Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel$\rightarrow$Restart) and then **run all cells** (in the menubar, select Cell$\rightarrow$Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

```
NAME = "Anusha Shishir Paranjpe"
COLLABORATORS = ""
```

Student(s) Information

Please provide information about yourself.
Name1: Anusha Shishir Paranjpe
Name2 (optional) :
NetID 1: ap1993
NetID 2 (optional):
Group Project Declaration (please provide a brief description of your partners contribution to the project. Do not discuss with partner before writing this): **IMPORTANT** Your work will not be graded withour your initials below
I certify that this lab represents my own work and I have read the RU academic intergrity policies at
https://www.cs.rutgers.edu/academic-integrity/introduction
Initials: AP

Grader Notes

Your Grade:
Grader Initials:
Grader Comments (optional):

Important

This project is provided to you as a class project in CS439. **DO NOT** post this notebook in any public or commerical space. Doing so may result in severe penalties. Please discuss with instructor if you have questions.

CS 439 - Introduction to Data Science

Spring 2023

Midsemester Project : Twitter Data Analysis

Due Date : Sunday March 26th, 2023 by 11:59 PM

Completing this project

This is your mid-semester project. You can work on this project individually or as part of a group of 2.

1. Working with a partner

Only one partner (max two per group) is allowed. Each person in the group is expected to contribute equally to the project. The team needs to do work equivalent to 2 individuals. We will be strict in grading criteria for groups. The team also needs to work hard to find good answers for Part 1 and Part 7. A post questionaire may be given to assess each person contribution. Each person in the team MUST submit a copy of the project. You MUST complete this survey by 3/10 to work as a team CLICK HERE TO FILL THE SURVEY

2. Working by yourself

If you work by yourself, you have two parts optional. Part 1 and Part 7. You are welcome to try those Parts just to learn things. No extra credit.

Project Purpose

The goal of this mid-semester project is to work with Twitter API to analyze tweets from a person, and in this case, Former President Donald Trump. @RealDonaldTrump tweets provide a great opportunity to understand how online media can be used to communicate over the traditional media. In fact, social media post are so influential, now the traditional media spends considerable amount of time discussing social media posts. Tweets from people like Donald Trump and Elon Musk have become so consequential, they can move the stock market on short term and get network TV to debate and discuss hours and hours about what Trump or Musk meant.

We hope this project will be fun as we can analyze range of emotions, hope, controversy, vagueness that are part of Trump tweets. We are interested in seeing what conclusions you can draw from former US Presidents tweets.

- DISCLAIMER: This project is not designed with any bias in mind. Note that we can pick any person (Hillary Clinton or Donald Trump or Elon Musk) or anyone else to do the same analysis. We hope your analysis is objective, independent of any political bias you may have. As Data Scientists, it is our responsiblity to do independent analysis of the data we try to understand. You should follow data and interpret insights w/o any bias.

Grading of the Project

You can test your project with the files provided. We may test the correctness of your code using **different files**. As a result, we will not provide sample outputs for this project. You will need to determine if the output received is reasonable. We are not looking for 100% compatibility with any one data set.

Set up

Let us get all the libaries initialized as necessary

```
# Run this cell to set up your notebook
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import zipfile
import json

# Ensure that Pandas shows at least 280 characters in columns, so we can see full tweets
pd.set_option('max_colwidth', 280)

%matplotlib inline
plt.style.use('fivethirtyeight')
import seaborn as sns
sns.set()
sns.set_context("talk")
import re
```

Downloading Recent Tweets (group only)

It is important to download the most recent tweets (especially if you are working as a group). You cannot download the recent tweets by @realdonaldtrump as he was inactive for last two years. But you can download tweets from @elonmusk or @joebiden to see how things work. Those who are working by themselves are allowed to use the downloaded files in data folder w/o setting up access to any twitter API (which can sometime be bit complicated). Twitter provides the API Tweepy (http://www.tweepy.org/) that makes it easy to access twitter content that is publicly available. We will also provide example code as needed.

```
## Make sure you have set up tweepy if you are working locally.
# https://www.pythoncentral.io/introduction-to-tweepy-twitter-for-python/
# After set up, the following should run:
import tweepy
```

PART 1: Accessing Twitter API (optional for individuals)

In order to access Twitter API, you need to get keys by signing up as a Twitter developer. We will walk you through this process.

- if you are working by yourself on this project, you can skip PART 1, and complete the project using the data files provided in the data folder instead. We highly recommend that you do Part 1 as an individual (after completing the project with offline data). You will "learn" how to use Twitter API that might be useful for learning how to work with API's.

Task 1.1

Follow the instructions below to get your Twitter API keys. **Read the instructions completely before starting.**

1. Create a Twitter account. You can use an existing account if you have one; if you prefer to not do this assignment under your regular account, feel free to create a throw-away account.
2. Under account settings, add your phone number to the account.
3. Create a Twitter developer account by clicking the 'Apply' button on the top right of the page. Attach it to your Twitter account. You'll have to fill out a form describing what you want to do with the developer account. Explain that you are doing this for a class at Rutgers University and that you don't know exactly what you're building yet and just need the account to get started. These applications are approved by some sort of AI system, so it doesn't matter exactly what you write. Just don't enter a bunch of alweiofalwiuhflawiuehflawuihflaiwhfe type stuff or you might get rejected.
4. Once you're logged into your developer account, create an application for this assignment. You can call it whatever you want, and you can write any URL when it asks for a web site. You don't need to provide a callback URL.
5. On the page for that application, find your Consumer Key and Consumer Secret.
6. On the same page, create an Access Token. Record the resulting Access Token and Access Token Secret.
7. Edit the file keys.json and replace the placeholders with your keys.

WARNING (Please Read) !!!!

Protect your Twitter Keys

If someone has your authentication keys, they can access your Twitter account and post as you! So don't give them to anyone, and **don't write them down in this notebook**. The usual way to store sensitive information like this is to put it in a separate file and read it programmatically. That way, you can share the rest of your code without sharing your keys. That's why we're asking you to put your keys in `keys.json` for this assignment.

Avoid making too many API calls.

Twitter limits developers to a certain rate of requests for data. If you make too many requests in a short period of time, you'll have to wait awhile (around 15 minutes) before you can make more. So carefully follow the code examples you see and don't rerun cells without thinking. Instead, always save the data you've collected to a file. We've provided templates to help you do that.

Be careful about which functions you call!

This API can retweet tweets, follow and unfollow people, and modify your twitter settings. Be careful which functions you invoke! It is possible that you can accidentally re-tweet some tweets because you typed `retweet` instead of `retweet_count`.

Reading Keys.json

```
import json
key_file = 'keys.json'
# Loading your keys from keys.json (which you should have filled in question 1):
with open(key_file) as f:
    keys = json.load(f)
# if you print or view the contents of keys be sure to delete the cell!
```

Task 1.2 Testing Twitter Authentication

This following code should run w/o errors or warnings and display Rutgers University's twitter username

```
import tweepy
from tweepy import TweepyException
import logging

try:
    auth = tweepy.OAuthHandler(keys["consumer_key"], keys["consumer_secret"])
    redirect_url = auth.get_authorization_url()
    auth.set_access_token(keys["access_token"], keys["access_token_secret"])
    api = tweepy.API(auth)
    print("Rutgers username is:", api.get_user(screen_name="RutgersU").name)
except TweepyException as e:
    logging.warning("There was a Tweepy error. Double check your API keys and try again.")
    logging.warning(e)
```

Getting more information from RutgersU

Find the following information about RutgersU. Show code and use a print statement to print the output.

```
# What is RutgersU screen name?

# What is the location RutgersU?

# What is a description for RutgersU?

# How many follow RutgersU?

# When was RutgersU account created?

# Is RutgersU a verified account?
```

Task 1.3

Refactor and Extend Code

Re-factor the above twitter authentication code and extend the code into reusable snippets below.

```
def load_keys(path):
    """Loads your Twitter authentication keys from a file on disk.
```

```
    Args:
        path (str): The path to your key file.  The file should
          be in JSON format and look like this (but filled in):
            {
                "consumer_key": "<your Consumer Key here>",
                "consumer_secret":  "<your Consumer Secret here>",
                "access_token": "<your Access Token here>",
                "access_token_secret": "<your Access Token Secret here>"
            }

    Returns:
        dict: A dictionary mapping key names (like "consumer_key") to
          key values."""

    ### BEGIN ANSWER

    # your solution here

    ### END ANSWER
```

```
def download_recent_tweets_by_user(user_account_name, keys):
    """Downloads tweets by one Twitter user.

    Args:
        user_account_name (str): The name of the Twitter account
          whose tweets will be downloaded.
        keys (dict): A Python dictionary with Twitter authentication
          keys (strings), like this (but filled in):
            {
                "consumer_key": "<your Consumer Key here>",
                "consumer_secret":  "<your Consumer Secret here>",
                "access_token": "<your Access Token here>",
                "access_token_secret": "<your Access Token Secret here>"
            }

    Returns:
        list: A list of Dictonary objects, each representing one tweet."""
    import tweepy
```

```
    ### BEGIN ANSWER

    # your solution here

    ### END ANSWER
```

```
def load_tweets(path):
    """Loads tweets that have previously been saved.

    Calling load_tweets(path) after save_tweets(tweets, path)
    will produce the same list of tweets.

    Args:
        path (str): The place where the tweets were be saved.

    Returns:
        list: A list of Dictionary objects, each representing one tweet."""

    ### BEGIN ANSWER

    # your solution here

    ### END ANSWER
```

```
def get_tweets_with_cache(user_account_name, keys_path):
    """Get recent tweets from one user, loading from a disk cache if available.

    The first time you call this function, it will download tweets by
    a user.  Subsequent calls will not re-download the tweets; instead
    they'll load the tweets from a save file in your local filesystem.
    All this is done using the functions you defined in the previous cell.
    This has benefits and drawbacks that often appear when you cache data:

    +: Using this function will prevent extraneous usage of the Twitter API.
    +: You will get your data much faster after the first time it's called.
    -: If you really want to re-download the tweets (say, to get newer ones,
       or because you screwed up something in the previous cell and your
       tweets aren't what you wanted), you'll have to find the save file
       (which will look like <something>_recent_tweets.pkl) and delete it.
```

```
    Args:
        user_account_name (str): The Twitter handle of a user, without the @.
        keys_path (str): The path to a JSON keys file in your filesystem.
    """

    ### BEGIN ANSWER

    # your solution here

    ### END ANSWER
```

Task 1.4

If everything was implemented correctly you should be able to obtain roughly the last max number of tweets by @RutgersU. (This may take a few minutes)

```
# When you are done, run this cell to load latest @RutgersU 's tweets. This is to get the latest twe
rutgers_tweets = download_recent_tweets_by_user("RutgersU", key_file)
print("Number of tweets downloaded:", len(rutgers_tweets))
```

PART 2 - Working with Twitter Data (group/individual)

The json file in data folder contains some loaded tweets from @RutgersU and @realdonaldtrump. Run the folllowing code and read and understand and what it does. Groups must download the latest tweets from @RutgersU using tweepy (and call that). Individuals can use the given file.

```
from pathlib import Path
import json

ds_tweets_save_path = "data/RutgersU_recent_tweets.json"    # need to get this file

# Guarding against attempts to download the data multiple
# times:
if not Path(ds_tweets_save_path).is_file():
    # Getting as many recent tweets by @RutgersU as Twitter will let us have.
    # We use tweet_mode='extended' so that Twitter gives us full 280 character tweets.
    # This was a change introduced in September 2017.

    # The tweepy Cursor API actually returns "sophisticated" Status objects but we
```

```
    # will use the basic Python dictionaries stored in the _json field.
    example_tweets = [t._json for t in tweepy.Cursor(api.user_timeline, screen_name="RutgersU",
                                                     tweet_mode='extended').items()]

    # Saving the tweets to a json file on disk for future analysis
    with open(ds_tweets_save_path, "w") as f:
        json.dump(example_tweets, f)

# Re-loading the json file:
with open(ds_tweets_save_path, "r") as f:
    example_tweets = json.load(f)
```

If things ran as expected, you should be able to look at the first tweet by running the code below. It probabably does not make sense to view all tweets in a notebook, as size of the tweets can freeze your browser (always a good idea to press ctrl-S to save the latest, in case you have to restart Jupyter)

```
# Looking at one tweet object, which has type Status:
from pprint import pprint # ...to get a more easily-readable view.
pprint(example_tweets[0])
```

```
# print the first 50 tweets from the cached file
temp = [sub["full_text"] for sub in example_tweets[:50]]
print(temp)
```

Task 2.2

To be consistent we are going to use the same dataset no matter what you get from your twitter api. So from this point on, if you are working as a group or individually, be sure to use the data sets provided to you in the data folder. One of the files is 'TrumpTweets_1.json', the other one is 'TrumpTweets_2.json'. First load TrumpTweets_1.

```
def load_tweets(path):
    """Loads tweets that have previously been saved.

    Calling load_tweets(path) after save_tweets(tweets, path)
    will produce the same list of tweets.

    Args:
        path (str): The place where the tweets will be saved.
```

```
    Returns:
        list: A list of Dictionary objects, each representing one tweet."""

    with open(path, "rb") as f:
        import json
        return json.load(f)
```

```
dest_path = "data/TrumpTweets_1.json" # Enter path of 'TrumpTweets_1.json' here
trump_tweets = load_tweets(dest_path)
```

```
# print the first 10 Trump tweets
temp2 = [sub["text"] for sub in trump_tweets[:10]]
print(temp2)
```

Task 2.3

Find the number of the month of the oldest tweet.

```
# Find the number of the month of the oldest tweet (e.g. 1 for January)
import pandas as pd
trump_tweets = pd.DataFrame(trump_tweets)

### BEGIN ANSWER
def oldest_tweet(df):
    x = trump_tweets["created_at"]
    x = pd.to_datetime(x)
    mindate = min(x)
    monthOfOldestTweet = mindate.month


    # your solution here
    return monthOfOldestTweet
### END ANSWER

oldest_month = oldest_tweet(trump_tweets)
print("The number of the month of the oldest tweet is:")
print(oldest_month)
```

PART 3 Twitter Source Analysis (group/individual)

Task 3.1

Merge the two dataframes created from TrumpTweets_1 and TrumpTweets_2. Call this new dataframe all_tweets

```
### BEGIN ANSWER
dest_path2 = "data/TrumpTweets_2.json" # Enter path of 'TrumpTweets_2.json' here
trump_tweets2 = load_tweets(dest_path2)
trump_tweets2 = pd.DataFrame(trump_tweets2)

trump_tweets["id"] = trump_tweets["id"].astype(int)
trump_tweets2["id"] = trump_tweets2["id"].astype(int)

all_tweets = pd.merge(trump_tweets, trump_tweets2, how = "outer")

### END ANSWER
all_tweets
all_tweets.head()
```

Task 3.2

Construct a DataFrame called `df_trump` containing all the tweets stored in `all_tweets`.
Important: There may/will be some overlap so be sure to **eliminate duplicate tweets**. If you do not eliminate the duplicates properly, your results might not be compatible with the test solution. **Hint:** the `id` of a tweet is always unique.

The index of the dataframe should be the ID of each tweet (looks something like `907698529606541312`). It should have these columns:

- `time`: The time the tweet was created encoded as a datetime object. (Use `pd.to_datetime` to encode the timestamp.)
- `source`: The source device of the tweet.
- `text`: The text of the tweet.
- `retweet_count`: The retweet count of the tweet.
- `favorite_count`: The favorite count of the tweet.

Finally, **the resulting dataframe should be sorted by date/time.**

Warning: *Some tweets may store the text in the `text` field and other will use the `full_text` field.*

```
# merged dataframe sorted by date/time (earliet tweet first)

### BEGIN ANSWER
```

```
#eliminating duplicate tweets:
df_trump = all_tweets
df_trump = df_trump.drop_duplicates(subset='id', keep="first")
df_trump = df_trump[["id", "created_at", "source", "text","full_text", "retweet_count", "favorite_cc
df_trump['text'] = df_trump['text'].fillna(df_trump['full_text'])
df_trump = df_trump.drop(['full_text'], axis=1)
df_trump["time"] = pd.to_datetime(df_trump["created_at"])
df_trump = df_trump.drop(['created_at'], axis=1)
df_trump = df_trump[["id", "time", "source", "text", "retweet_count", "favorite_count"]]
df_trump = df_trump.set_index("id")

df_trump = df_trump.sort_values('time')
display(df_trump)



    # your solution here

### END ANSWER
```

In the following questions, we are going to find out the charateristics of Trump tweets and the devices used for the tweets.

First let's examine the source field:

```
df_trump['source'].unique()
```

Task 3.3

Remove the HTML tags from the source text field.

Hint: Use `df_trump['source'].str.replace` and your favorite regular expression.

```
import re
### BEGIN ANSWER
df_trump['source'] = df_trump['source'].str.replace(r'<.+?>', '', regex = True)
df_trump['source'].unique()
    # your solution here

### END ANSWER
```

Question. What is the most common device used for Trump tweets? Make a plot to find out the most common device types used

Sort the plot in decreasing order of the most common device type

```
### BEGIN ANSWER
sourcecount = df_trump['source'].value_counts()
sourcecount = pd.DataFrame(sourcecount)
x = sns.barplot(x=sourcecount.source, y=sourcecount.index, orient='h')
x.set(xlabel = "Count" ,ylabel = "Source",title = "Devices used for Trump Tweets")
sourcecount = sourcecount.rename(columns = {'source':'count'}, inplace = True)
sourcecount
    # your solution here

### END ANSWER
```

Task 3.4

Is there a difference between his Tweet behavior across these devices? We will attempt to answer this question in our subsequent analysis.

First, we'll take a look at whether Trump's tweets from an Android come at different times than his tweets from an iPhone. Note that Twitter gives us his tweets in the UTC timezone (notice the `+0000` in the first few tweets)

Note - If your `time` column is not in datetime format, the following code will not work.

```
df_trump['time'][0:3]
```

We'll convert the tweet times to US Eastern Time, the timezone of New York and Washington D.C., since those are the places we would expect the most tweet activity from Trump.

```
df_trump['est_time'] = (
    df_trump['time'] # Set initial timezone to UTC
                .dt.tz_convert("EST") # Convert to Eastern Time
)
df_trump.head()
```

What you need to do:

Add a column called `hour` to the `df_trump` table which contains the hour of the day as floating point number computed by:

$$
\text{hour} + \frac{\text{minute}}{60} + \frac{\text{second}}{60^2}
$$

```
df_trump['hour'] = ((df_trump['est_time'].dt.hour)+((df_trump['est_time'].dt.minute)/60)+((df_trump|
# a new column that contains the rounded hour
df_trump['roundhour']=round(df_trump['hour'])
df_trump
```

Use the `roundhour` column and plot the number of tweets at every hour of the day.
Order the plot using the hour of the day (1 to 24). Use seaborn `countplot`

```
# make a bar plot here
### BEGIN ANSWER
tempo = df_trump[["roundhour", "text"]]
tempo = tempo.sort_values(by = 'roundhour')
sns.countplot(data=tempo, x='roundhour')
    # your solution here

### END ANSWER
```

Now, use this data along with the seaborn `distplot` function to examine the distribution over hours of the day in eastern time that trump tweets on each device for the 2 most commonly used devices. Your plot should look somewhat similar to the following.
Drawing

```
### BEGIN ANSWER
temp2 = df_trump[["roundhour", "text", "source"]]
iPhone = temp2[temp2['source'] == 'Twitter for iPhone']
Android = temp2[temp2['source'] == 'Twitter for Android']

p = sns.distplot(iPhone.roundhour, rug = False, hist = False, color = "blue")
p = sns.distplot(Android.roundhour, rug = False, hist = False, color = "red")
p.set(xlim=(-5, 31), xlabel = "Hour", ylabel = "Fraction")
p.legend(labels=['iPhone','Android'])
    # your solution here
```

```
### END ANSWER
```

Task 3.5

According to this Verge article, Donald Trump switched from an Android to an iPhone sometime in March 2017.

Create a figure identical to your figure from 3.4, except that you should show the results only from 2016. If you get stuck consider looking at the `year_fraction` function from the next problem.

Use this data along with the seaborn `distplot` function to examine the distribution over hours of the day in eastern time that trump tweets on each device for the 2 most commonly used devices. Your plot should look somewhat similar to the following.

During the campaign, it was theorized that Donald Trump's tweets from Android were written by him personally, and the tweets from iPhone were from his staff. Does your figure give support the theory?

Your Response: Yes, the figure supports the theory as it was said that Trump mostly tweets in the morning and he switched to iphone in 2017. So, as we can see that Android tweets were sometime in the morning hence, it was Donald Trump tweeting, whereas, iPhone tweets were during afternoon hence it is belived that the tweets were from his staff. As the peaks of the tweets can be seen in the graph.

In 2016, the time allocation for the usage of the iphone centered in the afternoon, while his tweets from 2015 to present shows that he mostly tweets in the morning. It seems that the tweets from iphone in 2016 were from his staff, not himself.

\
 Drawing

```
### BEGIN ANSWER
Year2016 = df_trump[["roundhour", "est_time", "text", "source"]]
Year2016 = Year2016[Year2016["est_time"].dt.year == 2016]
iPh = Year2016[Year2016['source'] == 'Twitter for iPhone']
And = Year2016[Year2016['source'] == 'Twitter for Android']

p = sns.distplot(iPh.roundhour, rug = False, hist = False, color = "blue")
p = sns.distplot(And.roundhour, rug = False, hist = False, color = "red")
p.set(xlim=(-5, 31), xlabel = "Hour", ylabel = "Fraction")
p.legend(labels=['iPhone','Android'], loc = "upper left")
sns.set(rc={"figure.figsize":(8, 4)}) #width=8, height=4
```

```
    # your solution here

### END ANSWER
```

Task 3.6

Edit this cell to answer the following questions.

- What time of the day the Android tweets were made by Trump himself? (eg: morning, late night etc)

Answer: Based on the dataset, in the morning the tweets were made by Trump himself.

- What time of the day the Android tweets were made by paid staff?

Answer: Based on the dataset, in the evening and towards the night the tweets were made by the Staff.

Note that these are speculations based on what you observe in the data set.

Task 3.7 Device Analysis

Let's now look at which device he has used over the entire time period of this dataset.

To examine the distribution of dates we will convert the date to a fractional year that can be plotted as a distribution.

(Code borrowed from https://stackoverflow.com/questions/6451655/python-how-to-convert-datetime-dates-to-decimal-years)

```
import datetime
def year_fraction(date):
    start = datetime.date(date.year, 1, 1).toordinal()
    year_length = datetime.date(date.year+1, 1, 1).toordinal() - start
    return date.year + float(date.toordinal() - start) / year_length


df_trump['year'] = df_trump['time'].apply(year_fraction) #should be df_trump
```

Use the `sns.distplot` to overlay the distributions of the 2 most frequently used web technologies over the years. Your final plot should be similar to:

```
#plt.figure(figsize=(15,15))
### BEGIN ANSWER
plotting = df_trump[["year", "source", "text"]]
phone = plotting[plotting['source'] == 'Twitter for iPhone']
droid = plotting[plotting['source'] == 'Twitter for Android']

p = sns.distplot(phone.year, rug = False, hist = True)
p = sns.distplot(droid.year, rug = False, hist = True)
p.set(xlabel = "year", ylabel = "")
p.legend(labels=['iPhone','Android'], loc = "upper right")
sns.set(rc={'figure.figsize':(9,5)})

#p.figure(figsize=(15,15))

    # your solution here

### END ANSWER
```

PART 4 - Sentiment Analysis (group/individual)

It turns out that we can use the words in Trump's tweets to calculate a measure of the sentiment of the tweet. For example, the sentence "I love America!" has positive sentiment, whereas the sentence "I hate taxes!" has a negative sentiment. In addition, some words have stronger positive / negative sentiment than others: "I love America." is more positive than "I like America."

We will use the VADER (Valence Aware Dictionary and sEntiment Reasoner) lexicon to analyze the sentiment of Trump's tweets. VADER is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media which is great for our usage.

The VADER lexicon gives the sentiment of individual words. Run the following cell to show the first few rows of the lexicon:

```
print(''.join(open("data/vader_lexicon.txt").readlines()[:10]))
```

Task 4.1

As you can see, the lexicon contains emojis too! The first column of the lexicon is the *token*, or the word itself. The second column is the *polarity* of the word, or how positive / negative it is.

Question How did they decide the polarities of these words? What are the other two columns in the lexicon? (See the link above.)

Read in the lexicon into a DataFrame called `df_sent`. The index of the DF should be the tokens in the lexicon. `df_sent` should have one column: `polarity`: The polarity of each token.

```
### BEGIN ANSWER
#reading the txt into dataframe
df_sent = pd.read_csv("data/vader_lexicon.txt", sep = '\t', header = None)
#Naming the columns
df_sent.columns = ["token", "polarity", "none", "none2"]
#dropping the columns not req
df_sent = df_sent.drop(["none", "none2"], axis=1)
#setting the index to token
df_sent = df_sent.set_index('token')
# your solution here
df_sent
### END ANSWER
```

Task 4.2

Now, let's use this lexicon to calculate the overall sentiment for each of Trump's tweets. Here's the basic idea:

1. For each tweet, find the sentiment of each word.
2. Calculate the sentiment of each tweet by taking the sum of the sentiments of its words.

Be sure to lowercase the text in the tweets since the lexicon is also lowercase. Set the `text` column of the `df_trump` DF to be the lowercased text of each tweet.

```
### BEGIN ANSWER
#making the text lowercase:
df_trump['text'] = df_trump['text'].apply(str.lower)
sentiment = 0
listofsenti = []
#tweet = df_trump['text'].groupby(df_sent.index)
for tweet in df_trump['text']:
    words = tweet.split()
    for word in words:
        if word in df_sent.index:
            indi = df_sent.index.get_loc(word)
            sentiment = sentiment + df_sent["polarity"][indi]
    listofsenti.append(sentiment)
```

```
#print(listofsenti)
#df_trump
# your solution here
### END ANSWER
```

Task 4.3

Now, let's get rid of punctuation since it'll cause us to fail to match words. Create a new column called `no_punc` in the `df_trump` to be the lowercased text of each tweet with all punctuation replaced by a single space. We consider punctuation characters to be any character that isn't a Unicode word character or a whitespace character. You may want to consult the Python documentation on regexes for this problem.

Question Why don't we simply remove punctuation instead of replacing with a space? See if you can figure this out by looking at the tweet data.

```
# Save your regex in punct_re
punct_re = r'[^\w\s\\n]'
### BEGIN ANSWER
df_trump["no_punc"] = df_trump['text'].str.replace(punct_re, ' ', regex = True)

# your solution here
df_trump
### END ANSWER
```

```
assert isinstance(punct_re, str)
assert re.search(punct_re, 'this') is None
assert re.search(punct_re, 'this is ok') is None
assert re.search(punct_re, 'this is\nok') is None
assert re.search(punct_re, 'this is not ok.') is not None
assert re.search(punct_re, 'this#is#ok') is not None
assert re.search(punct_re, 'this^is ok') is not None
assert df_trump['no_punc'].loc[800329364986626048] == 'i watched parts of  nbcsnl saturday night liv
assert df_trump['text'].loc[884740553040175104] == 'working hard to get the olympics for the united
```

Task 4.4

Now, let's convert the tweets into what's called a *tidy format* to make the sentiments easier to calculate. Use the `no_punc` column of `df_trump` to create a table called `tidy_format`. The index of the table should be the IDs of the tweets, repeated once for every word in the tweet. It has two columns:

1. `num`: The location of the word in the tweet. For example, if the tweet was "i love america", then the location of the word "i" is 0, "love" is 1, and "america" is 2.
2. `word`: The individual words of each tweet.

The first few rows of our `tidy_format` table look like:

	num	word
894661651760377856	0	i
894661651760377856	1	think
894661651760377856	2	senator
894661651760377856	3	blumenthal
894661651760377856	4	should

You can double check that your tweet with ID `894661651760377856` has the same rows as ours. Our tests don't check whether your table looks exactly like ours.

As usual, try to avoid using any for loops. Our solution uses a chain of 5 methods on the 'trump' DF, albeit using some rather advanced Pandas hacking.

- **Hint 1:** Try looking at the `expand` argument to pandas' `str.split`.
- **Hint 2:** Try looking at the `stack()` method.
- **Hint 3:** Try looking at the `level` parameter of the `reset_index` method.

```
#tidy_format = ...

### BEGIN ANSWER
#creating a temporary df to clean:
temp = df_trump[["no_punc"]]
temp = temp['no_punc'].str.split(" ", expand = True)

temp = temp.stack(dropna = True)
temp = temp.reset_index(level = "id")
temp['num'] = temp.index
temp = temp.set_index("id")
temp = temp.rename(columns = {0 : "word"})
#transferring it to required df:
tidy_format = temp[["num", "word"]]
```

```
tidy_format = tidy_format.replace(r'^s*$', float('NaN'), regex = True)  # Replace blanks by NaN
tidy_format.dropna(inplace = True)                                      # Remove rows with NaN
# your solution here
tidy_format
### END ANSWER
```

```
assert tidy_format.loc[894661651760377856].shape == (27, 2)
assert ' '.join(list(tidy_format.loc[894661651760377856]['word'])) == 'i think senator blumenthal sh
```

Task 4.5

Now that we have this table in the tidy format, it becomes much easier to find the sentiment of each tweet: we can join the table with the lexicon table.

Add a `polarity` column to the `df_trump` table. The `polarity` column should contain the sum of the sentiment polarity of each word in the text of the tweet.

Hint you will need to merge the `tidy_format` and `df_sent` tables and group the final answer.

```
#df_trump['polarity'] = ...

### BEGIN ANSWER
tempdf = pd.merge(tidy_format, df_sent, right_index=True, left_on="word")
# your solution here
tempdf = tempdf.groupby(tempdf.index).sum()
df_trump["polarity"] = tempdf["polarity"]
df_trump["polarity"] = df_trump["polarity"].fillna(0)

### END ANSWER
df_trump
```

```
assert np.allclose(df_trump.loc[744701872456536064, 'polarity'], 8.4)
assert np.allclose(df_trump.loc[745304731346702336, 'polarity'], 2.5)
assert np.allclose(df_trump.loc[744519497764184064, 'polarity'], 1.7)
assert np.allclose(df_trump.loc[894661651760377856, 'polarity'], 0.2)
assert np.allclose(df_trump.loc[894620077634592769, 'polarity'], 5.4)
# If you fail this test, you dropped tweets with 0 polarity
assert np.allclose(df_trump.loc[744355251365511169, 'polarity'], 0.0)
```

Task 4.6

Now we have a measure of the sentiment of each of his tweets! You can read over the VADER readme to understand a more robust sentiment analysis.
Now, write the code to see the 20 most positive and most 20 negative tweets from Trump in your dataset:
Find the most negative and most positive tweets made by Trump

```
print('Most negative tweets:')

### BEGIN ANSWER
# most negative tweets: low polarity:
neg = df_trump.sort_values(by='polarity', ascending = True)['text'].values[:20]
neg
# your solution here

### END ANSWER
```

```
print('Most positive tweets:')

### BEGIN ANSWER
pos = df_trump.sort_values(by='polarity', ascending = False)['text'].values[:20]
pos
# your solution here

### END ANSWER
```

Task 4.7

Plot the distribution of tweet sentiments broken down by whether the text of the tweet contains `nyt` or `fox`. Then in the box below comment on what we observe?


```
### BEGIN ANSWER
nyt = df_trump[df_trump['text'].str.contains('nyt')]
fox = df_trump[df_trump['text'].str.contains('fox')]

plo = sns.distplot(nyt.polarity, rug = False, hist = True)
plo = sns.distplot(fox.polarity, rug = False, hist = True)
```

```
plo.set(xlabel = "", ylabel = "")
plo.legend(labels=['nyt','fox'], loc = "upper right")
sns.set(rc={'figure.figsize':(12,7)})
# your solution here

### END ANSWER
```

Comment on what you observe:

BEGIN ANSWER

According to the plot above, one can say the "fox" plot's polarity was mostly neutral. As we can see the highest peak of the historgram, which means that Fox was mostly mentioned in neutral tweets. The second highest peak lies between the polarity of 0 and 5 hence, proving that after neutral tweets fox was mostly in positive tweets. We can also observe the rest of the bars which show their presence mostly on the positive side. But it can't be denied that some tweets containing fox were on the negative side as well.

Whereas, in the case of "nyt", it's highest peak lies between some negative value of polarity and a little above 0. This shows that nyt was mostly mentioned in negative tweets, while it's second highest peak lies in the negative side which also proves the mention of negative tweets. If we talk about the rest of the bars, they mostly show their presence on the negative side of polarity thus proving nyt's mention mostly in negative tweets.

Hence, the above plot shows us how the polarity of tweets containing fox and nyt ranges form a negative value to a positive value. The more negative the value of polarity is, the more negative is the comment. Same for the positive side.

END ANSWER

PART 5 - Principal Component Analysis (PCA) and Twitter (group and individual)

A look at the top words used and the sentiments expressed in Trump tweets indicates that, some words are used with others almost all the time. A notable example is the slogan like **Make America Great Again**. As such, it may be beneficial to look at groups of words rather than individual words. For that, we will look at an approach applying a Principal Component Analysis.

The PCA

The Principal Component Analysis, or PCA, is a tool generally used to identify patterns and to reduce the number of variables you have to consider in your analysis. For example, if you have data with 200 columns, it may be that a significant amount of the variance in your data can be explained by just 100 principal components. In the PCA, the first component is chosen in such a way that has the largest variance, subsequent components are orthogonal and continue covering as much variance as possible. In this way, the PCA samples as much of the variability in the data set with the first few components. Mathematically, each component is a linear combination of all the input parameters times coefficients specific for that component. These coefficients, or loading factors, are constrained such that the sum of the squares of them are equal to 1. As such, the loading factors serve as weights describing

how strongly certain parameters contribute to the specific principal component. Parameters with large values of positive or negative loading factors are correlated with each other, which can serve to identify trends in your data.

Task 5.1 Cleaning up the Data

Using NLTK (Natural Language Toolkit) package for language processing and other python libraries, parse the json file to deal with inflected words, such as plurals, and removed stop words like common English words (the, and, it, etc) and certain political terms (the candidates names, for example). You can start with the top 50 words, but full analysis may require large number of words. Create a document-frequecy (df) matrix with 5000 rows and 50 columns where each column is a particular word (feature) and each row is a tweet (observation). The values of the matrix is how often the word appears. Apply the techniques we learned to reduce the weight of most common words (if necessary). Since this is a sparse matrix, you can use the sparse martix libraries to make things a bit more efficient (we can also use a regular numpy arrays to store these things since the dimensions are not too large). See demo notes books and lecture slides for some sparse matrix methods.
Print the first 10 rows of the df to show the matrix you created

Start with the `tidy_format` dataframe

```
### BEGIN ANSWER
## code to plot the first 10 rows of the matrix
import nltk
import nltk.corpus
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords



# create a dataframe called tmp to store all words appear in the tweets
tmp = pd.DataFrame(tidy_format["word"])
tmp

# remove stopwords
sw = stopwords.words('english')
tmp = tmp['word'][~tmp['word'].isin(sw)]
tmp = pd.DataFrame(tmp)

# deal with plurals
lemmatizer = WordNetLemmatizer()
tmp['word'] = tmp['word'].apply(lambda s: lemmatizer.lemmatize(s))
```

```
# Remove numbers
tmp['word'] = tmp['word'].str.replace('\d+', '', regex = True)

#Removing things like http:
tmp['word'] = tmp['word'].str.replace('http', '', regex = True)
tmp['word'] = tmp['word'].str.replace('\nhttps', '', regex = True)
tmp['word'] = tmp['word'].str.replace(r'^amp$', '', regex = True)



#Removing political terms: like candidate names:
tmp['word'] = tmp['word'].str.replace('trump', '', regex = True)
tmp['word'] = tmp['word'].str.replace('hillary', '', regex = True)
tmp['word'] = tmp['word'].str.replace('clinton', '', regex = True)
tmp['word'] = tmp['word'].str.replace('realdonaldtrump', '', regex = True)
tmp['word'] = tmp['word'].str.replace('realdonald', '', regex = True)
tmp['word'] = tmp['word'].str.replace('obama', '', regex = True)

# Remove words with only 1 or 2 length
tmp['word'] = tmp['word'].str.replace(r'\b(\w{1,2})\b', '', regex = True)
tmp = tmp.replace(r'^s*$', float('NaN'), regex = True)  # Replace blanks by NaN
tmp.dropna(inplace = True)    # Remove rows with NaN


#document frequency matrix:
words = tmp.value_counts(ascending=False)
top50words = words.reset_index()['word'][:50].to_list()
top50words

wordstoindex = {}
for i in range(len(top50words)):
    wordstoindex[top50words[i]] = i

X = np.zeros((5000, 50))

for i in range(5000):
    if tmp.iloc[i]['word'] in top50words:
        X[i, wordstoindex[tmp.iloc[i]['word']]] += 1
X[:10, :]
```

```
### END ANSWER
```

Task 5.2 Find the PCA's

Write the code to find the first 50 PCA's for the document-frequency matrix. Pass the document-term-matrix to scikit-learn's (https://scikit-learn.org/stable/modules/decomposition.html#decompositions) PCA method to obtain the components and loading factors.

```
### BEGIN ANSWER
from sklearn.decomposition import PCA

pca = PCA(n_components=50)
pca.fit(X)
print(pca.components_)
print(pca.explained_variance_)
component50 = pca.components_
# your solution here

### END ANSWER
```

Task 5.3 Examine the PCA

We can examine the PCA results to look at the heatmap. Make a grid plot which shows the various principal component along the x-axis and the individual words along the y-axes. Each grid box should be color-coded based on the sign of the loading factor and how large the square of that value is. Looking at it vertically, you can see which words constitute your principal components. Looking at it horizontally, you can see how individual terms are shared between components. Your answer will look closer to this.


```
### BEGIN ANSWER

pcalabel = []
for i in range(1,51):
    pcalabel.append('PC'+str(i))
plt.figure(figsize=(15,15))
cmap = sns.diverging_palette(100, 400, sep = 20, n = 250, as_cmap=True)
sns.heatmap(component50, cmap=cmap, yticklabels = top50words, xticklabels = pcalabel)
plt.show()
# your solution here
```

```
### END ANSWER
```

Task 5.4 PCA Compare

We can determine how many words and how many components are needed to do a good visualization. Plot PC1 and PC2 in a 2D plot. The results should be similar to following scatter plot

This is a scatter plot of the values of the components, but with arrows indicating some of the prominent terms as indicated by their loading factors. The values of the loading factors are used to determine the length and direction of these arrows and as such they serve as a way of expressing direction. That is, tweets which use these terms will be moved along the length of those arrows. Shown are the most important parameters.

```
### BEGIN ANSWER
pc1 = (component50[0])
pc2 = (component50[1])
fig = sns.jointplot(x=pc1, y=pc2)
fig.set_axis_labels('PC1', 'PC2', fontsize=16)
# your solution here

### END ANSWER
```

PART 6 - Twitter Engagement

In this problem, we'll explore which words led to a greater average number of retweets. For example, at the time of this writing, Donald Trump has two tweets that contain the word 'oakland' (tweets 932570628451954688 and 1016609920031117312) with 36757 and 10286 retweets respectively, for an average of 23,521.5.

Your `top_20` table should have this format:

	retweet_count
word	
---	---
jong	40675.666667
try	33937.800000
kim	32849.595745
un	32741.731707
maybe	30473.192308

Task 6.1

Find the top 20 most retweeted words. Include only words that appear in at least 25 tweets. As usual, try to do this without any for loops. You can string together ~5-7 pandas commands and get everything done on one line.

```
#top_20 = ...
### BEGIN ANSWER
tempor = tmp
#words present in atleast 25 tweets:
tempor = tempor.reset_index()
groups = tempor.groupby('word').count()
groups = groups[groups.id >= 25].sort_values('id', ascending = False)
groups = groups.reset_index()
top_20 = pd.merge(tempor, groups['word'], on = 'word')
top_20 = top_20.set_index('id')
top_20['retweet_count'] = df_trump['retweet_count']

#getting the top 20 words with most retweets
top_20 = top_20.groupby('word').mean().sort_values('retweet_count', ascending = False)[:20]
top_20
### END ANSWER
```

Task 6.2

Plot a bar chart of your results:

```
### BEGIN ANSWER
plto = sns.barplot(x=top_20["retweet_count"],y=top_20.index, orient = 'h')
plto.set(title = "Top 20 Most Retweeted Words")

# your solution here

### END ANSWER
```

PART 7 - Kim Jong Un and Musk Tweet Analysis (Optional for Individual)

This is for the groups to do. What else can we study? Let us ask some open ended questions.

Task 7.1

"kim", "jong" and "un" are apparently really popular in Trump's tweets! It seems like we can conclude that his tweets involving jong

are more popular than his other tweets. Or can we?

Consider each of the statements about possible confounding factors below. State whether each statement is true or false and explain. If the statement is true, state whether the confounding factor could have made kim jong un related tweets higher in the list than they should be.

1. We didn't restrict our word list to nouns, so we have unhelpful words like "let" and "any" in our result.
    - That might be why 'un' is the most popular.
2. We didn't remove hashtags in our text, so we have duplicate words (eg. #great and great).
    - Some may only have '#great' not 'great' which make the average lower
3. We didn't account for the fact that Trump's follower count has increased over time.
    - This can affect a lot. As Trump's follower count has increased, the more popular every word be

```
#plt.figure(figsize=(20,20))

### BEGIN ANSWER

# your solution here

### END ANSWER
```

Task 7.2

Using the `df_trump` tweets construct an interesting plot describing a property of the data and discuss what you found below.

Ideas:

1. How has the sentiment changed with length of the tweets?
2. Does sentiment affect retweet count?
3. Are retweets more negative than regular tweets?
4. Are there any spikes in the number of retweets and do the correspond to world events?
5. What terms have an especially positive or negative sentiment?

You can look at other data sources and even tweets. Do some plots and discuss. You can add more cells here as needed.

```
### BEGIN ANSWER

# your solution here

### END ANSWER
```

BEGIN ANSWER

Discussion: "Enter question you tried answering"

Answer:

END ANSWER

Task 7.3 - Elon Musk and Twitter

Elon Musk purchased Twitter in October 2022. In November, 2022, he laid off over 50% of the employees. In this task, extract the Musk tweets from 2022 (where the conversation about twitter purchase began). Call the dataframe df_musk_2022

```
### BEGIN ANSWER

# your solution here

### END ANSWER
df_musk_2022.head()
```

Task 7.4 - Elon Musk and PCA

Plot a heatmap similar to Task 5.3 for musk 2022 tweets

```
### BEGIN ANSWER

# your solution here

### END ANSWER
```

What are the most dominant themes in his 2022 tweets?

Task 7.5 - Elon Musk Twitter Analysis

Do a thorough analysis of how Elon Musk tweets emphasized why he want to buy twitter. This is an open ended question and be as detailed as possible.

BEGIN ANSWER

END ANSWER

Submission Instructions

Submission Instructions

File Name: Please rename the file as yourNetID_midsemester.jpynb

Group Projects: Each person in the group must submit a copy with both names listed and your partners contribution answered. If you are doing a group project, you must inform complete the form prior to 3/10 that you intend to work as a group and submit your name and your partner name via the form. https://forms.gle/RdTYYx7GRMQFVsvd7 We will **not accept group work** if your TA has not been notified.

Submit To: Codebench (Do not submit data files)

Warning: Failure to follow directions may result in loss points.

Created by Andy Guna @2019-2023 Credits: Josh Hug, and Berkeley Data Science Group, Steve Skiena, David Rodreguez