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

```
Mounted at /content/drive
```

```
import pandas as pd

# Load text data into a pandas DataFrame
data = pd.read_csv('/content/drive/MyDrive/UROP/0819_UkraineCombinedTweetsDeduped.csv')
```

```
print('Count of columns in the dataset is:  ', len(data.columns))
print('Count of rows in the dataset is:  ', len(data))
```

```
Count of columns in the dataset is:   29
Count of rows in the dataset is:   47994
```

```
data.columns
```

```
Index(['Unnamed: 0', 'userid', 'username', 'acctdesc', 'location', 'following',
       'followers', 'totaltweets', 'usercreatedts', 'tweetid',
       'tweetcreatedts', 'retweetcount', 'text', 'hashtags', 'language',
       'coordinates', 'favorite_count', 'is_retweet', 'original_tweet_id',
       'original_tweet_userid', 'original_tweet_username',
       'in_reply_to_status_id', 'in_reply_to_user_id',
       'in_reply_to_screen_name', 'is_quote_status', 'quoted_status_id',
       'quoted_status_userid', 'quoted_status_username', 'extractedts'],
      dtype='object')
```

```
data[["text"]].head()
```

	text
0	Dear vaccine advocate\n\nDo take the COVID19 m...
1	#Mundo \n\nAl menos 6 muertos y 16 heridos en ...
2	Animal shelter Dogs and Cats, we need your hel...
3	Welcome to our shelter!\nLocated in Ukraine, K...
4	Tensión, debido a que #Rusia sigue en pie en l...

```
import pandas as pd

# Load text data into a pandas DataFrame
data = pd.read_csv('/content/drive/MyDrive/UROP/0819_UkraineCombinedTweetsDeduped.csv')

# Filter out rows that contain unwanted data
unwanted_data = ['WTI:', 'Brent:', 'EST', '#putin', '#petroleumengineering', '#bakerhughes']
filtered_data = data[~data['text'].str.contains('|'.join(unwanted_data))]

# Select only the 'username' and 'text' columns
new_data = filtered_data[['username', 'text']]
```

```
# Rename the 'text' column to 'tweet'
new_data.rename(columns={'text': 'tweet'}, inplace=True)

# Add an index of serial number to the new data
new_data.insert(0, 'serial_no', range(1, 1 + len(new_data)))

# Save the new DataFrame to a new CSV file
new_data.to_csv('/content/drive/MyDrive/Group_4_UROP/UserNameTweet_stage_1.csv', index=False)
```

```
<ipython-input-31-98bf7b0cf21b>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_data.rename(columns={'text': 'tweet'}, inplace=True)
```

```
import pandas as pd

# Load text data into a pandas DataFrame
df = pd.read_csv('/content/drive/MyDrive/Group_4_UROP/UserNameTweet_stage_1.csv',
                 lineterminator='\n')

df.head()
```

	serial_no	username	tweet
0	1	JoeMokolobetsi	Dear vaccine advocate\n\nDo take the COVID19 m...
1	2	XclusivasPuebla	#Mundo \n\nAl menos 6 muertos y 16 heridos en ...
2	3	ShelterAnimalUA	Animal shelter Dogs and Cats, we need your hel...
3	4	DogandCatHelpe1	Welcome to our shelter!\nLocated in Ukraine, K...
4	5	ElMananaOnline	Tensión, debido a que #Rusia sigue en pie en l...

```
print('Count of columns in the dataset is:  ', len(df.columns))
print('Count of rows in the dataset is:  ', len(df))
```

```
Count of columns in the dataset is:   3
Count of rows in the dataset is:   47463
```

```
df.columns
```

```
Index(['serial_no', 'username', 'tweet'], dtype='object')
```

```
df[["tweet"]].head()
```

	tweet
0	Dear vaccine advocate\n\nDo take the COVID19 m...

```
!pip install swifter
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting swifter
  Downloading swifter-1.3.4.tar.gz (830 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 830.9/830.9 kB 14.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (1.5.3)
Requirement already satisfied: psutil>=5.6.6 in /usr/local/lib/python3.10/dist-packages (from swifter) (5.9.5)
Requirement already satisfied: dask[dataframe]>=2.10.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (2022.12.1)
Requirement already satisfied: tqdm>=4.33.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (4.65.0)
Requirement already satisfied: ipywidgets>=7.0.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (7.7.1)
Requirement already satisfied: cloudpickle>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from swifter) (2.2.1)
Requirement already satisfied: parso>0.4.0 in /usr/local/lib/python3.10/dist-packages (from swifter) (0.8.3)
Requirement already satisfied: bleach>=3.1.1 in /usr/local/lib/python3.10/dist-packages (from swifter) (6.0.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from bleach>=3.1.1->swifter) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach>=3.1.1->swifter) (0.5.1)
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swifter) (8.1.3)
Requirement already satisfied: fsspec>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swifter) (2023.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swifter) (23.1)
Requirement already satisfied: partd>=0.3.10 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swifter) (1.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swifter) (6.0)
Requirement already satisfied: toolz>=0.8.2 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swifter) (0.12.0)
Requirement already satisfied: numpy>=1.18 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]>=2.10.0->swifter) (1.22.4)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.10/dist-packages (from ipywidgets>=7.0.0->swifter) (5.5.6)
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets>=7.0.0->swifter) (0.2.0)
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.10/dist-packages (from ipywidgets>=7.0.0->swifter) (5.7.1)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets>=7.0.0->swifter) (3.6.4)
Requirement already satisfied: ipython>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets>=7.0.0->swifter) (7.34.0)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets>=7.0.0->swifter) (3.0.7)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->swifter) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->swifter) (2022.7.1)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets>=7.0.0->swifter) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets>=7.0.0->swifter) (6.3.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (67.7.2)
Collecting jedi>=0.16 (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter)
  Downloading jedi-0.18.2-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 68.3 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (3.0
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (2.14.0)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (4.8.0)
Requirement already satisfied: locket in /usr/local/lib/python3.10/dist-packages (from partd>=0.3.10->dask[dataframe]>=2.10.0->swifter) (1.0.0)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.10/dist-packages (from widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (6.4.8)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (3.1.2)
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (23.2.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (21.3.0)
Requirement already satisfied: jupyter-core>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (5.8.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (6.5.4)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (1.
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (1.
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (0.1
```

```
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets>=7.0.0->swifter) (0.
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython>=4.0.0->ipywidgets>=7.0.0->swifter) (0.7.0)
```

```
!pip install langdetect
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting langdetect
  Downloading langdetect-1.0.9.tar.gz (981 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 981.5/981.5 kB 13.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from langdetect) (1.16.0)
Building wheels for collected packages: langdetect
  Building wheel for langdetect (setup.py) ... done
  Created wheel for langdetect: filename=langdetect-1.0.9-py3-none-any.whl size=993224 sha256=fba2432652a527ad019fd3b72902259c05943014116ef546cc9047ae7d12d164
  Stored in directory: /root/.cache/pip/wheels/95/03/7d/59ea870c70ce4e5a370638b5462a7711ab78fba2f655d05106
Successfully built langdetect
Installing collected packages: langdetect
Successfully installed langdetect-1.0.9
```

```
import pandas as pd
import swifter
import string
import re
from langdetect import detect
from langdetect.lang_detect_exception import LangDetectException

# Load the data into a pandas dataframe
df = pd.read_csv('/content/drive/MyDrive/Group_4_UROP/UserNameTweet_stage_1.csv', lineterminator='\n')

# Define contractions dictionary
contractions = {
    "ain't": "am not / are not / is not / has not / have not",
    "aren't": "are not / am not",
    "can't": "cannot",
    "can't've": "cannot have",
    "'cause": "because",
    "could've": "could have",
    "couldn't": "could not",
    "couldn't've": "could not have",
    "didn't": "did not",
    "doesn't": "does not",
    "don't": "do not",
    "hadn't": "had not",
    "hadn't've": "had not have",
    "hasn't": "has not",
    "haven't": "have not",
    "he'd": "he had / he would",
    "he'd've": "he would have",
    "he'll": "he shall / he will",
    "he'll've": "he shall have / he will have",
    "he's": "he has / he is",
    "how'd": "how did",
    "how'd'y": "how do you",
    "how'll": "how will",
    "how's": "how has / how is / how does",
```

```
    "I'd": "I had / I would",
    "I'd've": "I would have",
    "I'll": "I shall / I will",
    "I'll've": "I shall have / I will have",
    "I'm": "I am",
    "I've": "I have",
    "isn't": "is not",
    "it'd": "it had / it would",
    "it'd've": "it would have",
    "it'll": "it shall / it will",
    "it'll've": "it shall have / it will have",
    "it's": "it has / it is",
    "let's": "let us",
    "ma'am": "madam",
    "mayn't": "may not",
    "might've": "might have",
    "mightn't": "might not",
    "mightn't've": "might not have",
    "must've": "must have",
    "mustn't": "must not",
    "mustn't've": "must not have",
    "needn't": "need not",
    "needn't've": "need not have",
    "o'clock": "of the clock",
    "oughtn't": "ought not",
    "oughtn't've": "ought not have",
    "shan't": "shall not",
    "sha'n't": "shall not",
    "shan't've": "shall not have",
    "she'd": "she had / she would",
    "she'd've": "she would have",
    "she'll": "she shall / she will",
    "she'll've": "she shall have / she will have",
    "she's": "she has / she is",
    "should've": "should have",
    "shouldn't": "should not",
    "shouldn't've": "should not have",
    "shouldn't've": "should not have",
    "should've": "should have",
    "there's": "there is",
    "they'd": "they would",
    "they'll": "they will",
    "they're": "they are",
    "they've": "they have",
    "we'd": "we would",
    "we're": "we are",
    "we've": "we have",
    "weren't": "were not",
    "what'll": "what will",
    "what're": "what are",
    "what's": "what is",
    "what've": "what have",
    "where's": "where is",
    "who'd": "who would",
    "who'll": "who will",
    "who're": "who are",
```

```
        "who's": "who is",
        "who've": "who have",
        "won't": "will not",
        "wouldn't": "would not",
        "you'd": "you would",
        "you'll": "you will",
        "you're": "you are",
        "you've": "you have"
    }

    # Create regular expression pattern for finding contractions
    pattern = re.compile('({})'.format('|'.join(contractions.keys())), flags=re.IGNORECASE|re.DOTALL)

    # Define a function to remove non-textual data from a string
    def remove_non_textual(tweet):
        if not isinstance(tweet, str):
            tweet = str(tweet)
        # Remove URLs
        tweet = re.sub(r'http\S+', '', tweet)
        tweet = re.sub(r'www\S+', '', tweet)
        # Remove mentions
        tweet = re.sub(r'@\S+', '', tweet)
        # Remove hashtags
        tweet = re.sub(r'#\S+', '', tweet)
        # Remove special characters and numbers
        tweet = re.sub(r'[^\w\s]', '', tweet)
        tweet = re.sub(r'\d+', '', tweet)
        # Remove numbers
        tweet = tweet.translate(str.maketrans('', '', string.digits))
        # Remove punctuation
        tweet = tweet.translate(str.maketrans('', '', string.punctuation))
        # Convert to lowercase
        tweet = tweet.lower()
        # Remove extra whitespace
        tweet = re.sub(r'\s+', ' ', tweet).strip()
        return tweet

    # Define a function to detect English tweets
    def is_english(tweet):
        # Check if text is at least 10 characters long and contains at least 5 alphabetic characters
        if len(tweet) >= 10 and sum(c.isalpha() for c in tweet) >= 5:
            try:
                # Detect language and return True if the tweet is in English
                if detect(tweet) == 'en':
                    return True
            except LangDetectException:
                pass
        return False

    #Define function to replace contraction with expanded form
    def replace(match):
        return contractions[match.group(0).lower()]

    # Define a function to expand contractions
    def expand_contractions(tweet):
        # Replace contractions in tweet
```

```
        expanded_tweet = pattern.sub(replace, tweet)
        return expanded_tweet

    # Define a function to remove non-ASCII characters
    def remove_non_ascii(tweet):
        return re.sub(r'[^\x00-\x7f]', r'', tweet)

    # Apply the cleaning functions in sequence
    df['tweet'] = df['tweet'].swifter.apply(remove_non_textual)
    df['tweet'] = df['tweet'].apply(expand_contractions)
    df['tweet'] = df['tweet'].apply(remove_non_ascii)

    # Remove rows with empty tweets
    df = df[df['tweet'] != '']

    # Drop rows with missing tweet values
    df.dropna(subset=['tweet'], inplace=True)

    # Apply the is_english function to filter out non-English tweets
    df = df[df['tweet'].apply(is_english)]

    # Reset the index of the "serial_no" column to have sequential order starting from 1
    df['serial_no'] = range(1, len(df) + 1)

    # Keep only the desired columns
    df = df[['serial_no', 'username', 'tweet']]

    # Save the processed data to a new CSV file
    df.to_csv('/content/drive/MyDrive/Group_4_UROP/Processed_data_stage_2.csv', index=True)
```

```
Pandas Apply: 100%                                                 47463/47463 [00:02<00:00, 21525.91it/s]
<ipython-input-48-78f99d484ce8>:165: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  df.dropna(subset=['tweet'], inplace=True)
<ipython-input-48-78f99d484ce8>:171: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  df['serial_no'] = range(1, len(df) + 1)
```

```
import pandas as pd
import numpy as np

# Read the dataset
df = pd.read_csv('/content/drive/MyDrive/Group_4_UROP/LIWC-22 Results - Processed_data_stage_2 - LIWC Analysis.csv')

# Define the linguistic features for OCEAN score calculation
linguistic_features = [
    'Tone', 'WPS', 'BigWords', 'pronoun', 'prep', 'auxverb', 'adverb', 'conj', 'negate', 'verb', 'adj',
    'Drives', 'affiliation', 'achieve', 'Cognition', 'cogproc', 'insight', 'cause', 'discrep', 'tentat',
    'certitude', 'differ', 'memory', 'Affect', 'tone_pos', 'tone_neg', 'emotion', 'emo_pos', 'emo_neg',
    'emo_anx', 'emo_anger', 'emo_sad', 'swear', 'Social', 'socbehav', 'prosocial', 'polite', 'conflict',
```

```
        'moral', 'comm', 'socrefs', 'family', 'friend', 'female', 'male', 'Culture', 'politic', 'ethnicity',
        'tech', 'Lifestyle', 'leisure', 'home', 'work', 'money', 'relig', 'Physical', 'health', 'illness',
        'wellness', 'mental', 'substances', 'sexual', 'food', 'death', 'need', 'want', 'acquire', 'lack',
        'fulfill', 'fatigue', 'reward', 'risk', 'curiosity', 'allure', 'Perception', 'attention', 'motion',
        'space', 'visual', 'auditory', 'feeling', 'time', 'focuspast', 'focuspresent', 'focusfuture',
        'Conversation', 'netspeak', 'assent', 'nonflu', 'filler'
    ]

    # Define the Big Five personality traits and their corresponding linguistic features
    traits = {
        'openness': ['Tone', 'WPS', 'BigWords', 'cogproc', 'insight', 'cause', 'discrep', 'tentat',
                     'certitude', 'differ', 'memory'],
        'conscientiousness': ['Analytic', 'Clout', 'Tone', 'WPS', 'BigWords', 'cogproc', 'cause',
                              'certitude', 'differ', 'work'],
        'extraversion': ['Clout', 'Tone', 'WPS', 'BigWords', 'pronoun', 'Social', 'affiliation',
                         'achieve', 'cogproc', 'insight', 'cause', 'discrep', 'tentat', 'certitude',
                         'differ', 'memory'],
        'agreeableness': ['Tone', 'WPS', 'BigWords', 'pronoun', 'Social', 'affiliation', 'cogproc',
                          'insight', 'cause', 'discrep', 'tentat', 'certitude', 'differ', 'memory',
                          'socbehav', 'prosocial', 'polite', 'conflict', 'comm', 'socrefs',
                          'family', 'friend', 'female', 'male'],
        'neuroticism': ['Tone', 'WPS', 'BigWords', 'pronoun', 'Affect', 'tone_pos', 'tone_neg',
                        'emotion', 'emo_pos', 'emo_neg', 'emo_anx', 'emo_anger', 'emo_sad',
                        'death', 'illness', 'mental']
    }

    # Calculate OCEAN scores for each individual
    ocean_scores = pd.DataFrame(columns=['username'] + list(traits.keys()), dtype=int)
    for i, row in df.iterrows():
        scores = {}
        for trait, trait_features in traits.items():
            trait_score = np.mean(row[trait_features])

            mapped_number = ((trait_score * 3) // 25 + trait_score % 5) % 6 + 3
            scaled_score = int(mapped_number)

            scores[trait] = scaled_score

        ocean_scores.loc[i] = [row['username']] + list(scores.values())


    # Save the OCEAN scores to a new dataset
    ocean_scores.to_csv('/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_Using_LIWC_Analysis.csv', index=False)


    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.svm import SVC
    from sklearn.metrics import accuracy_score

    # Load the data
    data = pd.read_csv('/content/drive/MyDrive/Group_4_UROP/training_dataset.csv')
    df = pd.read_csv('/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_Using_LIWC_Analysis.csv')
    X_train = data[['openness','conscientiousness','extraversion','agreeableness','neuroticism']]
    y_train=data['Personality']
    y_test = df[['openness','conscientiousness','extraversion','agreeableness','neuroticism']]
```

```
clf = SVC(kernel='poly')
clf.fit(X_train,y_train)

y_pred = clf.predict(y_test)

df['personality'] = y_pred
df.to_csv('/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv', index=False)


import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the dataset
data_path = '/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv'
data = pd.read_csv(data_path)

# Split the dataset into training and testing sets
features = ['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism']
target = 'personality'
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target], test_size=0.5, random_state=100)

# Initialize and train the Support Vector Classifier (SVC)
svc_clf = SVC(C=0.005)  # Adjust the C parameter here
svc_clf.fit(X_train, y_train)

# Make predictions on the testing data
svc_pred = svc_clf.predict(X_test)

# Evaluate the model
svc_accuracy = accuracy_score(y_test, svc_pred)
svc_precision = precision_score(y_test, svc_pred, average='weighted')
svc_recall = recall_score(y_test, svc_pred, average='weighted')
svc_f1 = f1_score(y_test, svc_pred, average='weighted')

# Print the evaluation metrics
print("Support Vector Classifier (SVC):")
print("Accuracy:", svc_accuracy)
print("Precision:", svc_precision)
print("Recall:", svc_recall)
print("F1-score:", svc_f1)
```

```
Support Vector Classifier (SVC):
Accuracy: 0.8599982154010886
Precision: 0.7976813919250675
Recall: 0.8599982154010886
F1-score: 0.8245770930580905
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predict
  _warn_prf(average, modifier, msg_start, len(result))
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
# Load the dataset
data_path = '/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv'
data = pd.read_csv(data_path)

# Split the dataset into training and testing sets
features = ['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism']
target = 'personality'
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target], test_size=0.5, random_state=50)

# Initialize and train the Random Forest Classifier with modified parameters
rf_clf = RandomForestClassifier(n_estimators=3, max_depth=3, random_state=50)
rf_clf.fit(X_train, y_train)

# Make predictions on the testing data
rf_pred = rf_clf.predict(X_test)

# Calculate evaluation metrics
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_precision = precision_score(y_test, rf_pred, average='weighted')
rf_recall = recall_score(y_test, rf_pred, average='weighted')
rf_f1 = f1_score(y_test, rf_pred, average='weighted')

print("\nRandom Forest Classifier:")
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("F1-score:", rf_f1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predict
  _warn_prf(average, modifier, msg_start, len(result))

Random Forest Classifier:
Accuracy: 0.8023556705630409
Precision: 0.7411248009911064
Recall: 0.8023556705630409
F1-score: 0.76735956962452
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the dataset
data_path = '/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv'
data = pd.read_csv(data_path)

# Split the dataset into training and testing sets
features = ['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism']
target = 'personality'
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target], test_size=0.5, random_state=72)

# Initialize and train the MLP Classifier with modified parameters
mlp_clf = MLPClassifier(hidden_layer_sizes=(10, 10), alpha=0.2, max_iter=10)
```

```
mlp_clf.fit(X_train, y_train)

# Make predictions on the testing data
mlp_pred = mlp_clf.predict(X_test)

mlp_accuracy = accuracy_score(y_test, mlp_pred)
mlp_precision = precision_score(y_test, mlp_pred, average='weighted')
mlp_recall = recall_score(y_test, mlp_pred, average='weighted')
mlp_f1 = f1_score(y_test, mlp_pred, average='weighted')

print("\nMLP Classifier:")
print("Accuracy:", mlp_accuracy)
print("Precision:", mlp_precision)
print("Recall:", mlp_recall)
print("F1-score:", mlp_f1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predict
  _warn_prf(average, modifier, msg_start, len(result))

MLP Classifier:
Accuracy: 0.8810564825555457
Precision: 0.8453993740263046
Recall: 0.8810564825555457
F1-score: 0.8512236079698261
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the dataset
data_path = '/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv'
data = pd.read_csv(data_path)

# Split the dataset into training and testing sets
features = ['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism']
target = 'personality'
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target], test_size=0.5, random_state=62)

# Define custom priors for each class
custom_priors = [0.1, 0.3, 0.2, 0.15, 0.25]

# Initialize and train the Naive Bayes Classifier with modified priors
nb_clf = GaussianNB(priors=custom_priors)
nb_clf.fit(X_train, y_train)

# Make predictions on the testing data
nb_pred = nb_clf.predict(X_test)

nb_accuracy = accuracy_score(y_test, nb_pred)
nb_precision = precision_score(y_test, nb_pred, average='weighted')
nb_recall = recall_score(y_test, nb_pred, average='weighted')
nb_f1 = f1_score(y_test, nb_pred, average='weighted')
```

```
print("\nNaive Bayes Classifier:")
print("Accuracy:", nb_accuracy)
print("Precision:", nb_precision)
print("Recall:", nb_recall)
print("F1-score:", nb_f1)
```

```
Naive Bayes Classifier:
Accuracy: 0.770589809940216
Precision: 0.8673348541225273
Recall: 0.770589809940216
F1-score: 0.7969629966805309
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
from keras.utils import to_categorical

# Load the dataset
data_path = '/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv'
data = pd.read_csv(data_path)

# Split the dataset into training and testing sets
features = ['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism']
target = 'personality'
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target], test_size=0.5, random_state=82)

# Encode the target labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
num_classes = len(label_encoder.classes_)

# Build and train the LSTM model
lstm_model = Sequential()
lstm_model.add(Dense(100, input_dim=len(features), activation='relu'))
lstm_model.add(Dense(num_classes, activation='softmax'))
lstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Define early stopping callback to monitor validation loss
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the LSTM model
history = lstm_model.fit(X_train, to_categorical(y_train_encoded),
                         validation_data=(X_test, to_categorical(y_test_encoded)),
                         epochs=5, batch_size=60, callbacks=[early_stopping])

# Get the evaluation metrics
lstm_loss = history.history['loss'][-1]
lstm_accuracy = history.history['accuracy'][-1]
val_loss = history.history['val_loss'][-1]
val_accuracy = history.history['val_accuracy'][-1]
```

```
print("\nLSTM Model:")
print("Loss:", lstm_loss)
print("Accuracy:", lstm_accuracy)
print("Validation Loss:", val_loss)
print("Validation Accuracy:", val_accuracy)
```

```
Epoch 1/5
187/187 [==============================] - 1s 3ms/step - loss: 0.6698 - accuracy: 0.7957 - val_loss: 0.4542 - val_accuracy: 0.8644
Epoch 2/5
187/187 [==============================] - 0s 2ms/step - loss: 0.3904 - accuracy: 0.8881 - val_loss: 0.3663 - val_accuracy: 0.8924
Epoch 3/5
187/187 [==============================] - 1s 3ms/step - loss: 0.3353 - accuracy: 0.9018 - val_loss: 0.3327 - val_accuracy: 0.9036
Epoch 4/5
187/187 [==============================] - 1s 3ms/step - loss: 0.3073 - accuracy: 0.9095 - val_loss: 0.3060 - val_accuracy: 0.9139
Epoch 5/5
187/187 [==============================] - 1s 3ms/step - loss: 0.2870 - accuracy: 0.9143 - val_loss: 0.2881 - val_accuracy: 0.9200

LSTM Model:
Loss: 0.2870079278945923
Accuracy: 0.9143316149711609
Validation Loss: 0.28813108801841736
Validation Accuracy: 0.9200499653816223
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Embedding, GRU, Dense
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping

# Load the dataset
data_path = '/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv'
data = pd.read_csv(data_path)

# Split the dataset into training and testing sets
features = ['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism']
target = 'personality'
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target], test_size=0.5, random_state=72)

# Encode the target labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
num_classes = len(label_encoder.classes_)

# Build and train the GRU model
gru_model = Sequential()
gru_model.add(Dense(100, input_dim=len(features), activation='relu'))
gru_model.add(Dense(num_classes, activation='softmax'))
gru_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Define early stopping callback to monitor validation loss
early_stopping = EarlyStopping(monitor='val_loss', patience=1, restore_best_weights=True)

# Train the GRU model
```

```
# Train the GRU model
history = gru_model.fit(X_train, to_categorical(y_train_encoded),
                        validation_data=(X_test, to_categorical(y_test_encoded)),
                        epochs=5, batch_size=50, callbacks=[early_stopping])

# Get the evaluation metrics
gru_loss = history.history['loss'][-1]
gru_accuracy = history.history['accuracy'][-1]
val_loss = history.history['val_loss'][-1]
val_accuracy = history.history['val_accuracy'][-1]

print("\nGRU Model:")
print("Loss:", gru_loss)
print("Accuracy:", gru_accuracy)
print("Validation Loss:", val_loss)
print("Validation Accuracy:", val_accuracy)
```

```
Epoch 1/5
225/225 [==============================] - 2s 4ms/step - loss: 0.6138 - accuracy: 0.8021 - val_loss: 0.4274 - val_accuracy: 0.8750
Epoch 2/5
225/225 [==============================] - 1s 3ms/step - loss: 0.3715 - accuracy: 0.8888 - val_loss: 0.3600 - val_accuracy: 0.8948
Epoch 3/5
225/225 [==============================] - 1s 3ms/step - loss: 0.3286 - accuracy: 0.9018 - val_loss: 0.3402 - val_accuracy: 0.8991
Epoch 4/5
225/225 [==============================] - 1s 3ms/step - loss: 0.3016 - accuracy: 0.9133 - val_loss: 0.3042 - val_accuracy: 0.9082
Epoch 5/5
225/225 [==============================] - 1s 3ms/step - loss: 0.2846 - accuracy: 0.9167 - val_loss: 0.2866 - val_accuracy: 0.9150

GRU Model:
Loss: 0.2846258878707886
Accuracy: 0.9167410135269165
Validation Loss: 0.2865751385688782
Validation Accuracy: 0.9149638414382935
```

```
!pip install keras_preprocessing
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting keras_preprocessing
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
     ---------------------------------------- 42.6/42.6 kB 2.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from keras_preprocessing) (1.22.4)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from keras_preprocessing) (1.16.0)
Installing collected packages: keras_preprocessing
Successfully installed keras_preprocessing-1.1.2
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dropout
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

data = pd.read_csv('/content/drive/MyDrive/Group_4_UROP/OCEAN_Scores_With_Personality_Traits.csv')
X = data[['openness', 'conscientiousness', 'extraversion', 'agreeableness', 'neuroticism']].values.astype(str)
```

```
y = data['personality'].values

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
num_classes = len(label_encoder.classes_)

# Convert X to a list of strings
X = [' '.join(x) for x in X]

tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X)
X = tokenizer.texts_to_sequences(X)
X = pad_sequences(X, maxlen=100)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=52)

y_train = to_categorical(y_train, num_classes=num_classes)

model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_length=100))
model.add(Conv1D(64, 5, activation='relu'))
model.add(MaxPooling1D(pool_size=4))
model.add(Bidirectional(LSTM(64)))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='RMSprop', metrics=['accuracy'])

model.fit(X_train, y_train, validation_data=(X_test, to_categorical(y_test, num_classes=num_classes)), epochs=3, batch_size=25)

y_pred = model.predict(X_test)
y_pred = [list(x).index(max(x)) for x in y_pred]
y_test = [list(x).index(max(x)) for x in to_categorical(y_test, num_classes=num_classes)]
accuracy1 = accuracy_score(y_test, y_pred)
accuracy1 = accuracy1 * 100
print("Accuracy: {:.2f}%".format(accuracy1))
```

```
Epoch 1/3
449/449 [==============================] - 25s 49ms/step - loss: 0.6377 - accuracy: 0.7673 - val_loss: 0.2520 - val_accuracy: 0.9109
Epoch 2/3
449/449 [==============================] - 18s 40ms/step - loss: 0.2650 - accuracy: 0.9150 - val_loss: 0.1519 - val_accuracy: 0.9501
Epoch 3/3
449/449 [==============================] - 19s 42ms/step - loss: 0.1948 - accuracy: 0.9396 - val_loss: 0.1783 - val_accuracy: 0.9495
351/351 [==============================] - 4s 9ms/step
Accuracy: 94.95%
```

✓ 4s completed at 2:16 AM