IDENTIFYING DISINFORMATION

AUTHORS: DAVID BURRUS, GINNA WOO, JIANCI ZHAI (ANGELA)

TABLE OF CONTENTS

WHAT IS DISINFORMATION

- Dissemination of information that is deliberately misleading (disinformation) or simply inaccurate (misinformation) has been recognized as a threat to society.

- Disinformation that elicits emotional response from users tend to receive more click-throughs and is therefore favored by search engines.

- It has become more and more difficult for individuals to determine which content is based on facts because of the unlimited sources of information available today.

- By providing a "disinformation score" for each search result, our "search result markup" browser extension will influence user click-through by navigating the users towards more reliable sources.

- We use COVID-related content as an example to elaborate how we can add the extra score to help users identify misinformation.

PROJECT OBJECTIVES (AS PROPOSED)

1. Screen scrape search results from user query, focusing on organic rather than sponsored or paid results.

2. Visit each URL in the search results to scrape relevant text data, including headings, body, out-links, and author.

3. Analyze the content returned with topic mining and sentiment analysis.

4. The URL provides direct context by indicating an affiliation, so incorporate judgments from "authorities" who provide reliable, transparent assessments of the trustworthiness of websites.

5. Display search engine results to the user with an additional icon indicating level of accuracy or disinformation, so that users can make a more informed judgement.

DESIGN & CONCEPT

The original concept involved developing an extension (plug-in) to modify search engine results by injecting our disinformation analysis directly onto the search result page (see below).
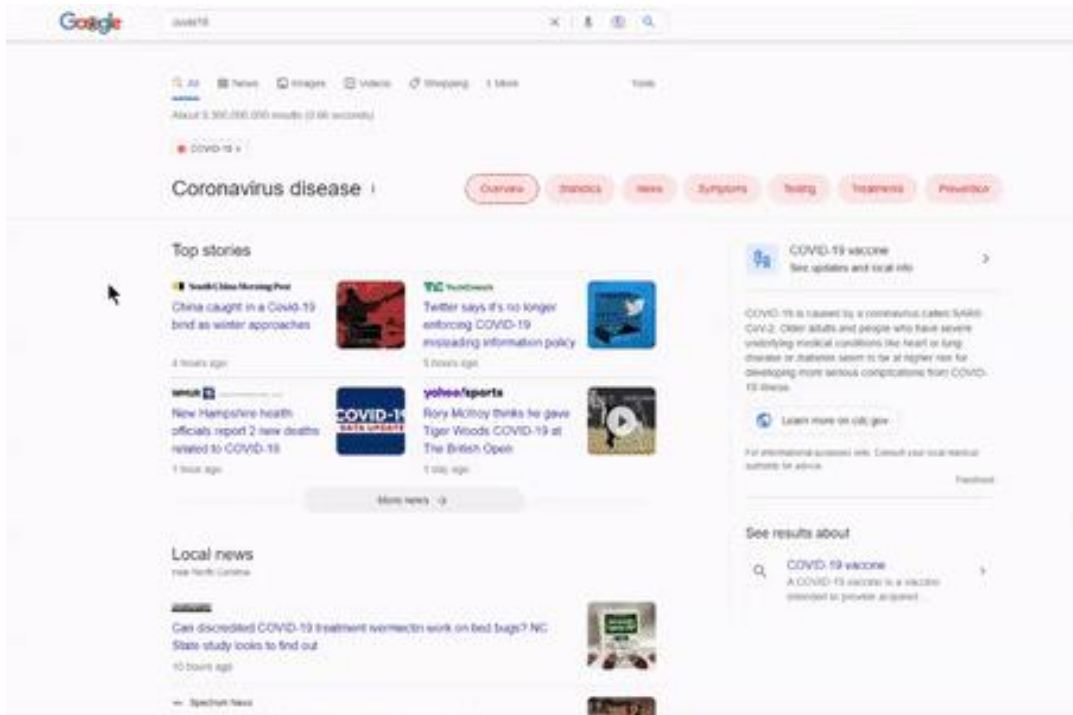

Figure 0.1 - Original concept involves projecting misinformation/disinformation scores.

However, due to limitations (issues with integrating Python and JavaScript codes) the team opted to create an interface to allow our users to input values for the program to run its analysis.
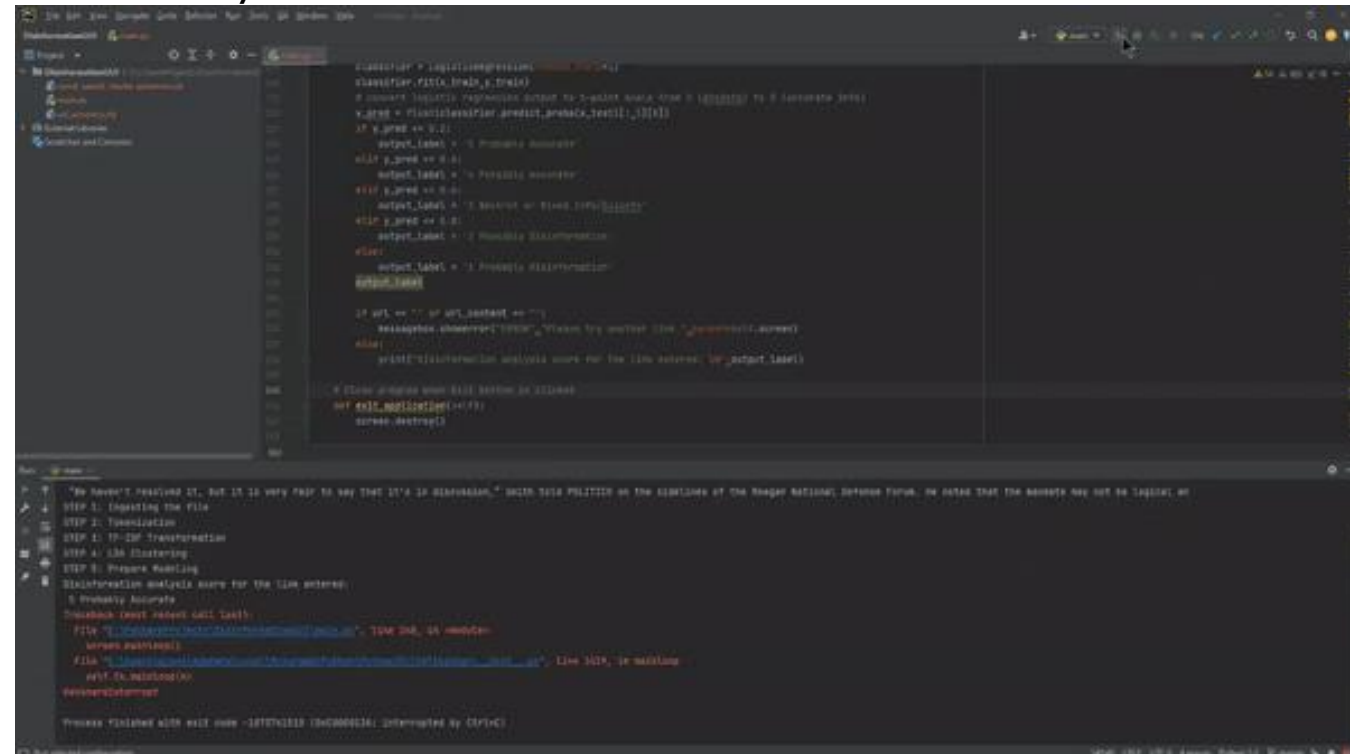

Figure 0.2 - Revised concept captures user inputs and projects analysis.

PROJECT SUBMISSION

Our project submission includes four files in total – two data files and two code files:

- **covid_search_results_summary.xls** : Excel file containing over 240 search engine results for various COVID-related queries. These data are used to train and test the analytical models.

- **url_accuracy.xls** : Excel file containing over 3,400 judgments of website factual accuracy, based on assessments from organizations like MediaBiasFactCheck, NewsGuard, HON, and others. These data are used by the models.

- **CS410_COVID_Notebook_A.ipynb** : Python notebook that scrapes results from three search engines, builds a model, and assesses its accuracy. This program requires the two data files above. Of the five project objectives, Notebook A achieves 1 through 4. However, it does not provide an interactive way for a user to enter his or her own choice of websites to analyze.

- **CS410_COVID_Notebook_B.ipynb** : Python notebook that provides an interactive assessment of user input. This program also requires the two data files above. Of the five project objectives, Notebook B achieves 2 through 4 and provides a way for users to enter a website for analysis. Finally, Notebook B uses a different modeling technique, so the reviewer can see the performance of different methods.

All files may be downloaded from the GitHub repository containing the project proposal and progress report.

NOTEBOOK A – OVERVIEW & IMPLEMENTATION

Notebook A has six sections:

1. Scrape search results – This code section demonstrates that is possible to scrape search engines such as Bing, Google, and Yahoo. However, this task proved much more complex than expected. We found that some search engines block attempts at scraping or crawling. Such blocks can be circumvented sometimes, but not always, by continuously changing USER_AGENT and other parameters.

2. Collect "search engine result" text data as well as context and judgment data – This section:

    a. Loads 'covid_search_results_summary.xls' is lieu of scraped search engine results. To minimize running time, the Excel file already contains website text and URL.

    b. Visits each URL in the "search results" to collect additional context such as heading text and out-links. (Attempts to collect author were not successful, because different website indicate author in different ways or not at all.) This operation can take two or three minutes. Websites that 'time out' or refuse the request are bypassed and an error is reported.

    c. Loads 'url_accuracy.xls' and incorporate the judgments of these authorities into the data set.

    d. Adds top-level domain (e.g., edu, gov, com, net) to the data set as a possible feature for the model. This is based on the supposition that certain TLDs such as edu and gov tend to be very reliable.

    e. Plots some exploratory data analysis. Please note that all program output is displayed within the Jupyter notebook.

3. Perform sentiment analysis using the NLTK package

4. Pre-process text data – This code cleans, tokenizes, and stems the text data in preparation for modeling, then plots word frequencies.

5. Process text data – This code builds features that will be used to train and test the model.

6. Build classifier – This final section creates a Support Vector Machine model, trains and tests it, then displays the model's accuracy, precision, recall, and F1 score.

NOTE BOOK A – USAGE

- Notebook A requires the following packages to be installed prior to use: tldextract, requests, bs4, urllib, requests_html, nltk, and sklearn as well as standard packages numpy, pandas, matplotlib, os, string, and collections.

- Notebook A can be run cell by cell. This allows the user to see the output of each section and to read code comments that are more detailed than the overview on the previous page.

- Alternatively, Notebook A can be run all at once by selecting "Run All" from the Jupyter "Cell" menu. The two most time-consuming operations are (a) collecting context data from every URL and (b) tokenizing the text. Both should finish in about three minutes or less, but users who do not wish to wait may select "Run All" then review the output at their leisure.

- Notebook A has been tested with Anaconda running Python 3.7 and higher.

NOTEBOOK B – OVERVIEW & IMPLEMENTATION

Notebook B has three parts:

- Part I – Implements an interactive user interface in lieu of the proposed browser extension. A full browser extension proved impossible to implement within the time constraints of a class project. When this cell is run, a dialog box will prompt the user to enter (a) a COVID-related web site address and (b) the text contents of that address. LDA clustering and Logistic Regression in this notebook then will try to determine if the web site is more likely to contain accurate information or disinformation. The program's output will be displayed below, in this notebook -- not in the dialog box.

- Part II – This code is substantially similar to Part I, except that (a) it is divided into many cells for the reviewer's convenience with detailed explanations and remarks and (b) the web address and contents to evaluate appear as two variables that the reviewer may fill in. This allows the reviewer to see the program's operation, step by step.

- Part III – This code, which requires Part II to have been run first, helped the team choose an optimal number of clusters/topics for modeling. Results from this analysis already have been fed back into part I as a constant (n_cnt).
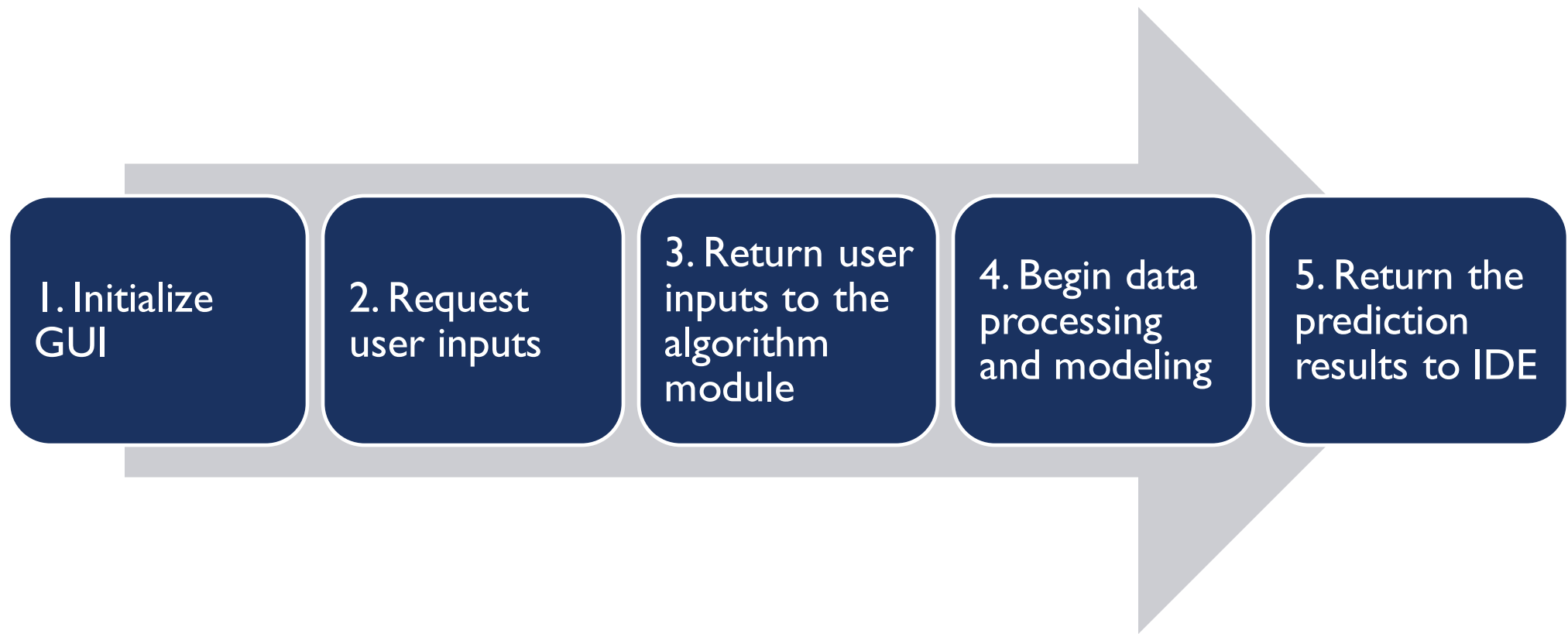
NOTEBOOK B – USAGE

- Notebook B requires the following packages to be installed prior to use: openpyxl, tkinter, tldextract, nltk, gensim, sklearn, and seaborn as well as standard packages numpy, pandas, string, matplotlib, pyplot, and plotly. Packages pyplot, plotly, and seaborn are not required by Part 1, because they are used to generate charts of the data and will help you understand the methodology better. Python 3.7 is strongly recommended.

- To run the GUI program (Part I), open the notebook in Jupyter and run the first code cell (immediately under the heading "Part I"). When this cell is run, a dialog box will appear with fields for the user to enter (a) a COVID-related web site address and (b) the text contents of that address.

- Example inputs (URL and text content) are on the next page, for the reviewer to try.

- Results of the program's analysis will be displayed in the Jupyter notebook (not the dialog box).

- Notebook B will work with PyCharm, Anaconda, or Visual Studio Code. If the notebook doesn't work, you can always try to convert it into a simpler .py file and run using python directly.

NOTEBOOK B – EXAMPLE INPUTS

URL	Text Content
https://www.who.int/westernpacific/emergencies/covid-19/information/asymptomatic-covid-19	Asymptomatic COVID-19 - A person infected with COVID-19 also may not experience any symptoms and, without knowing, can transmit the virus to others. Get vaccinated and keep up all the protective measures to protect yourself, your loved ones and your community. Quarantine is the separation of contacts from others after exposure to a probable or confirmed COVID-19 case – you may or may not be infected. Quarantine is important in helping to limit the spread of COVID-19. If you have had exposure to a probable or confirmed COVID-19 case, your national health authority may ask you to stay home or at a designated centre for a certain period of time to help break the chain of COVID-19 transmission. Self-monitor your health and watch for any coronavirus symptoms. Always follow the advice of your national health authority on when and how to seek medical attention.
https://www.bloomberg.com/news/articles/2021-08-27/previous-covid-prevents-delta-infection-better-than-pfizer-shot	People who recovered from a bout of Covid-19 during one of the earlier waves of the pandemic appear to have a lower risk of contracting the delta variant than those who got two doses of the vaccine from Pfizer Inc. and BioNTech SE. The largest real-world analysis comparing natural immunity -- gained from an earlier infection -- to the protection provided by one of the most potent vaccines currently in use showed that reinfections were much less common. The paper from researchers in Israel contrasts with earlier studies, which showed that immunizations offered better protection than an earlier infection, though those studies were not of the delta variant.
https://www.niaid.nih.gov/news-events/nih-clinical-trial-investigational-vaccine-covid-19-begins	A Phase 1 clinical trial evaluating an investigational vaccine designed to protect against coronavirus disease 2019 (COVID-19) has begun at Kaiser Permanente Washington Health Research Institute (KPWHRI) in Seattle. The National Institute of Allergy and Infectious Diseases (NIAID), part of the National Institutes of Health, is funding the trial. KPWHRI is part of NIAID's Infectious Diseases Clinical Research Consortium. The open-label trial will enroll 45 healthy adult volunteers ages 18 to 55 years over approximately 6 weeks. The first participant received the investigational vaccine today. The study is evaluating different doses of the experimental vaccine for safety and its ability to induce an immune response in participants. This is the first of multiple steps in the clinical trial process for evaluating the potential benefit of the vaccine. The vaccine is called mRNA-1273 and was developed by NIAID scientists and their collaborators at the biotechnology company Moderna, Inc., based in Cambridge, Massachusetts. The Coalition for Epidemic Preparedness Innovations (CEPI) supported the manufacturing of the vaccine candidate for the Phase 1 clinical trial. "Finding a safe and effective vaccine to prevent infection with SARS-CoV-2 is an urgent public health priority," said NIAID Director Anthony S. Fauci, M.D. "This Phase 1 study, launched in record speed, is an important first step toward achieving that goal."
https://weeksmd.com/2020/03/5g-worsens-coronavirus/	Did 5G's Neurotoxic and Immunotoxic Electromagnetic Radiation Make Wuhan's Coronavirus Epidemic Deadlier? A Discussion of Some of the Un-mentioned/Censored-out Co-Factors in the Coronavirus Pandemic. The unwanted truths about the connections between 1) our immune systems' resistance to infectious organisms and 2) the immunotoxic electromagnetic radiation of 5G networks have been intentionally kept out of the breathless 24/7 media coverage by our "Bought-and-Sold" Politicians, Medical "Experts", Public Health Bureaucrats and the Obedient Mainstream Media's Talking Heads – each of whom appear to be controlled by the ruling class's multibillionaire 1% that profits from every crisis, whether designed/man-made or accidental (except for their occasional irritating losses in stock market crashes (that the most of the savvy ones manage by short-selling maneuvers before the crashes occur). One could ask the logical question: Is what is happening in the world of mandatory over-vaccinating everybody who will stand still the Zombie Apocalypse or is it just another false flag Swine Flu, SARS, MERS or Zika "Pseudo-epidemic"?
https://twitter.com/CristianTerhes/status/1589262808084250624	The new COVID bivalent boosters aren't better than the old injections, a study finds. But these bivalent boosters were not tested in humans before they were approved in USA. So how really "safe and effective" are they? We need to demand answers to these questions. #TheyAllLied Doc Mc - fringe minority Replying to @CristianTerhes Thank you for exposing this. Canada is still in a coma where this is concerned. Our leaders are a big fail!
https://www.bitchute.com/video/pyczBzMfOnuo/	WARNING FOR HUMANITY / COVID-19 VACCINE & TRANSHUMANISM / BY DR CARRIE MADEJ – StopWorldControl Dr Carrie Madej explains how the proposed vaccine for COVID-19 can change humanity forever. Human 2.0, transhumanism, AI artificial intelligence.

NOTEBOOK B: DIVING INTO THE CODE

1. Initialize GUI

2. Request user inputs

3. Return user inputs to the algorithm module

4. Begin data processing and modeling
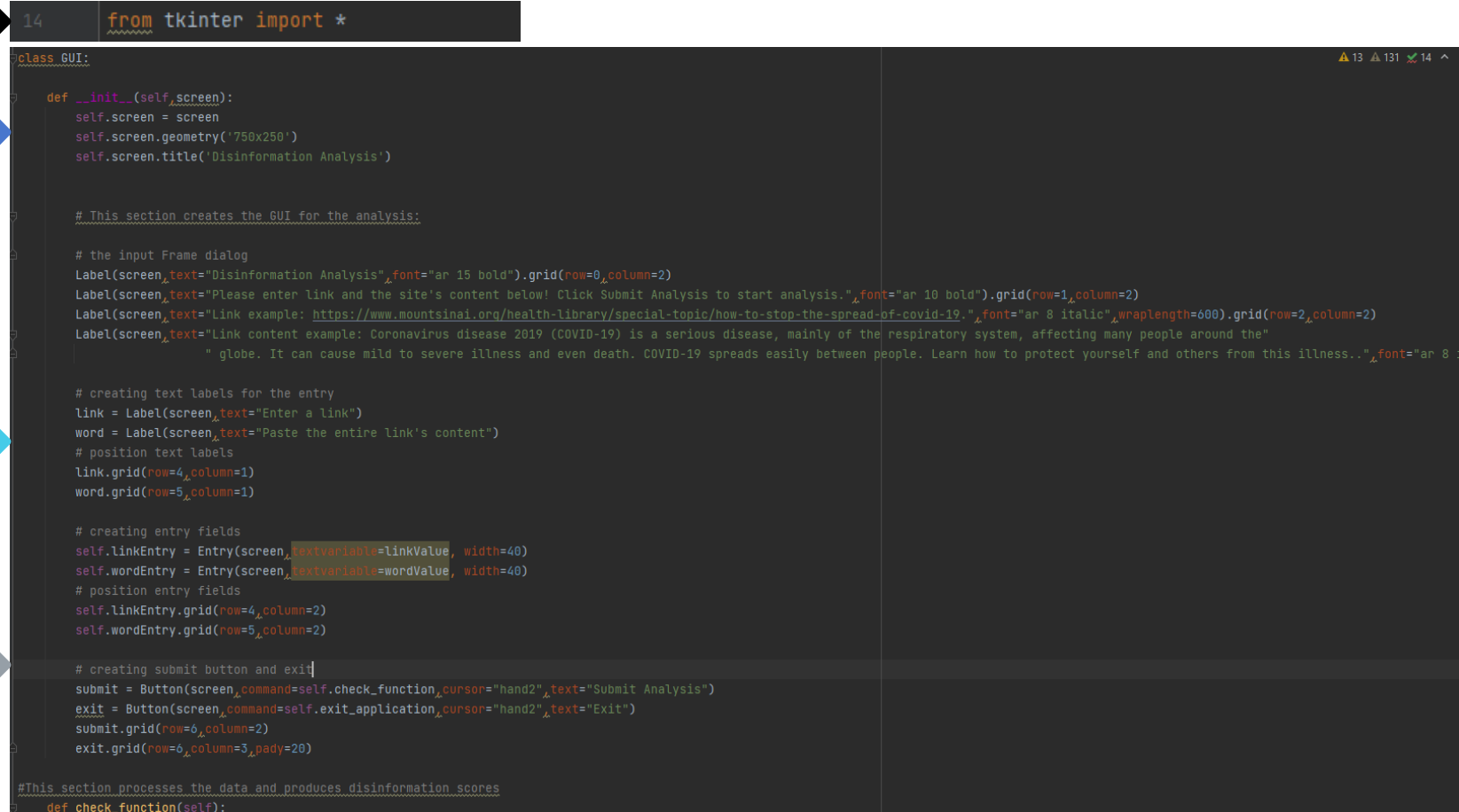
5. Return the prediction results to IDE

1. INITIALIZE USER INTERFACE

User interface requires Tkinter package.

The code begins with a function that creates the user interface.

The code initializes two variables to capture two of the main user inputs: link and the site's content.

Two button actions are created to allow the user to kick-off the program analysis.

```python
14    from tkinter import *

class GUI:

    def __init__(self,screen):
        self.screen = screen
        self.screen.geometry('750x250')
        self.screen.title('Disinformation Analysis')


        # This section creates the GUI for the analysis:

        # the input Frame dialog
        Label(screen,text="Disinformation Analysis",font="ar 15 bold").grid(row=0,column=2)
        Label(screen,text="Please enter link and the site's content below! Click Submit Analysis to start analysis.",font="ar 10 bold").grid(row=1,column=2)
        Label(screen,text="Link example: https://www.mountsinai.org/health-library/special-topic/how-to-stop-the-spread-of-covid-19.",font="ar 8 italic",wraplength=600).grid(row=2,column=2)
        Label(screen,text="Link content example: Coronavirus disease 2019 (COVID-19) is a serious disease, mainly of the respiratory system, affecting many people around the"
                           " globe. It can cause mild to severe illness and even death. COVID-19 spreads easily between people. Learn how to protect yourself and others from this illness..",font="ar 8 i

        # creating text labels for the entry
        link = Label(screen,text="Enter a link")
        word = Label(screen,text="Paste the entire link's content")
        # position text labels
        link.grid(row=4,column=1)
        word.grid(row=5,column=1)

        # creating entry fields
        self.linkEntry = Entry(screen,textvariable=linkValue, width=40)
        self.wordEntry = Entry(screen,textvariable=wordValue, width=40)
        # position entry fields
        self.linkEntry.grid(row=4,column=2)
        self.wordEntry.grid(row=5,column=2)

        # creating submit button and exit
        submit = Button(screen,command=self.check_function,cursor="hand2",text="Submit Analysis")
        exit = Button(screen,command=self.exit_application,cursor="hand2",text="Exit")
        submit.grid(row=6,column=2)
        exit.grid(row=6,column=3,pady=20)

#This section processes the data and produces disinformation scores
    def check_function(self):
```

Figure 1.0 - User interface code

2. REQUEST USER INPUTS

The interface requests user to provide two (2) inputs: *link* and the *site's content*. User can provide any link related to COVID19 research or articles and can simply copy and paste the site's content into the text box to evaluate the level of disinformation in the article. When the user clicks "Submit Analysis" the application will process the analysis on the backend from the IDE and project running processes. The application will then produce the final analysis results.
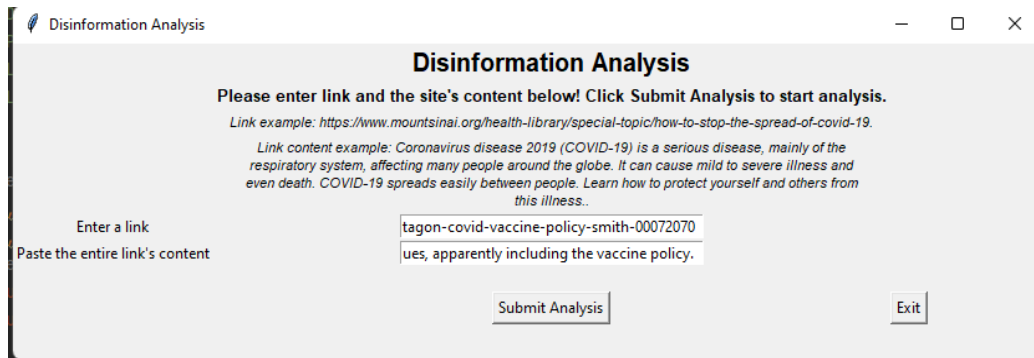

Figure 2.0 - Graphic User Interface for running Disinformation Analysis.

Figure 2.1 - Process status of the running analysis in IDE.

Note: The analysis may take some time to ingest the input and produce results - Please be patient.
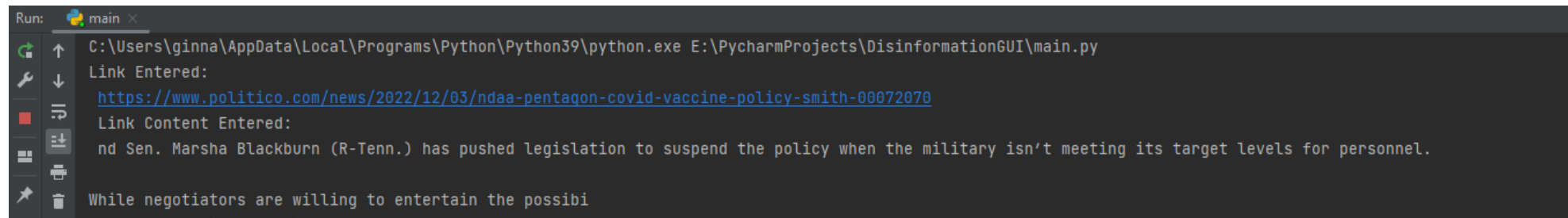
3. RETURN USER INPUTS TO THE ALGORITHM MODULE

Program will retrieve the user inputs from the textbox in the GUI and return the values in the IDE to the user before beginning the data processing stage.

```python
# This section processes the data and produces disinformation scores

def check_function(self):
    # grab user entry value from link box and word box
    url = self.linkEntry.get()
    url_content = self.wordEntry.get()
    print("Link Entered: \n", url)
    print(" Link Content Entered: \n", url_content[0:200])
```

Figure 3.0 - code that grabs user inputs and project out the values entered in the IDE (image below)

```
Run:    main

    C:\Users\ginna\AppData\Local\Programs\Python\Python39\python.exe E:\PycharmProjects\DisinformationGUI\main.py
    Link Entered:
        https://www.politico.com/news/2022/12/03/ndaa-pentagon-covid-vaccine-policy-smith-00072070
    Link Content Entered:
        nd Sen. Marsha Blackburn (R-Tenn.) has pushed legislation to suspend the policy when the military isn't meeting its target levels for personnel.

    While negotiators are willing to entertain the possibi
```

4. BEGIN DATA PROCESSING AND MODELING – CLEANING

In this stage, the model begins by processing a file called covid_search_results_summary.xls (training dataset) with about 250 samples of link and link's content. The user's input will be added the last of the list as the training dataset.

The model begins to stem and tokenize the content and transfer it into a matrix of doc * word. Then the model will normalize the value by tf-idf method and filter out words with very low tf-idf weighted values.

The word matrix then merge with url_accuracy.xls. The file carries the consolidated list of ranking of site accuracy from different websites that are specialized in evaluating the transparency and reliability of website.

Finally, we created the sentiment score for negative, positive, and neutral sentiment.

```python
import numpy as np
import pandas as pd

df = pd.read_excel('covid_search_results_summary.xls').dropna()

print('STEP1: Ingesting the file')

# org means original input
df.columns = df.columns.str.lower().str.replace(' ','_') + '_org'
# original row count of the file, and add new input at the very end as the testing data
org_len = len(df)
df.loc[org_len] = [org_len + 1, url_content, url, 0, len(url_content)]

import tldextract
df['link_clean'] = df['link_org'].apply(lambda x: tldextract.extract(x)[2])
df['host'] = df['link_org'].apply(lambda x: tldextract.extract(x)[1] + "." + tldextract.extract(x)[2])

url_ref = pd.read_excel('url_accuracy.xls').dropna()
df = pd.merge(df, url_ref, how='left')
df['accuracy_score'] = df['accuracy_score'].fillna(3)

def categorise(row):
    if row['accuracy_score'] == 1:
        return 1
    elif row['accuracy_score'] == 5:
        return 5
    else:
        return row['given_by_user1_org']
```

```python
import nltk
nltk.download('vader_lexicon', quiet=True)
from nltk.sentiment import SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()
```

```python
df['judgement_score'] = df.apply(lambda row: categorise(row), axis=1)
df['polarity_scores'] = df['content_org'].apply(sia.polarity_scores)
df['sentiment_neg'] = df['polarity_scores'].apply(lambda x: x['neg'])
df['sentiment_neu'] = df['polarity_scores'].apply(lambda x: x['neu'])
df['sentiment_pos'] = df['polarity_scores'].apply(lambda x: x['pos'])
df['sentiment_compound'] = df['polarity_scores'].apply(lambda x: x['compound'])
```

Figure 4.0 - Importing file to ingest test data to generate disinformation scoring after data cleaning along with generating sentiment analysis.

4. BEGIN DATA PROCESSING AND MODELING - TOKENIZATION

The code at a high level consists of tokenizing the data to remove punctuation, stop words, non-English and numbers. The analysis is carried out using Natural Language Toolkit (NLTK), which is a popular API (application programming interfaces) for Natural Language Processing (NLP) in Python. NLTK is used for stemming (reduces the number of unique words by analyzing just the stem of a word), and lemmatization (uses context to determine the meaning of a word). See lines 120.

The Python package genism.parsing.preprocessing will help remove stop words in our data to give focus to important words (see lines 112 above).

```python
10    from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
11    from sklearn.decomposition import LatentDirichletAllocation
12    from sklearn.model_selection import GridSearchCV
13    from sklearn.linear_model import LogisticRegression
14    from gensim.parsing.preprocessing import remove_stopwords
15    import string
16    from nltk.stem import PorterStemmer
17    from nltk.tokenize import word_tokenize
```

```python
110
111        # Remove stopwords, punctuation and numbers
112        nltk.download('punkt', quiet=True)
113        text1 = df['content_org']
114        text2 = [remove_stopwords(x) \
115                   .translate(str.maketrans('','',string.punctuation)) \
116                   .translate(str.maketrans('','',string.digits)) \
117                for x in text1]
118
119        # Stem and make lower case
120        def stemSentence(sentence):
121            porter = PorterStemmer()
122            token_words = word_tokenize(sentence)
123            stem_sentence = [porter.stem(word) for word in token_words]
124            return ' '.join(stem_sentence)
125
126        print('STEP2: Tokenization')
```

Figure 4.1 - Performing tokenization using NLTK and additional preprocessing packages from Python.

4. BEGIN DATA PROCESSING AND MODELING - TF-IDF TRANSFORMATION

We also remove non-English words from the results, and then perform a normalization of the datasets using TF-IDF methods.

To improve the computation efficiency, we removed words (see lines 155) that the overall normalized values prove to be too small to have a serious impact in results (value <= 25% percentile).

```
145    text3 = pd.Series([stemSentence(x) for x in text2])
146
147    # remove non-ascii characters and perform TF-IDF vectorization at the word level
148    vectorizer_wtf = TfidfVectorizer(analyzer='word', strip_accents="ascii")  # strip_accent: remove non-English
149    X_wtf = vectorizer_wtf.fit_transform(text3)
150    content_tfidf = pd.DataFrame(X_wtf.toarray())
151    content_tfidf.columns = vectorizer_wtf.get_feature_names_out()
152    # To improve the computation efficiency, remove words that the overall normalized values show are too small
153    # to have a serious impact on the results (value <= 25% percentile)
154    token_weight_normalized = content_tfidf.sum(axis=0, skipna=True)
155    low_bound = 0.02  # (around 25%)
156    token_weight_normalized = token_weight_normalized[token_weight_normalized >= low_bound]
157    content_tfidf_filtered = content_tfidf[token_weight_normalized.index]
158
159    print('STEP 3: TF-IDF Transformation')
```

Figure 4.1 - Generating normalized data using TF-IDF.

4. BEGIN DATA PROCESSING AND MODELING - LDA CLUSTERING

There are many ways to vectorize or combine features (normalized word weight). Based on our course, there is no absolute method that proves to be the best. We choose Latent Dirichlet Allocation (LDA) which is a generative statistical model that explains a set of observations through unobserved groups, to discover document topics. We choose to use LDA because:

1) we never use it before among the programming assignments; and

2) unlike k-means/tree-based algorithm, LDA is specifically used in natural language processing (NLP).

The function plot_top_words plots the topics obtained from the data set. Each topic is represented as a bar plot using top few words based on weights. *This function serves to provide our users a better understanding of our data only (see slide 20 for more details if interested) – please note that this function is not needed to run the program!*

```python
print('STEP3: TF-IDF TRANSFORMATION')

n_cnt = 11  # variable for topic count, created based on the results of algorithm optimization (Part III)
lda = LatentDirichletAllocation(n_components=n_cnt, learning_decay=0.9, random_state=35)
X_lda = lda.fit(content_tfidf_filtered)


# Commented out IPython magic to ensure Python compatibility.
from matplotlib import pyplot as plt


# Plot topics function.
# Code from: https://scikit-learn.org/stable/auto_examples/applications/plot_topics_extraction_with_nmf_lda.html
def plot_top_words(model, feature_names, n_top_words, title):
    fig, axes = plt.subplots(4, 3, figsize=(30, 30), sharex=True)
    axes = axes.flatten()
    for topic_idx, topic in enumerate(model.components_):
        top_features_ind = topic.argsort()[:-n_top_words - 1:-1]
        top_features = [feature_names[i] for i in top_features_ind]
        weights = topic[top_features_ind]

        ax = axes[topic_idx]
        ax.barh(top_features, weights, height=0.7)
        ax.set_title(f'Topic {topic_idx + 1}',
                     fontdict={'fontsize': 30})
        ax.invert_yaxis()
        ax.tick_params(axis='both', which='major', labelsize=20)
        for i in 'top right left'.split():
            ax.spines[i].set_visible(False)
        fig.suptitle(title, fontsize=40)
    plt.subplots_adjust(top=0.90, bottom=0.05, wspace=0.90, hspace=0.3)
    plt.show()
```

Figure 4.2 - Leveraging LDA to vectorize the data

TOPICS IDENTIFIED

A representation of the topics identified from the algorithm – COVID related terms are weighted much larger and is represented in the blue bar graph below (larger weight indicates higher chances that the topic is focused on COVID). Based on the charts, we found: Topic-2 is the most dominant one, which covers the general background and most popular topics around COVID (almost like a COVID specific background). Topic10 seem to related to disinformation that tried to falsely claim certain products can prevent covid. Topic 7 and 9 posts COVID related religious content. Finally, Topic 8 focuses on conspiracy theories around COVID. Topic 3 focuses on cruise related/TSA.
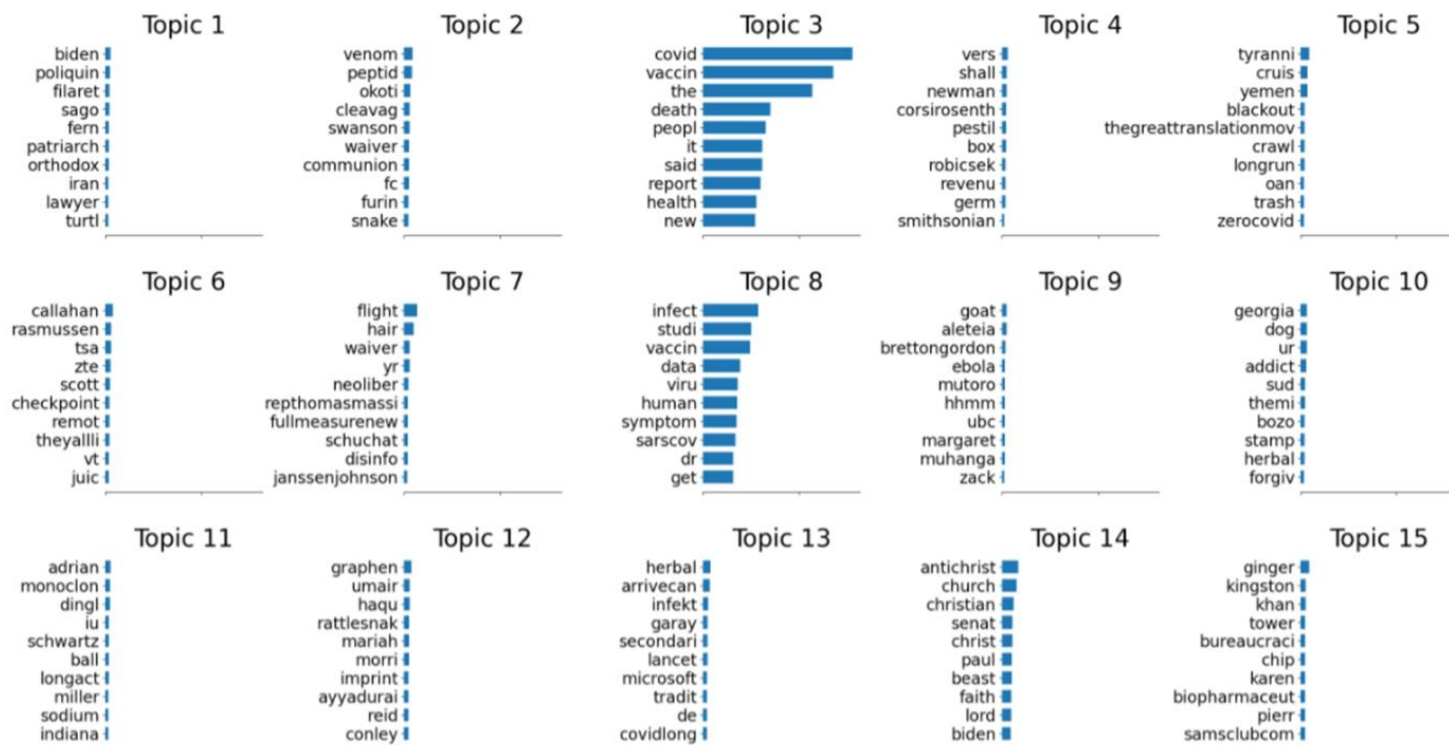
Different from the final chosen one in the model, but we want to leave it here so you can compare and see what topics are consistent in both versions.

4. BEGIN DATA PROCESSING AND MODELING - LDA CLUSTERING CONTINUE

In this section, we are creating the LDA scores based on document topics obtained from the previous step while normalizing the data using TF-IDF Transformation. We are combining the df data with the topic weights obtained previously. By doing so we are creating the weights of topics as our dependent variables.

Then, we converted the dependent variable into a dummy variable for the logistic regression.

The length of the content is much bigger than the other variables, so we use log-transformation and then divide the number by 10 to make sure it falls within a range comparable to other variables (mostly below 1).

```
174
175         feature_names = content_tfidf_filtered.columns
176
177         # create LDA score by document by topic
178         lda_score_by_topic = pd.DataFrame(X_lda.components_.dot(content_tfidf_filtered.T).T)
179
180         print('STEP4: LDA CLUSTERING')
181
182         lst_n_cnt = list(range(1,n_cnt + 1))
183         topic_n_cnt = ['topic_' + str(i) for i in lst_n_cnt]
184         lda_score_by_topic.columns = topic_n_cnt
185
186         df_normalized = pd.concat([df,content_tfidf_filtered,lda_score_by_topic],axis=1)
187         df_normalized['y_misinfo_tag'] = np.where(df_normalized['judgement_score'] <= 3,1,0)
188         df_normalized['len_log'] = np.log10(df_normalized['len_org']) / 10
```

Figure 4.3 - Generating LDA scores.

4. BEGIN DATA PROCESSING AND MODELING - PREPARE MODELING

In this final piece of the algorithm, we are using sklearn.linear_model for running the logistic regression to train the model.

The original data inputs from the user will serve as the training data: x_train. The y_train will serve as the outputs generated from the model. We incorporated a x_test variable as a testing input data for our model (generated from the user's input).

The y_pred stores the prediction generated from the model as a probability for our implementation to create our scoring for levels of disinformation.

The result produces an output_label variable.

```python
print('STEP 5: Prepare Modeling')
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

df_mdl_prep = df_normalized[['y_misinfo_tag',
                             'len_log',
                             'accuracy_score',
                             'sentiment_neg',
                             'sentiment_neu',
                             'sentiment_pos',
                             ] + topic_n_cnt]

# set up training and testing data sets
y_train = df_mdl_prep.iloc[:org_len,0].values
x_train = df_mdl_prep.iloc[:org_len,1:len(df_mdl_prep)].values
x_test = df_mdl_prep.iloc[org_len:org_len + 1,1:len(df_mdl_prep)].values
# fit the logistic regression classifier to the training data set
classifier = LogisticRegression(random_state=1)
classifier.fit(x_train,y_train)
# convert logistic regression output to 5-point scale from 1 (disinfo) to 5 (accurate info)
y_pred = float(classifier.predict_proba(x_test)[:,1][0])
if y_pred <= 0.2:
    output_label = '5 Probably Accurate'
elif y_pred <= 0.4:
    output_label = '4 Possibly Accurate'
elif y_pred <= 0.6:
    output_label = '3 Neutral or Mixed Info/Disinfo'
elif y_pred <= 0.8:
    output_label = '2 Possibly Disinformation'
else:
    output_label = '1 Probably Disinformation'
output_label
```

Figure 4.4 - Training the model using logistic regression and assigning scores based on the probabilities produced from the regression model.

5. RETURN THE PREDICTION RESULTS TO IDE

Finally, the prediction (the output_label variable) will be printed out on the IDE to show the user the disinformation score (i.e, 2 Probably Disinformation) ⟶
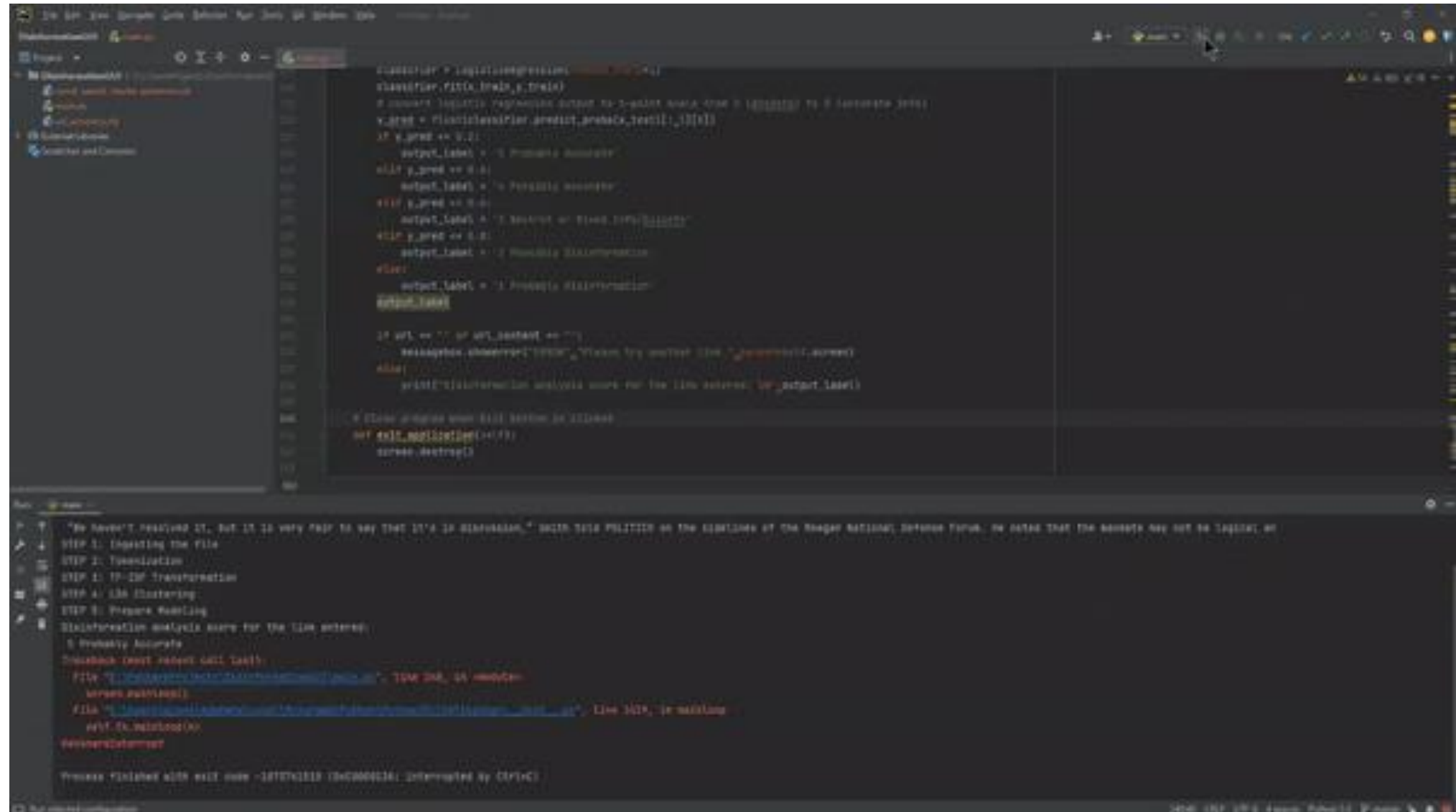

Figure 5.0 - print statement for the final analysis.

Figure 5.1 - outputs from the model in the IDE.

FINAL OUTCOME

CHALLENGES

TECHNICAL

- Search engines incorporate defensive measures to make scraping difficult.

- Google recently added countermeasures that remove most disinformation from search results related to COVID. This made it difficult to collect a balanced data set as originally planned for training.

- Search engines typically return approximately ten organic results per page. Collecting data from multiple search engine pages (e.g., 200 results from the first 20 search result pages) proved much more complex than expected

TIME

- The design of the websites are different, so it proves to be very hard to scrape text into organized forms, and impossible within the time available.

- As we had to manually collect training data, the dataset is still comparatively small for our purposes. With a much bigger training dataset, our algorithm should be able to provide more accurate results.

SKILLSETS

- Browser extensions are written in JavaScript while the algorithm is written in Python code. This created difficulties integrating the extension and Python codes to process the data.

- We can't normalize the testing text content independently because it will rely on the whole corpus, so for each iteration of prediction, we had to ingest and process all testing dataset, which will decrease the efficiencies of our program.

WORKAROUND FOR THE CHALLENGES

Web Scraping

- We ask users to manually input the URL and content of search result
- We manually collect a set of search results to provide diverse scores and topics

User Interface

- We use the most popular UI package for Python, tkinter, to eliminate the problem to connect different languages (e.g.: Python and Java)

Modeling

- We overwrite the judgment from one user with judgments provided by "authorities" websites when the source is identified as highly trustworthy or confirmed source of disinformation
- We adjust our testing datasets to teach the algorithm some additional features – for example, the resource of the content is more useful than the length of the content

TEAM CONTRIBUTION

Overall, we've evenly contributed to this project. We are glad everyone stretched their muscle and really applied what we learned into this project. Yet, everyone had a slightly different focus:

David: remember everything Professor Zhai said, and every word written on the book. Diligently keep track of what we did or did not. Created the amazing dictionary to tag the accuracy of website (3500+ sources to be checked guys!) Though epically fell before Google's security stonewall blocking his crawler, he has more reason to be proud rather sad.

Ginna: had created the initial plug-in in Java, and then heroically crashed a course to learn how to use tinker within a day to re-write everything. The logistic-master to keep our files in check and organized. She also shot the amazing demo and design the template of our presentation. Last but not least, Ginna is the glue of the team and a great cheerleader.

Angela: thanks to her background as data scientist, she was able to help the project on track and drove it to completion. She has manually created the testing dataset after our initial scrawler didn't work out. She also helped to write most of the analytics module, and figure out how to incorporate everyone's work together.

~~~~~~~~~~~~~~~~   **Remember: it's always the journey, not the destination**   ~~~~~~~~~~~~~~~~

THANK YOU

PLEASE REACH OUT TO OUR TEAM FOR FURTHER INQUIRIES.

EMAILS:

- DAVID BURRIS - (DBURRUS3@ILLINOIS.EDU)

- JIANCI (ANGELA) ZHAI - (JIANCIZ2@ILLINOIS.EDU)

- GINNA WOO - (GINNAW2@ILLINOIS.EDU)