = '.2f' if normalize else 'd'  
24    thresh = cm.max() / 2.  
25    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
26        plt.text(j, i, format(cm[i, j], fmt),  
27                horizontalalignment="center",  
28                color="white" if cm[i, j] > thresh else "black")  
29  
30    plt.tight_layout()  
31    plt.ylabel('Predicted label')  
32    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')
```

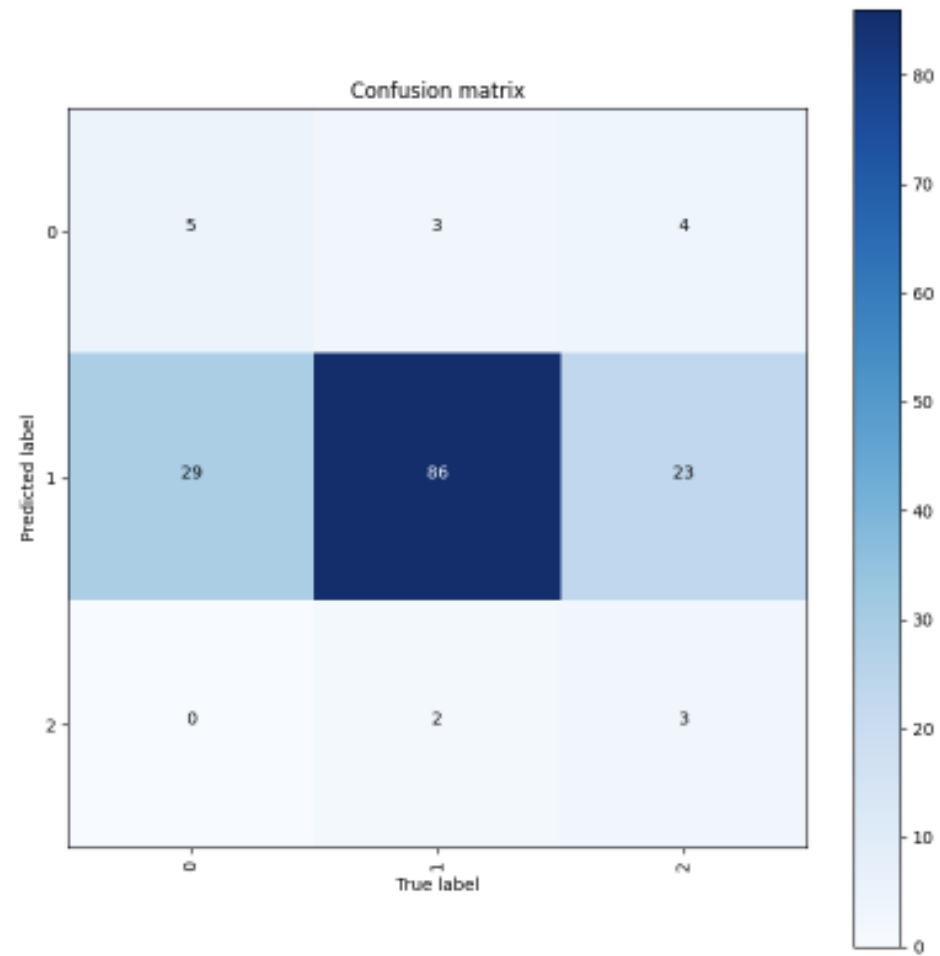
Read in memory file of previous notebooks features, Truth-Fake, Sarcasm then write the new feature Stance to the memory file

```
1 import csv
2
3 # Step 1: Read the existing CSV file and store its content
4 csv_file_path = 'data_to_graph.csv'
5 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
6     reader = csv.reader(file)
7     data = list(reader)
8
9 # Step 2: Define the values for the new column
10 new_values = valid_pred # df_new_texts['predicted_sarcasm'] # ['New York', 'Los Angeles', 'Chicago', 'San Francisco']
11
12 # Step 3: Add the new column header to the first row of the data
13 data[0].append('predicted_stance')
14
15 # Step 4: Add the new column values to the remaining rows of the data
16 for i in range(1, len(data)):
17     data[i].append(new_values[i - 1])
18
19 # Step 5: Write the updated data back to the CSV file
20 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
21     writer = csv.writer(file)
22     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

```
1 # TO BE POINTED AT THE NEW DATA #
2 # TO BE POINTED AT THE NEW DATA #
3 # TO BE POINTED AT THE NEW DATA # - originally from Truth Fake module
4
5 import time
6 import matplotlib.pyplot as plt
7 from matplotlib.ticker import StrMethodFormatter
8
9 # measure User ; Sys ; and Wall time for evaluation
10 %%time
11 start_time = time.process_time()
12 # Code to measure
13
14 plt.subplot(1, 3, 1) # row 1, column 2, count 1
15 # x-axis values
16 x = [0] # [1,2,3,4,5,6,7,8,9,10]
17 # y-axis values
18
19 df_new_texts = pd.read_csv('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv')
20 y = df_new_texts['predicted_true_fake'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
21
22 if y == 0:
23     # plotting points as a scatter plot
24     plt.scatter(x, y, label= "fake tweet", color= "red",
25                marker= "X", s=450)
26     plt.legend(loc='lower left')
27 if y >= 0.99:
28     # plotting points as a scatter plot
29     plt.scatter(x, y, label= "true tweet", color= "green",
30                marker= "+", s=450)
31     plt.legend(loc='upper right')
32
33 # x-axis label
34 plt.xlabel('RealDonaldTrump')
35 # frequency label
36 plt.ylabel('Fake or Truth')
37
```

```

35 # frequency label
36 plt.ylabel('Fake or Truth')
37 # plot title
38 plt.title('X Platform')
39 # showing legend
40 plt.legend()
41 # Setting the axis range
42 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
43 plt.ylim(-0.1, 1.1)
44 plt.gca().set_xticklabels([])
45
46 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
47 plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
48 plt.xticks([-1, 1])
49 plt.yticks([0, 1])
50
51
52 plt.subplot(1, 3, 2) # row 1, column 2, count 1
53 # x-axis values
54 x = [0] # [1,2,3,4,5,6,7,8,9,10]
55 # y-axis values
56 #y = df_new_texts['target'].iloc[0] # [2,4,5,7,6,8,9,11,12,12]
57 #y = int(prediction[0]) # [2,4,5,7,6,8,9,11,12,12]
58 #y = prediction[0] # [2,4,5,7,6,8,9,11,12,12]
59 y = df_new_texts['predicted_sarcasm'].iloc[0]
60
61
62 if y >= 0.99:
63     # plotting points as a scatter plot
64     plt.scatter(x, y, label= "direct tweet", color= "green",
65     | | | | marker= "*", s=450)
66     plt.legend(loc='upper right')
67 if y == 0:
68     # plotting points as a scatter plot
69     plt.scatter(x, y, label= "sarcastic tweet", color= "red",
70     | | | | marker= "x", s=450)
71     plt.legend(loc='lower right')
72
73 # x-axis label
74 plt.xlabel('RealDonaldTrump')

```

```

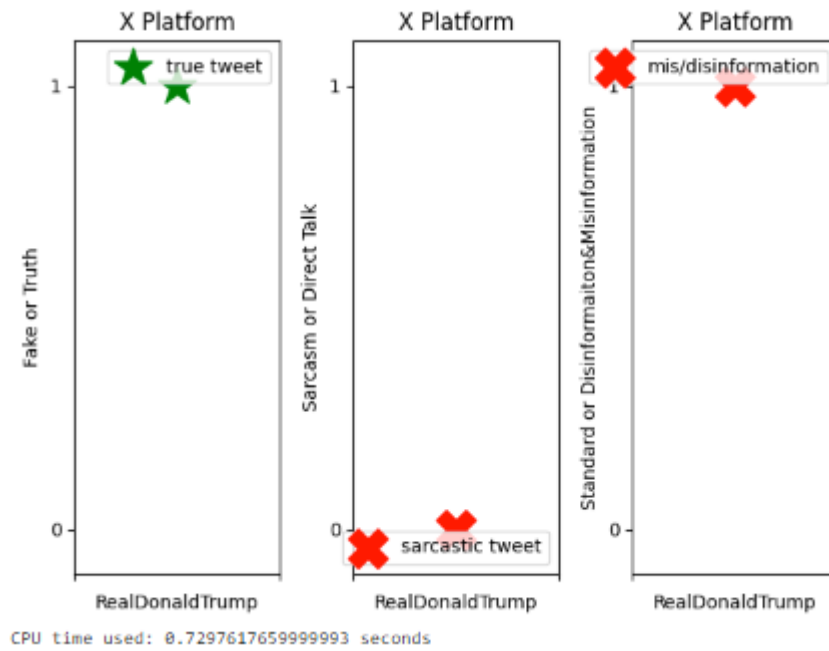
73 # x-axis label
74 plt.xlabel('RealDonaldTrump')
75 # frequency label
76 plt.ylabel('Sarcasm or Direct Talk')
77 # plot title
78 plt.title('X Platform')
79 # showing legend
80 plt.legend()
81 #plt.legend(['Legend'], loc='upper left')
82 # Setting the axis range
83 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
84 plt.ylim(-0.1, 1.1)
85 plt.gca().set_xticklabels([])
86
87 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
88 plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
89 plt.xticks([-1, 1])
90 plt.yticks([0, 1])
91
92 # function to show the plot
93 plt.show()
94
95
96 plt.subplot(1, 3, 3) # row 1, column 2, count 1
97 # x-axis values
98 x = [0] # [1,2,3,4,5,6,7,8,9,10]
99 # y-axis values
100
101 #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]
102 y = df_new_texts['predicted_stance'].iloc[0]
103
104 if y == 0:
105     # plotting points as a scatter plot
106     plt.scatter(x, y, label= "neutral tweet", color= "green",
107                marker= "star", s=450)
108     plt.legend(loc='lower left')
109 if y >= 0.99:
110     # plotting points as a scatter plot

```

```

109 if y >= 0.99:
110     # plotting points as a scatter plot
111     plt.scatter(x, y, label= "mis/disinformation", color= "red",
112                marker= "X", s=450)
113     plt.legend(loc='upper right')
114
115 # x-axis label
116 plt.xlabel('RealDonaldTrump')
117 # frequency label
118 plt.ylabel('Standard or Disinformaton&Misinformation')
119 # plot title
120 plt.title('X Platform')
121 # showing legend
122 plt.legend()
123 # Setting the axis range
124 plt.xlim(0,0) # (-0.5, 0.5) # X-axis range from 0 to 5
125 plt.ylim(-0.1, 1.1)
126 plt.gca().set_xticklabels([])
127
128 #plt.set_yticklabels([str(round(float(label), 0)) for target in y])
129 plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}')) # No decimal
130 plt.xticks([-1, 1])
131 plt.yticks([0, 1])
132
133
134 # function to show the plot
135 plt.tight_layout()
136 plt.show()
137
138 end_time = time.process_time()
139 print(f"CPU time used: {end_time - start_time} 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**

```
[ ] 1 # LSTM
2
3 # Text pre-processing
4 # import tensorflow as tf
5 # from tensorflow.keras.preprocessing.text import Tokenizer
6 # from tensorflow.keras.preprocessing.sequence import pad_sequences
7 # from tensorflow.keras.callbacks import EarlyStopping
8 # # Modeling
9 # from tensorflow.keras.models import Sequential
10 # from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional
11
12
13 # !pip install keras
14 # import keras.preprocessing.text Tokenizer
15 # from keras.preprocessing.sequence import pad_sequences
16 # from keras.layers import Embedding, LSTM, Dense
17 # from keras.models import Sequential
18
19 import keras
20 import tensorflow as tf
21 from tensorflow.keras.preprocessing.text import Tokenizer
22 from tensorflow.keras.preprocessing.sequence import pad_sequences
23 from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding, Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D, Bidirectional
24 from keras.models import Sequential
25 from keras.metrics import F1Score # (average=None, threshold=None, name="f1_score", dtype=None)
--
```

[+ Code](#)[+ Text](#)

```

25 from keras.metrics import F1Score # (average=None, threshold=None, name="f1_score", dtype=None)
26
27
28 # The input text, example could be list of sentences
29 texts = train_texts[:] # [...]
30
31 # The labels corresponding to the input text
32 labels = y_train[:] # train_df['label'] # train_texts['Author'] # [...]
33
34 # Hyperparameters
35 max_words = 10000 # max number of words to use in the vocabulary
36 max_len = 200 # max length of each text (in terms of number of words)
37 embedding_dim = 200 # dimension of word embeddings
38 lstm_units = 64 # number of units in the LSTM layer
39 num_classes = len(set(labels)) # 2 # len(y_train) # y_train.unique() # 2 # len(set(labels)) # number of classes
40
41 # Tokenize the texts and create a vocabulary
42 tokenizer = Tokenizer(num_words=max_words)
43 tokenizer.fit_on_texts(texts.values.astype('U'))
44 sequences = tokenizer.texts_to_sequences(texts.values.astype('U'))
45
46 # Pad the sequences so they all have the same length
47 x = pad_sequences(sequences, maxlen=max_len)
48
49 # Create one-hot encoded labels
50 y = keras.utils.to_categorical(labels,
51 | | | | | | | | | | | | | | num_classes)
52
53 # Build the model
54 model = Sequential()
55 model.add(Embedding(max_words, embedding_dim, input_length=max_len))
56 model.add(LSTM(lstm_units))
57 model.add(Dense(num_classes, activation='softmax')) # 'relu' ; "rmsprop"
58
59 # optimizer='adam', loss='mse'
57 model.add(Dense(num_classes, activation='softmax')) # 'relu' ; "rmsprop"
58
59 # optimizer='adam', loss='mse'
60
61
62 # Compile the model
63 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # 'binary_crossentropy'
64
65 # train_texts, valid_texts, y_train, y_valid
66
67 # Train the model
68 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 demonstration)

```
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

```
[ ] 1 # Using NEW data -> predict using NEW model LSTM prviously trained on reference ground truth labelled stance data
2 # write Stance prediction results to shared dataframe
3
4 import numpy as np
5
6 #X = df_new_texts['Text']
7
8 # Tokenize the texts and create a vocabulary
9 tokenizer = Tokenizer(num_words=max_words)
10 #tokenizer.fit_on_texts(texts.values.astype('U'))
11 tokenizer.fit_on_texts(df_new_texts['content'].values.astype('U'))
12 # sequences = tokenizer.texts_to_sequences(texts.values.astype('U'))
13 sequences = tokenizer.texts_to_sequences(df_new_texts['content'].values.astype('U'))
14
15 # Pad the sequences so they all have the same length
16 X = pad_sequences(sequences, maxlen=max_len)
17
18 prediction = []
19 prediction = model.predict(X)
20
21 # Creating multi-dimension array
22 #array1 = [1, 2, 4, [5, [6, 7]]]
23
24 # Object Data type is accept all data-type
25 Data_type = int
26
27 # Now we fix the error
28 #np_array = numpy.array(prediction, dtype=Data_type)
29 # TEMP commented out
30 #prediction = np.array(prediction, dtype=Data_type)
31 #prediction = prediction.flatten('C')
32
33
34 df_new_texts['predicted_stance_neutral'] = 0
35 df_new_texts['predicted_stance_misinformation'] = 0
36 df_new_texts['predicted_stance_disinformation'] = 0
37 df_new_texts['predicted_stance_all_types'] = 0
38
39
```

```

36 df_new_texts['predicted_stance_disinformation'] = 0
37 df_new_texts['predicted_stance_all_types'] = 0
38
39
40 #for i in range(len(prediction)):
41 i = 0
42 for p in prediction[:, 0]:
43     if p >= 0.50:
44         #print(p)
45         df_new_texts.loc[i, 'predicted_stance_neutral'] = 1
46         i += 1
47
48 i = 0
49 for p in prediction[:, 1]:
50     if p >= 0.50:
51         df_new_texts.loc[i, 'predicted_stance_misinformation'] = 1
52         i += 1
53
54 i = 0
55 for p in prediction[:, 2]:
56     if p >= 0.50:
57         df_new_texts.loc[i, 'predicted_stance_disinformation'] = 1
58         i += 1
59
60 i = 0
61 #for p in prediction[:, 2]:
62 for i in range(len(df_new_texts)):
63     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:
64         df_new_texts.loc[i, 'predicted_stance_all_types'] = 0
65     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:
66         df_new_texts.loc[i, 'predicted_stance_all_types'] = 1
67     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:
68         df_new_texts.loc[i, 'predicted_stance_all_types'] = 2
69     i += 1
70
71
72 df_new_texts = df_new_texts[['id', 'content', 'clean_text', 'clean_joined_text', 'predicted_true_fact', 'predicted_sarcasm', 'predicted_stance_neutral', 'predicted_stance_misinformation', 'predicted_stance_disinformation', 'predicted_stance_all_types']].copy()
73
74 #df_new_texts.drop(index='predicted_stance', axis=0, inplace=True)
75 print(df_new_texts)
76 df_new_texts.info()
77 df_new_texts.describe()
78 print(len(prediction))
79 #print(len(df_new_texts['predicted_stance']))
80 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 @ sean Hannity tonight a...
9723 1273442195161387008 pic.twitter.com/3lm1spbU8X
9724 1273442469066276864 pic.twitter.com/vpCESMadUz
9725 1273442528411385858 pic.twitter.com/VLlc0BHW41
```

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

✓ **IMPORTANT - Only run ONCE**

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

```
1 # IMPORTANT - Only run ONCE
2 # save shared data frame with Saracasm prediction results added to the shared file
3
4 import csv
5
6 # # Step 1: Read the existing CSV file and store its content
7 # csv_file_path = 'data_to_graph.csv'
8 # with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
9 #     reader = csv.reader(file)
10 #     data = list(reader)
11
12 # # Step 2: Define the values for the new "City" column
13 # new_values = df_new_texts['predicted_sarcasm'] # ['New York', 'Los Angeles', 'Chicago', 'San Francisco']
14
15 # # Step 3: Add the new "City" column header to the first row of the data
16 # data[0].append('predicted_sarcasm')
17
18 # # Step 4: Add the new "City" column values to the remaining rows of the data
19 # for i in range(1, len(data)):
20 #     data[i].append(new_values[i - 1])
21
```



```

20 #     data[i].append(new_values[i - 1])
21
22 # # Step 5: Write the updated data back to the CSV file
23 # with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
24 #     writer = csv.writer(file)
25 #     writer.writerows(data)
26
27
28 # Step 1: Read the existing CSV file and store its content
29 csv_file_path = 'data_to_graph.csv'
30 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
31     reader = csv.reader(file)
32     data = list(reader)
33 #Step 2 , 3, 4 repeated
34 new_values = df_new_texts['predicted_stance_neutral']
35 data[0].append('predicted_stance_neutral')
36 #data = data[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm']].copy()
37 for i in range(1, len(data)):
38     data[i].append(new_values[i - 1])
39 # Step 5: Write the updated data back to the CSV file
40 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
41     writer = csv.writer(file)
42     writer.writerows(data)
43
44 # Step 1: Read the existing CSV file and store its content
45 csv_file_path = 'data_to_graph.csv'
46 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
47     #file = file[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_stance_neutral']].copy()
48     reader = csv.reader(file)
49     data = list(reader)
50 #Step 2, 3, 4 repeated
51 new_values = df_new_texts['predicted_stance_misinformation']
52 data[0].append('predicted_stance_misinformation')
53 for i in range(1, len(data)):
54     data[i].append(new_values[i - 1])
55 # Step 5: Write the updated data back to the CSV file
56 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
57     writer = csv.writer(file)
58     writer.writerows(data)

```

```

58 | writer.writerow(data)
59
60 # Step 1: Read the existing CSV file and store its content
61 csv_file_path = 'data_to_graph.csv'
62 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
63     #file = file[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_stance_neutral', 'predicted_stance_misinformation']].copy()
64     reader = csv.reader(file)
65     data = list(reader)
66 #Step 2, 3, 4 repeated
67 new_values = df_new_texts['predicted_stance_disinformation']
68 data[0].append('predicted_stance_disinformation')
69 for i in range(1, len(data)):
70     data[i].append(new_values[i - 1])
71 # Step 5: Write the updated data back to the CSV file
72 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
73     writer = csv.writer(file)
74     writer.writerow(data)
75
76
77 # Step 1: Read the existing CSV file and store its content
78 csv_file_path = 'data_to_graph.csv'
79 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'r') as file:
80     #file = file[['Author', 'Text', 'clean_text', 'clean_joined_text', 'predicted_true_fake', 'predicted_sarcasm', 'predicted_stance_neutral', 'predicted_stance_misinformation', 'predicted_stance_disinformation']].copy()
81     reader = csv.reader(file)
82     data = list(reader)
83 #Step 2, 3, 4 repeated
84 new_values = df_new_texts['predicted_stance_all_types']
85 data[0].append('predicted_stance_all_types')
86 for i in range(1, len(data)):
87     data[i].append(new_values[i - 1])
88 # Step 5: Write the updated data back to the CSV file
89 with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
90     writer = csv.writer(file)
91     writer.writerow(data)
92
93 # # Step 5: Write the updated data back to the CSV file
94 # with open('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv', 'w', newline='') as file:
95 #     writer = csv.writer(file)
96 #     writer.writerow(data)
97 # # Step 5: Write the updated data back to the CSV file
98
99 print('predicted_sarcasm = ', df_new_texts['predicted_sarcasm'].sum())
100 print('predicted_stance_neutral = ', df_new_texts['predicted_stance_neutral'].sum())
101 print('predicted_stance_misinformation = ', df_new_texts['predicted_stance_misinformation'].sum())
102 print('predicted_stance_disinformation = ', df_new_texts['predicted_stance_disinformation'].sum())
103 print('predicted_stance_all_types = ', df_new_texts['predicted_stance_all_types'].sum())
104 #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

```
[ ] 1 # NEW combined chart
    2
    3 # importing package
    4 import matplotlib.pyplot as plt
    5 import numpy as np
    6 import pandas as pd
    7
    8 from matplotlib.ticker import StrMethodFormatter
    9
    10 import io
    11 import xml.etree.ElementTree as ET
    12 import time
    13
    14 from matplotlib.lines import Line2D
    15 from matplotlib.markers import MarkerStyle
    16 from matplotlib.transforms import Affine2D
    17
    18 from google.colab import drive
    19 drive.mount('/content/drive/')
    20
    21 # x-axis values
    22 x = [0] # [1,2,3,4,5,6,7,8,9,10]
    23 # y-axis values
    24
    25 # measure User ; Sys ; and Wall time for evaluation
    26
    27 start_time = time.process_time()
    28 # Code to measure
    29
    30 df_new_texts = pd.read_csv('/content/drive/MyDrive/Colab_Notebooks/MSCAITOP/data_to_graph.csv')
    31
    32
    33 for i in range(10):
    34     y1 = df_new_texts['predicted_true_fake'].iloc[i] # [2,4,5,7,6,8,9,11,12,12]
    35     y2 = df_new_texts['predicted_sarcasm'].iloc[i]
    36     #y3 = df_new_texts['predicted_stance_neutral'].iloc[i]
    37     y3 = df_new_texts['predicted_stance_all_types'].iloc[i]
    38
    39     # 'predicted_stance_misinformation'
    40     # 'predicted_stance_disinformation'
```

```

42 #y4 = df_new_texts['predicted_stance_misinformation'].iloc[0]
43 #y5 = df_new_texts['predicted_stance_disinformation'].iloc[0]
44
45 # create data
46 # x = [1,2,3,4,5]
47 # y = [3,3,3,3,3]
48
49 # text_style = dict(horizontalalignment='right', verticalalignment='center',
50 #                    fontsize=12, fontfamily='monospace')
51 # marker_style = dict(linestyle=':', color='0.8', markersize=10,
52 #                    markerfacecolor="tab:blue", markeredgecolor="tab:blue")
53
54 # y1 = [0]
55 # y2 = [0]
56 # y3 = [0]
57 # all good / all zeros / true news 0 / straight talk 0/ neutral stance 0
58 if y1 == [0] and y2 == [0] and y3 == [0]:
59     #print('Test = if block runs')
60     # plot lines
61     x1 = [3,0]
62     y1 = [3,0]
63     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
64     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
65     plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
66     plt.text(-3, -7, 'True News')
67     x2 = [3,0]
68     y2 = [5,0]
69     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
70     plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
71     plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
72     plt.text(-3, -7, 'Direct Talk')
73     x3 = [3,0]
74     y3 = [7,0]
75     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
76     plt.plot(x3, y3, label = "Stance_Neutral", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
77     plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")

```

```

81 # y1 = [1]
82 # y2 = [1]
83 # y3 = [1]
84 # all up / all ones / all fake 1 / sarcasm 1/ all misinformation 1
85 if y1 == [1] and y2 == [1] and y3 == [1]:
86     #print('Test = if block runs')
87     # plot lines
88     x1 = [-3,0]
89     y1 = [-3,0]
90     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
91     plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
92     plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
93     plt.text(-3, -3, 'Fake News')
94     x2 = [-3,0]
95     y2 = [-5,0]
96     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
97     plt.plot(x2, y2, label = "Sarcasm", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
98     plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
99     plt.text(-3, -5, 'Sarcasm')
100     x3 = [-3,0]
101     y3 = [-7,0]
102     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
103     plt.plot(x3, y3, label = "Misinformation", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
104     plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
105     plt.text(-3, -7, 'Misinformation')
106
107 # y1 = [1]
108 # y2 = [0]
109 # y3 = [1]
110 # all good / all zeros / true news 0 / straight talk 0/ neutral stance 0
111 if y1 == [1] and y2 == [0] and y3 == [1]:
112     #print('Test = if block runs')

```

```

111 if y1 == [1] and y2 == [0] and y3 == [1]:
112     #print('Test = if block runs')
113     # plot lines
114     x1 = [-3,0]
115     y1 = [-3,0]
116     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
117     plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
118     plt.plot(-3, -3, marker="o", markersize=20, markeredgewidth="red", markerfacecolor="red")
119     plt.text(-3, -3, 'Fake News')
120     x2 = [3,0]
121     y2 = [5,0]
122     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
123     plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
124     plt.plot(3, 5, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
125     plt.text(3, 5, 'Direct Talk')
126     x3 = [-3,0]
127     y3 = [-7,0]
128     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
129     plt.plot(x3, y3, label = "Stance Misinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
130     plt.plot(-3, -7, marker="o", markersize=20, markeredgewidth="red", markerfacecolor="red")
131     plt.text(-3, -7, 'Misinformation')
132
133 # y1 = [0]
134 # y2 = [0]
135 # y3 = [1]
136 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
137 if y1 == [0] and y2 == [0] and y3 == [1]:
138     x1 = [3,0]
139     y1 = [3,0]
140     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
141     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
142     plt.plot(3, 3, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
143     plt.text(3, 3, 'True News')
144     x2 = [3,0]
145     y2 = [5,0]
146     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
147     plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
148     plt.plot(3, 5, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
149     plt.text(3, 5, 'Direct Talk')

```

```

151 y3 = [-7,0]
152 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
153 plt.plot(x3, y3, label = "Misinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
154 plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
155 plt.text(-3, -7, 'Misinformation')
156
157 plt.legend(loc='lower right')
158 from matplotlib import pyplot as plt
159 plt.savefig('/content/drive/My Drive/Colab_Notebooks/MSCAITOP/MSCAITOP_graphic_v2a.png', bbox_inches='tight')
160
161
162 # y1 = [0]
163 # y2 = [1]
164 # y3 = [1]
165 # passed testing
166 # Mixed / all fake 0/sarcasm 1/all misinformation&disinformation 1
167 if y1 == [0] and y2 == [1] and y3 == [1]:
168     x1 = [3,0]
169     y1 = [3,0]
170     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
171     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
172     plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
173     plt.text(3, 3, 'True News')
174     x2 = [-3,0]
175     y2 = [-5,0]
176     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
177     plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
178     plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
179     plt.text(-3, -5, 'Sarcasm Talk')
180     x3 = [-3,0]
181     y3 = [-7,0]
182     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
183     plt.plot(x3, y3, label = "Misinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
184     plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
185     plt.text(-3, -7, 'Misinformation')

```



```

184 plt.plot(-3, -7, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
185 plt.text(-3, -7, 'Misinformation')
186
187 # y1 = [0]
188 # y2 = [1]
189 # y3 = [0]
190 # Mixed / all fake 0/sarcasm 1/all misinformation&disinformation 0
191 if y1 == [0] and y2 == [1] and y3 == [0]:
192     x1 = [3,0]
193     y1 = [3,0]
194     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
195     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="-", color='red'
196     plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
197     plt.text(3, 3, 'True News')
198     x2 = [-3,0]
199     y2 = [-5,0]
200     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
201     plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
202     plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
203     plt.text(-3, -5, 'Sarcasm Talk')
204     x3 = [3,0]
205     y3 = [7,0]
206     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
207     plt.plot(x3, y3, label = "Stance_Neutral", color='green') # , **marker_style, **text_style) # linestyle="-", color='red'
208     plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
209     plt.text(3, 7, 'Stance_Neutral')
210
211 # y1 = [1]
212 # y2 = [1]
213 # y3 = [0]
214 # Mixed / all fake 1/sarcasm 1/all misinformation&disinformation 0
215 if y1 == [1] and y2 == [1] and y3 == [0]:
216     x1 = [-3,0]
217     y1 = [-3,0]
218     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
219     plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
220     plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
221     plt.text(-3, -3, 'Fake News')
222     x2 = [-3,0]

```

```

220 plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
221 plt.text(-3, -3, 'Fake News')
222 x2 = [-3,0]
223 y2 = [-5,0]
224 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
225 plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
226 plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
227 plt.text(-3, -5, 'Saracasm Talk')
228 x3 = [3,0]
229 y3 = [7,0]
230 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
231 plt.plot(x3, y3, label = "Stance_Neutral", color='green') # , **marker_style, **text_style) # linestyle="-", color='red'
232 plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
233 plt.text(3, 7, 'Stance_Neutral')
234
235 # y1 = [1]
236 # y2 = [1]
237 # y3 = [0]
238 # Mixed / all fake 1/sarcasm 1/all misinformation&disinformation 0
239 if y1 == [1] and y2 == [1] and y3 == [0]:
240     x1 = [-3,0]
241     y1 = [-3,0]
242     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
243     plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
244     plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
245     plt.text(-3, -3, 'Fake News')
246     x2 = [-3,0]
247     y2 = [-5,0]
248     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
249     plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
250     plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
251     plt.text(-3, -5, 'Saracasm Talk')
252     x3 = [3,0]
253     y3 = [7,0]
254     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
255     plt.plot(x3, y3, label = "Stance_Neurtral", color='green') # , **marker_style, **text_style) # linestyle="-", color='red'
256     plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
257     plt.text(3, 7, 'Stance_Neutral')
258

```

```

256 plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
257 plt.text(3, 7, 'Stance_Neutral')
258
259 # y1 = [1]
260 # y2 = [0]
261 # y3 = [0]
262 #passed testing
263 # Mixed / all fake 1/sarcasm 0/all misinformation&disinformation 0
264 if y1 == [1] and y2 == [0] and y3 == [0]:
265     x1 = [-3,0]
266     y1 = [-3,0]
267     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
268     plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
269     plt.plot(-3, -3, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
270     plt.text(-3, -3, 'Fake News')
271     x2 = [3,0]
272     y2 = [5,0]
273     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
274     plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style) # linestyle="-", color='red'
275     plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
276     plt.text(3, 5, 'Direct Talk')
277     x3 = [3,0]
278     y3 = [7,0]
279     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="-", color='red'
280     plt.plot(x3, y3, label = "Stance_Neutral", color='green') # , **marker_style, **text_style) # linestyle="-", color='red'
281     plt.plot(3, 7, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
282     plt.text(3, 7, 'Stance_Neutral')
283
284
285
286 plt.xlim(-4, 6)
287 plt.ylim(-8, 9)
288 plt.gca().set_xticklabels([])
289 plt.gca().set_yticklabels([])
290
291 # y1 = [0]
292 # y2 = [0]
293 # y3 = [1]
294 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
295 if y1 == [0] and y2 == [0] and y3 == [1]:

```

```

288 plt.gca().set_yticklabels([])
289
290 # y1 = [0]
291 # y2 = [0]
292 # y3 = [1]
293 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
294 if y1 == [0] and y2 == [0] and y3 == [1]:
295     x1 = [3,0]
296     y1 = [3,0]
297     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
298     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
299     plt.plot(3, 3, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
300     plt.text(3, 3, 'True News')
301     x2 = [3,0]
302     y2 = [5,0]
303     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
304     plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
305     plt.plot(3, 5, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
306     plt.text(3, 5, 'Direct Talk')
307     x3 = [-3,0]
308     y3 = [-7,0]
309     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
310     plt.plot(x3, y3, label = "Stance Misinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
311     plt.plot(-3, -7, marker="o", markersize=20, markeredgewidth="red", markerfacecolor="red")
312     plt.text(-3, -7, 'Stance Misinformation')
313
314
315 # *** NEW for 0,1,2 single feature for stance column ***
316 # y1 = [0]
317 # y2 = [0]
318 # y3 = [2]
319 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
320 if y1 == [0] and y2 == [0] and y3 == [2]:
321     x1 = [3,0]
322     y1 = [3,0]
323     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
324     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
325     plt.plot(3, 3, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
326     plt.text(3, 3, 'True News')
327     x2 = [3,0]
328     y2 = [5,0]
329     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
330     plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
331     plt.plot(3, 5, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
332     plt.text(3, 5, 'Direct Talk')
333

```

```

332 plt.plot(3, 5, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
333 plt.text(3, 5, 'Direct Talk')
334 x3 = [-3,0]
335 y3 = [-9,0]
336 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
337 plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
338 plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
339 plt.text(-3, -9, 'Stance Disinformation')
340
341 # y1 = [0]
342 # y2 = [1]
343 # y3 = [2]
344 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
345 if y1 == [0] and y2 == [1] and y3 == [2]:
346     x1 = [3,0]
347     y1 = [3,0]
348     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
349     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
350     plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
351     plt.text(3, 3, 'True News')
352     x2 = [-3,0]
353     y2 = [-5,0]
354     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
355     plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
356     plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
357     plt.text(-3, -5, 'Sarcasm Talk')
358     x3 = [-3,0]
359     y3 = [-9,0]
360     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
361     plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
362     plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
363     plt.text(-3, -9, 'Stance Disinformation')
364
365 # y1 = [1]
366 # y2 = [1]
367 # y3 = [2]
368 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
369 if y1 == [1] and y2 == [1] and y3 == [2]:
370     x1 = [-3,0]

```

```

370 x1 = [-3,0]
371 y1 = [-3,0]
372 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
373 plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
374 plt.plot(-3, -3, marker="o", markersize=20, markeredgewidth="red", markerfacecolor="red")
375 plt.text(-3, -3, 'Fake News')
376 x2 = [-3,0]
377 y2 = [-5,0]
378 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
379 plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
380 plt.plot(-3, -5, marker="o", markersize=20, markeredgewidth="red", markerfacecolor="red")
381 plt.text(-3, -5, 'Sarcasm Talk')
382 x3 = [-3,0]
383 y3 = [-9,0]
384 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
385 plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
386 plt.plot(-3, -9, marker="o", markersize=20, markeredgewidth="red", markerfacecolor="red")
387 plt.text(-3, -9, 'Stance Disinformation')
388
389 # y1 = [1]
390 # y2 = [0]
391 # y3 = [2]
392 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
393 if y1 == [1] and y2 == [0] and y3 == [2]:
394     x1 = [-3,0]
395     y1 = [-3,0]
396     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
397     plt.plot(x1, y1, label = "Fake News", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
398     plt.plot(-3, -3, marker="o", markersize=20, markeredgewidth="red", markerfacecolor="red")
399     plt.text(-3, -3, 'Fake News')
400     x2 = [3,0]
401     y2 = [5,0]
402     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
403     plt.plot(x2, y2, label = "Direct Talk", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
404     plt.plot(3, 5, marker="o", markersize=20, markeredgewidth="green", markerfacecolor="green")
405     plt.text(3, 5, 'Direct Talk')
406     x3 = [-3,0]

```



```

405 plt.text(3, 5, 'Direct Talk')
406 x3 = [-3,0]
407 y3 = [-9,0]
408 #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
409 plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
410 plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
411 plt.text(-3, -9, 'Stance Disinformation')
412
413 # y1 = [0]
414 # y2 = [1]
415 # y3 = [2]
416 # Mixed / all fake 0/sarcasm 0/all misinformation&disinformation 1
417 if y1 == [0] and y2 == [1] and y3 == [2]:
418     x1 = [3,0]
419     y1 = [3,0]
420     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
421     plt.plot(x1, y1, label = "True News", color='green') # , **marker_style, **text_style) # linestyle="--", color='red'
422     plt.plot(3, 3, marker="o", markersize=20, markeredgecolor="green", markerfacecolor="green")
423     plt.text(3, 3, 'True News')
424     x2 = [-3,0]
425     y2 = [-5,0]
426     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
427     plt.plot(x2, y2, label = "Sarcasm Talk", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
428     plt.plot(-3, -5, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
429     plt.text(-3, -5, 'Sarcasm Talk')
430     x3 = [-3,0]
431     y3 = [-9,0]
432     #plt.plot(x1, y1, label = "line 1", marker='<', markersize=20, color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
433     plt.plot(x3, y3, label = "Stance Disinformation", color='red') # , **marker_style, **text_style) # linestyle="--", color='red'
434     plt.plot(-3, -9, marker="o", markersize=20, markeredgecolor="red", markerfacecolor="red")
435     plt.text(-3, -9, 'Stance Disinformation')
436
437 plt.legend(loc='lower right')
438
439 from matplotlib import pyplot as plt
440 plt.savefig('/content/drive/My Drive/Colab_Notebooks/MSCAITOP/MSCAITOP_graphic_v2b.png', bbox_inches='tight')
441
442 plt.show()
443
444 # 11, = 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>