Exercise

March 20, 2024

1 Lab 3: Text Analysis (20 Pts)

[ ]:
```
# Run this cell to set up your notebook
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re


# Ensure that Pandas shows at least 280 characters in columns, so we can see␣
↪full tweets
pd.set_option('max_colwidth', 280)
plt.style.use('fivethirtyeight')
sns.set()
sns.set_context("talk")

def horiz_concat_df(dict_of_df, head=None):
    """
    Horizontally concatenante multiple DataFrames for easier visualization.
    Each DataFrame must have the same columns.
    """
    df = pd.concat([df.reset_index(drop=True) for df in dict_of_df.values()],␣
↪axis=1, keys=dict_of_df.keys())
    if head is None:
        return df
    return df.head(head)
```

1.1 Question 1: Importing the Data

The data for this assignment was obtained using the Twitter APIs. To ensure that everyone has the same data and to eliminate the need for every student to apply for a Twitter developer account, we have collected a sample of tweets from several high-profile public figures. The data is stored in the folder `data`. Run the following cell to list the contents of the directory:

[ ]:
```
# just run this cell
from os import listdir
```

```python
for f in listdir("data"):
    print(f)
```

```
AOC_recent_tweets.txt
EmmanuelMacron_recent_tweets.txt
Cristiano_recent_tweets.txt
elonmusk_recent_tweets.txt
BernieSanders_recent_tweets.txt
BillGates_recent_tweets.txt
```

1.1.1 Question 1a

Let's examine the contents of one of these files. Using the `open` function and `read` operation on a python file object, read the first 1000 **characters** in `data/BernieSanders_recent_tweets.txt` and store your result in the variable `q1a`. Then display the result so you can read it.

Caution: Viewing the contents of large files in a Jupyter notebook could crash your browser. Be careful not to print the entire contents of the file.

Hint: You might want to try to use `with`:

```python
with open("filename", "r") as f:
    f.read(2)
```

[ ]:
```python
with open("data/BernieSanders_recent_tweets.txt", "r") as file:
    q1a = file.read(1000)
print(q1a)
```


```
[{"created_at": "Sat Feb 06 22:43:03 +0000 2021", "id": 1358184460794163202,
"id_str": "1358184460794163202", "full_text": "Why would we want to impeach and
convict Donald Trump \u2013 a president who is now out of office? Because it
must be made clear that no president, now or in the future, can lead an
insurrection against the government he or she is sworn to protect.",
"truncated": false, "display_text_range": [0, 243], "entities": {"hashtags": [],
"symbols": [], "user_mentions": [], "urls": []}, "source": "<a
href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for
iPhone</a>", "in_reply_to_status_id": null, "in_reply_to_status_id_str": null,
"in_reply_to_user_id": null, "in_reply_to_user_id_str": null,
"in_reply_to_screen_name": null, "user": {"id": 216776631, "id_str":
"216776631", "name": "Bernie Sanders", "screen_name": "BernieSanders",
"location": "Vermont", "description": "U.S. Senator for Vermont. Not me, us.",
"url": "https://t.co/jpg8Sp1GhR", "entities": {"
```

1.1.2 Question 1b

What format is the data in? Answer this question by entering the letter corresponding to the right format in the variable `q1b` below.

A. CSV B. HTML C. JavaScript Object Notation (JSON) D. Excel XML

Answer in the following cell. Your answer should be a string, either "A", "B", "C", or "D".

[ ]:
```
q1b = "C"
```

1.1.3 Question 1c

Pandas has built-in readers for many different file formats including the file format used here to store tweets. To learn more about these, check out the documentation for `pd.read_csv`, `pd.read_html`, `pd.read_json`, and `pd.read_excel`.

1. Use one of these functions to populate the `tweets` dictionary with the tweets for: `AOC`, `Cristiano`, and `elonmusk`. The keys of `tweets` should be the handles of the users, which we have provided in the cell below, and the values should be the DataFrames.
2. Set the index of each DataFrame to correspond to the `id` of each tweet.

Hint: You might want to first try loading one of the DataFrames before trying to complete the entire question.

[ ]:
```
import os
```

[ ]:
```
filename = ["AOC_recent_tweets.txt",
            "Cristiano_recent_tweets.txt",
            "elonmusk_recent_tweets.txt"]
df_list = []

for target in filename:
    with open(os.path.join("data",target), "r") as file:
        df = pd.read_json(file)
        df = df.set_index("id")
        df_list.append(df)
```

[ ]:
```
tweets = {}
for item in zip(df_list,filename):
    tweets[item[1].removesuffix("_recent_tweets.txt")] = item[0]

print(tweets.keys())
```

```
dict_keys(['AOC', 'Cristiano', 'elonmusk'])
```

If you did everything correctly, the following cells will show you the first 5 tweets for Elon Musk (and a lot of information about those tweets).

[ ]:
```
# just run this cell
tweets["elonmusk"].head()
```

[ ]:
```
                                    created_at                id_str  \
id
```

```
1357991946082418690 2021-02-06 09:58:04+00:00  1357991946082418688
1357973565413367808 2021-02-06 08:45:02+00:00  1357973565413367808
1357972904663687173 2021-02-06 08:42:25+00:00  1357972904663687168
1357970517165182979 2021-02-06 08:32:55+00:00  1357970517165182976
1357964347813687296 2021-02-06 08:08:24+00:00  1357964347813687296

                                                                  full_text  \
id
1357991946082418690        The Second Last Kingdom https://t.co/Je4EI88HmV
1357973565413367808  @DumDin7 @Grimezsz Haven't heard that name in years …
1357972904663687173                                     @Grimezsz Dogecake
1357970517165182979                          YOLT\n\nhttps://t.co/cnOf9yjpF1
1357964347813687296                             @Kristennetten That's Damian

                    truncated display_text_range  \
id
1357991946082418690      False            [0, 23]
1357973565413367808      False           [19, 53]
1357972904663687173      False           [10, 18]
1357970517165182979      False            [0, 29]
1357964347813687296      False           [15, 28]

                                                             entities  \
id
1357991946082418690  {'hashtags': [], 'symbols': [], 'user_mentions': [],
'urls': [], 'media': [{'id': 1357991942471094275, 'id_str':
'1357991942471094275', 'indices': [24, 47], 'media_url':
'http://pbs.twimg.com/media/EtiOegrVEAMCgZE.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtiOegrV…
1357973565413367808  {'hashtags': [], 'symbols': [], 'user_mentions':
[{'screen_name': 'DumDin7', 'name': 'Dum Din', 'id': 1279896279733145601,
'id_str': '1279896279733145601', 'indices': [0, 8]}, {'screen_name': 'Grimezsz',
'name': '         ', 'id': 276540738, 'id_str': '276540738', 'indi…
1357972904663687173
{'hashtags': [], 'symbols': [], 'user_mentions': [{'screen_name': 'Grimezsz',
'name': '         ', 'id': 276540738, 'id_str': '276540738', 'indices':
[0, 9]}], 'urls': []}
1357970517165182979
{'hashtags': [], 'symbols': [], 'user_mentions': [], 'urls': [{'url':
'https://t.co/cnOf9yjpF1', 'expanded_url':
'https://m.youtube.com/watch?v=05QJlF06F4s', 'display_url':
'm.youtube.com/watch?v=05QJlF…', 'indices': [6, 29]}]}
1357964347813687296
{'hashtags': [], 'symbols': [], 'user_mentions': [{'screen_name':
'Kristennetten', 'name': 'K10 ', 'id': 985686123123949568, 'id_str':
'985686123123949568', 'indices': [0, 14]}], 'urls': []}
```

```
                                                  extended_entities  \
id
1357991946082418690  {'media': [{'id': 1357991942471094275, 'id_str':
'1357991942471094275', 'indices': [24, 47], 'media_url':
'http://pbs.twimg.com/media/EtiOegrVEAMCgZE.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtiOegrVEAMCgZE.jpg', 'url':
'https://t.co/Je4EI88HmV', 'display_url': '…
1357973565413367808
NaN
1357972904663687173
NaN
1357970517165182979
NaN
1357964347813687296
NaN

                                 source  \
id
1357991946082418690  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1357973565413367808  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1357972904663687173  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1357970517165182979  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1357964347813687296  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>

                     in_reply_to_status_id  in_reply_to_status_id_str  …  \
id                                                                      …
1357991946082418690                    NaN                        NaN  …
1357973565413367808           1.357973e+18               1.357973e+18  …
1357972904663687173           1.357835e+18               1.357835e+18  …
1357970517165182979                    NaN                        NaN  …
1357964347813687296           1.357964e+18               1.357964e+18  …

                     favorite_count  favorited retweeted possibly_sensitive  \
id
1357991946082418690          352096      False     False                0.0
1357973565413367808            2155      False     False                NaN
1357972904663687173            5373      False     False                NaN
1357970517165182979           62717      False     False                0.0
1357964347813687296            5726      False     False                NaN

                     lang  retweeted_status  quoted_status_id  \
id
```

```
1357991946082418690     en                NaN                NaN
1357973565413367808     en                NaN                NaN
1357972904663687173     en                NaN                NaN
1357970517165182979     en                NaN                NaN
1357964347813687296     en                NaN                NaN

                     quoted_status_id_str  quoted_status_permalink  \
id
1357991946082418690                   NaN                      NaN
1357973565413367808                   NaN                      NaN
1357972904663687173                   NaN                      NaN
1357970517165182979                   NaN                      NaN
1357964347813687296                   NaN                      NaN

                     quoted_status
id
1357991946082418690            NaN
1357973565413367808            NaN
1357972904663687173            NaN
1357970517165182979            NaN
1357964347813687296            NaN

[5 rows x 30 columns]
```

1.2 Question 1d

There are many ways we could choose to read tweets. Why might someone be interested in doing data analysis on tweets? Name a kind of person or institution which might be interested in this kind of analysis. Then, give two reasons why a data analysis of tweets might be interesting or useful for them. Answer in 2-3 sentences.

Type your answer here, replacing this text.

1.3 Question 2: Source Analysis

In some cases, the Twitter feed of a public figure may be partially managed by a public relations firm. In these cases, the device used to post the tweet may help reveal whether it was the individual (e.g., from an iPhone) or a public relations firm (e.g., TweetDeck). The tweets we have collected contain the source information but it is formatted strangely :(

[ ]:
```
# just run this cell
tweets["Cristiano"][["source"]]
```

[ ]:
```
                                  source
id
1358137564587319299  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
```

```
1357379984399212545  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1356733030962987008  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1355924395064233986  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1355599316300292097  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
…
…
32514882561638401                              <a href="http://www.whosay.com"
rel="nofollow">WhoSay</a>
32513604662071296                              <a href="http://www.whosay.com"
rel="nofollow">WhoSay</a>
32511823722840064                              <a href="http://www.whosay.com"
rel="nofollow">WhoSay</a>
32510294081146881                              <a href="http://www.whosay.com"
rel="nofollow">WhoSay</a>
32508748819857410                              <a href="http://www.whosay.com"
rel="nofollow">WhoSay</a>

[3198 rows x 1 columns]
```

In this question we will use a regular expression to convert this messy HTML snippet into something more readable. For example: `<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>` should be `Twitter for iPhone`.

1.3.1 Question 2a

We will first use the Python `re` library to cleanup the above test string. In the cell below, write a regular expression that will match the **HTML tag** and assign it to the variable `q2a_pattern`. We then use the `re.sub` function to substitute anything that matches the pattern with an empty string `""`.

An HTML tag is defined as a `<` character followed by zero or more non-`>` characters, followed by a `>` character. That is `<a>` and `</a>` are both considered *separate* HTML tags.

[ ]:
```python
q2a_pattern = r"<[^>]*>"
test_str = '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter␣
↪for iPhone</a>'
re.sub(q2a_pattern, "", test_str)
```

[ ]:
```
'Twitter for iPhone'
```

1.3.2 Question 2b

Rather than writing a regular expression to detect and remove the HTML tags we could instead write a regular expression to **capture** the device name between the angle brackets. Here we will use **capturing groups** by placing parenthesis around the part of the regular expression we want to return. For example, to capture the 21 in the string `08/21/83` we could use the pattern `r"08/(..)/83"`.

Hint: The output of the following cell should be `['Twitter for iPhone']`.

[ ]:
```
q2b_pattern = r">(.+)<"
test_str = '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter␣
↪for iPhone</a>'
re.findall(q2b_pattern, test_str)
```

[ ]:
```
['Twitter for iPhone']
```

1.3.3 Question 2c

Using either of the two regular expressions you just created and `Series.str.replace` or `Series.str.extract`, add a new column called `"device"` to **all** of the DataFrames in `tweets` containing just the text describing the device (without the HTML tags).

[ ]:
```
for item in tweets.keys():
    tweets[item]["device"] =tweets[item]["source"].str.extract(q2b_pattern)

tweets["Cristiano"].head()
```

[ ]:
```
                                   created_at               id_str  \
id
1358137564587319299 2021-02-06 19:36:43+00:00  1358137564587319296
1357379984399212545 2021-02-04 17:26:21+00:00  1357379984399212544
1356733030962987008 2021-02-02 22:35:36+00:00  1356733030962987008
1355924395064233986 2021-01-31 17:02:22+00:00  1355924395064233984
1355599316300292097 2021-01-30 19:30:37+00:00  1355599316300292096

                                                                 full_text
\
id
1358137564587319299  Happy to score and help the team against a tough opponent!
3 important points! \nWell done lads    #finoallafine https://t.co/bVHENpx2X6
1357379984399212545
Done   \nHave a good day!   https://t.co/DN9lo4gMbS
1356733030962987008                                        Grande vittoria di
squadra! Abbiamo bisogno di questo spirito #finoallafine
https://t.co/lNyV5hGE2n
1355924395064233986
```

```
Home sweet home!   https://t.co/7MaSXDfTYm
1355599316300292097                                                   Altri 3
punti importantissimi ! Avanti così    #finoallafine https://t.co/l5HfUkfLcS

                     truncated display_text_range  \
id
1358137564587319299      False          [0, 113]
1357379984399212545      False           [0, 29]
1356733030962987008      False           [0, 81]
1355924395064233986      False           [0, 20]
1355599316300292097      False           [0, 63]

                                                            entities  \
id
1358137564587319299  {'hashtags': [{'text': 'finoallafine', 'indices': [100,
113]}], 'symbols': [], 'user_mentions': [], 'urls': [], 'media': [{'id':
1358137559772246023, 'id_str': '1358137559772246023', 'indices': [114, 137],
'media_url': 'http://pbs.twimg.com/media/EtkS6jZXMAcdl-P.jpg', 'media_…
1357379984399212545  {'hashtags': [], 'symbols': [], 'user_mentions': [],
'urls': [], 'media': [{'id': 1357379979147964421, 'id_str':
'1357379979147964421', 'indices': [30, 53], 'media_url':
'http://pbs.twimg.com/media/EtZh5jpXcAUg0BM.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtZh5jpX…
1356733030962987008  {'hashtags': [{'text': 'finoallafine', 'indices': [62,
75]}], 'symbols': [], 'user_mentions': [], 'urls': [], 'media': [{'id':
1356733026261225473, 'id_str': '1356733026261225473', 'indices': [82, 105],
'media_url': 'http://pbs.twimg.com/media/EtQVf8VXUAE7nJj.jpg', 'media_url…
1355924395064233986  {'hashtags': [], 'symbols': [], 'user_mentions': [],
'urls': [], 'media': [{'id': 1355924390752505857, 'id_str':
'1355924390752505857', 'indices': [21, 44], 'media_url':
'http://pbs.twimg.com/media/EtE2DKUXUAEOtyN.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtE2DKUX…
1355599316300292097  {'hashtags': [{'text': 'finoallafine', 'indices': [50,
63]}], 'symbols': [], 'user_mentions': [], 'urls': [], 'media': [{'id':
1355599311493607430, 'id_str': '1355599311493607430', 'indices': [64, 87],
'media_url': 'http://pbs.twimg.com/media/EtAOZDtXMAYYJSv.jpg', 'media_url_…

                                                   extended_entities  \
id
1358137564587319299  {'media': [{'id': 1358137559772246023, 'id_str':
'1358137559772246023', 'indices': [114, 137], 'media_url':
'http://pbs.twimg.com/media/EtkS6jZXMAcdl-P.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtkS6jZXMAcdl-P.jpg', 'url':
'https://t.co/bVHENpx2X6', 'display_url':…
1357379984399212545  {'media': [{'id': 1357379979147964421, 'id_str':
'1357379979147964421', 'indices': [30, 53], 'media_url':
'http://pbs.twimg.com/media/EtZh5jpXcAUg0BM.jpg', 'media_url_https':
```

```
'https://pbs.twimg.com/media/EtZh5jpXcAUgOBM.jpg', 'url':
'https://t.co/DN9lo4gMbS', 'display_url': '…
1356733030962987008  {'media': [{'id': 1356733026261225473, 'id_str':
'1356733026261225473', 'indices': [82, 105], 'media_url':
'http://pbs.twimg.com/media/EtQVf8VXUAE7nJj.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtQVf8VXUAE7nJj.jpg', 'url':
'https://t.co/lNyV5hGE2n', 'display_url': …
1355924395064233986  {'media': [{'id': 1355924390752505857, 'id_str':
'1355924390752505857', 'indices': [21, 44], 'media_url':
'http://pbs.twimg.com/media/EtE2DKUXUAE0tyN.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtE2DKUXUAE0tyN.jpg', 'url':
'https://t.co/7MaSXDfTYm', 'display_url': '…
1355599316300292097  {'media': [{'id': 1355599311493607430, 'id_str':
'1355599311493607430', 'indices': [64, 87], 'media_url':
'http://pbs.twimg.com/media/EtAOZDtXMAYYJSv.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtAOZDtXMAYYJSv.jpg', 'url':
'https://t.co/l5HfUkfLcS', 'display_url': '…

                                                                 source  \
id
1358137564587319299  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1357379984399212545  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1356733030962987008  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1355924395064233986  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1355599316300292097  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>

                     in_reply_to_status_id  in_reply_to_status_id_str  …  \
id                                                                     …
1358137564587319299                    NaN                        NaN  …
1357379984399212545                    NaN                        NaN  …
1356733030962987008                    NaN                        NaN  …
1355924395064233986                    NaN                        NaN  …
1355599316300292097                    NaN                        NaN  …

                     favorited  retweeted possibly_sensitive lang  \
id
1358137564587319299      False      False                0.0   en
1357379984399212545      False      False                0.0   en
1356733030962987008      False      False                0.0   it
1355924395064233986      False      False                0.0   en
1355599316300292097      False      False                0.0   it
```

```
                     quoted_status_id  quoted_status_id_str  \
id
1358137564587319299               NaN                   NaN
1357379984399212545               NaN                   NaN
1356733030962987008               NaN                   NaN
1355924395064233986               NaN                   NaN
1355599316300292097               NaN                   NaN

                    quoted_status_permalink  quoted_status  retweeted_status  \
id
1358137564587319299                     NaN            NaN               NaN
1357379984399212545                     NaN            NaN               NaN
1356733030962987008                     NaN            NaN               NaN
1355924395064233986                     NaN            NaN               NaN
1355599316300292097                     NaN            NaN               NaN

                                device
id
1358137564587319299  Twitter for iPhone
1357379984399212545  Twitter for iPhone
1356733030962987008  Twitter for iPhone
1355924395064233986  Twitter for iPhone
1355599316300292097  Twitter for iPhone

[5 rows x 31 columns]
```

1.3.4 Question 2d

To examine the most frequently used devices by each individual, implement the `most_freq` function that takes in a `Series` and returns a new `Series` containing the `k` most commonly occuring entries in the first series, where the values are the counts of the entries and the indices are the entries themselves.

For example:

`most_freq(pd.Series(["A", "B", "A", "C", "B", "A"]), k=2)`

would return:

```
A    3
B    2
dtype: int64
```

Hint Consider using `value_counts`, `sort_values`, `head`, and/or `nlargest` (for the last one, read the documentation here). Think of what might be the most efficient implementation.

[ ]:
```
series = tweets["Cristiano"]['device']
```

```
[ ]: %%timeit
grouped = series.groupby(series)
grouped.count().nlargest(5)
```

```
354 µs ± 3.2 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

```
[ ]: %%timeit
series.value_counts().sort_values(ascending=False).head(5)
```

```
147 µs ± 2.54 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

```
[ ]: # as from the findings above, we use the faster code block
def most_freq(series, k = 5):
    return series.value_counts().sort_values(ascending=False).head(k)

most_freq(tweets["Cristiano"]['device'])
```

```
[ ]: device
Twitter for iPhone     1183
Twitter Web Client      959
WhoSay                  453
MobioINsider.com        144
Twitter for Android     108
Name: count, dtype: int64
```

Run the following two cells to compute a table and plot describing the top 5 most commonly used devices for each user.

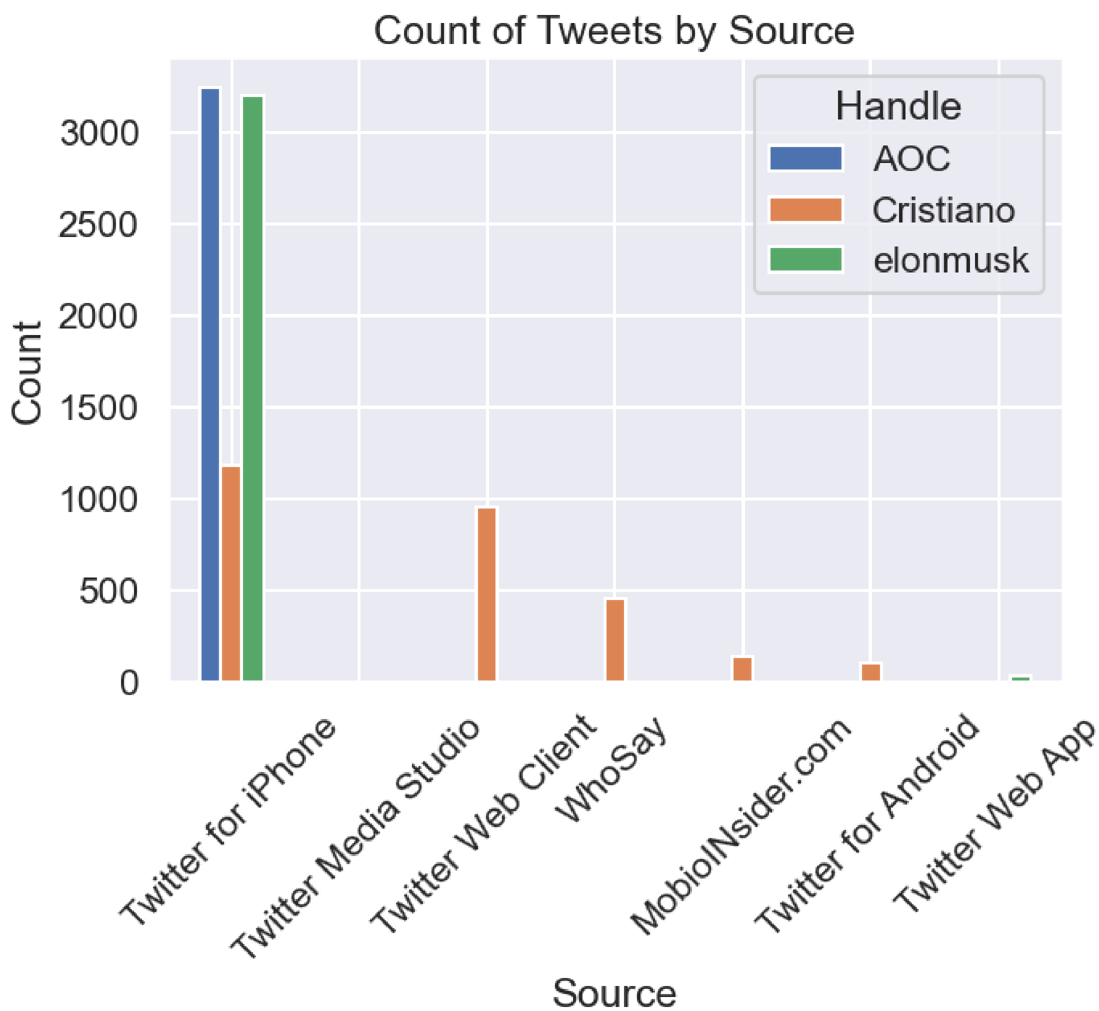
```
[ ]: # just run this cell
device_counts = pd.DataFrame(
    [most_freq(tweets[name]['device']).rename(name)
     for name in tweets]
).fillna(0)
device_counts
```

```
[ ]: device     Twitter for iPhone  Twitter Media Studio  Twitter Web Client  \
AOC                    3245.0                   2.0                 0.0
Cristiano              1183.0                   0.0               959.0
elonmusk               3202.0                   0.0                 0.0

device     WhoSay  MobioINsider.com  Twitter for Android  Twitter Web App
AOC           0.0               0.0                  0.0              0.0
Cristiano   453.0             144.0                108.0              0.0
elonmusk      0.0               0.0                  0.0             37.0
```

```
[ ]: # just run this cell
device_counts.T.plot.bar(xlabel="Source",ylabel="Count",title="Count of Tweets␣
 ↪by Source")
plt.xticks(rotation=45)
```

```
plt.legend(title="Handle")
```

[ ]: <matplotlib.legend.Legend at 0x179e80bf0>



1.3.5 Question 2e

What might we want to investigate further? Write a few sentences below.

AOC mainly has iPhone users. elonmusk also sees the same distribution of users. However, Chirstiano has a wider variety of users from different platform which is vastly different from the other 2. We can try to investigate if there is some indicative variable that allows us to identify why there is such a trend.

1.3.6 Question 2f

We just looked at the top 5 most commonly used devices for each user. However, we used the number of tweets as a measure, when it might be better to compare these distributions by comparing *proportions* of tweets. Why might proportions of tweets be better measures than numbers of tweets?

2 TODO

When using proportion of tweets, we are able to to compare the values based on the distribution of the data with other distributions.

2.1 Question 3: When?

Now that we've explored the sources of each of the tweets, we will perform some time series analysis. A look into the temporal aspect of the data could reveal insights about how a user spends their day, when they eat and sleep, etc. In this question, we will focus on the time at which each tweet was posted.

2.1.1 Question 3a

Complete the following function `add_hour` that takes in a tweets dataframe `df`, and two column names `time_col` and `result_col`. Your function should use the timestamps in the `time_col` column to store in a new column `result_col` the computed hour of the day as floating point number according to the formula:

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

Note: The below code calls your `add_hour` function and updates each tweets dataframe by using the `created_at` timestamp column to calculate and store the `hour` column.

Hint: See the following link for an example of working with timestamps using the `dt` accessors.

[ ]:
```
def add_hour(df, time_col, result_col):
    df[result_col] = df[time_col].dt.hour + (df[time_col].dt.minute / 60) +␣
 ↪(df[time_col].dt.second / 60**2)
    return df

# do not modify the below code
tweets = {handle: add_hour(df, "created_at", "hour") for handle, df in tweets.
 ↪items()}
tweets["AOC"]["hour"].head()
```

[ ]:
```
id
1358149122264563712    20.377222
1358147616400408576    20.277500
1358145332316667909    20.126389
1358145218407759875    20.118611
```

```
1358144207333036040    20.051667
Name: hour, dtype: float64
```

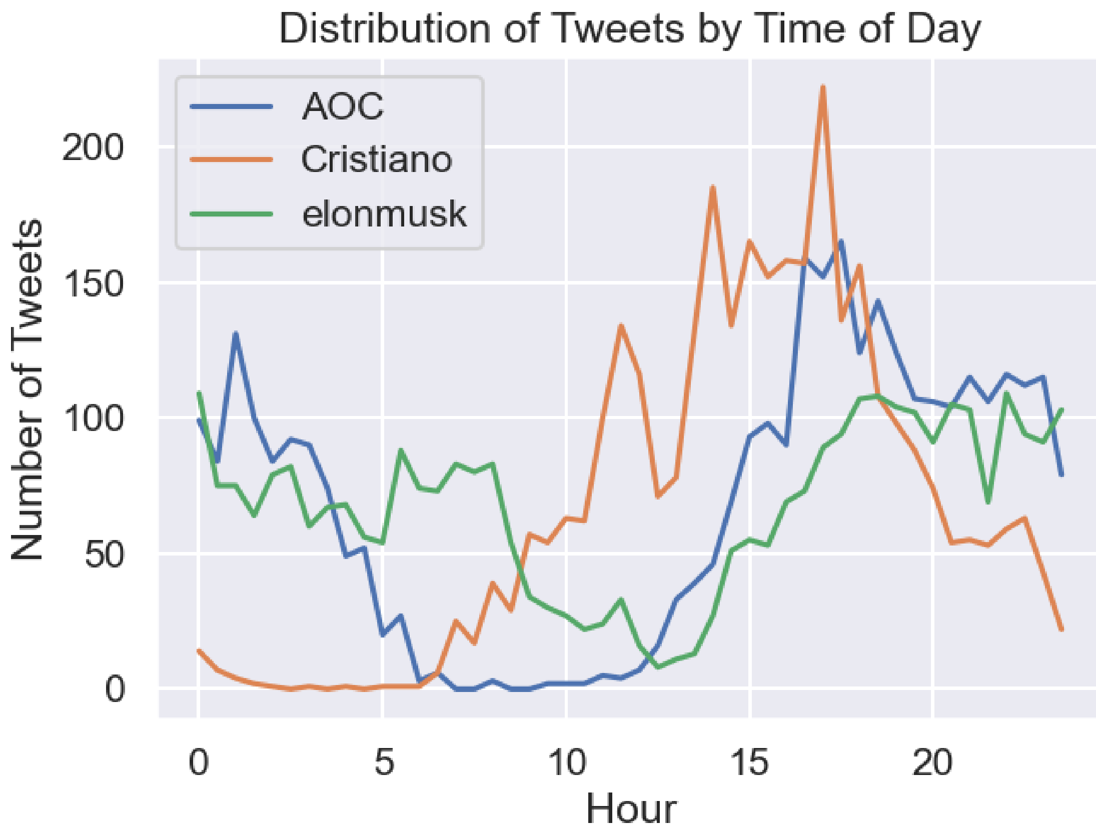
With our new `hour` column, let's take a look at the distribution of tweets for each user by time of day. The following cell helps create a density plot on the number of tweets based on the hour they are posted.

The function `bin_df` takes in a dataframe, an array of bins, and a column name; it bins the the values in the specified column, returning a dataframe with the bin lower bound and the number of elements in the bin. This function uses `pd.cut`, a pandas utility for binning numerical values that you may find helpful in the distant future.

Run the cell and answer the following question about the plot.

```
[ ]: # just run this cell
def bin_df(df, bins, colname):
    binned = pd.cut(df[colname], hour_bins).value_counts().sort_index()

    return pd.DataFrame({"counts": binned, "bin": bins[:-1]})

hour_bins = np.arange(0, 24.5, .5)
binned_hours = {handle: bin_df(df, hour_bins, "hour") for handle, df in tweets.
 ↪items()}

for handle, df in binned_hours.items():
        sns.lineplot(x="bin", y="counts", data=df, label=handle)
plt.title("Distribution of Tweets by Time of Day")
plt.xlabel("Hour")
plt.ylabel("Number of Tweets")
plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x17a444f20>
```



2.1.2 Question 3b

Compare Cristiano's distribution with those of AOC and Elon Musk. In particular, compare the distributions before and after Hour 6. What differences did you notice? What might be a possible cause of that? Do the data plotted above seem reasonable?

Before hour 6, the number of tweets by Cristiano was low, while AOC and elonmusk was high, however, we see that Cristiano had little tweets then. After hour 6, we start to see that the numver of tweets for AOC and elonmusk decreasing while Cristiano's was rapidly rising. My guess is that it seems there is a high and low time for these 3 users and that seems like the users have a active time of the day, and the low could be when they are sleeping. Then it could be that the low and high of these users are not around the same period because of the time difference because we did not account for the different time zones previously.

2.1.3 Question 3c

To account for different locations of each user in our analysis, we will next adjust the `created_at` timestamp for each tweet to the respective timezone of each user. Complete the following function `convert_timezone` that takes in a tweets dataframe `df` and a timezone `new_tz` and adds a new column `converted_time` that has the adjusted `created_at` timestamp for each tweet. The timezone for each user is provided in `timezones`.

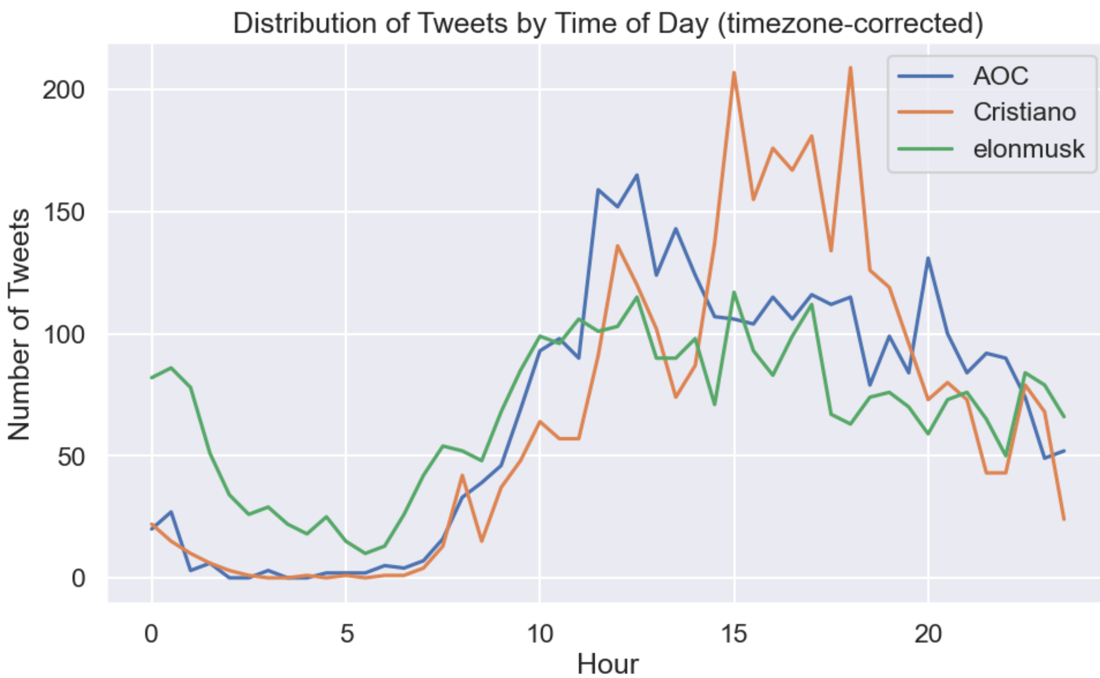
Hint: Again, please see the following link for an example of working with `dt` accessors.

```
[ ]: def convert_timezone(df, new_tz):
         df['converted_time']= df['created_at'].dt.tz_convert(new_tz)
         return df

     timezones = {"AOC": "EST", "Cristiano": "Europe/Lisbon", "elonmusk": "America/
      ↪Los_Angeles"}

     tweets = {handle: convert_timezone(df, tz) for (handle, df), tz in zip(tweets.
      ↪items(), timezones.values())}
```

With our adjusted timestamps for each user based on their timezone, let's take a look again at the distribution of tweets by time of day.

```
[ ]: # just run this cell
     def make_line_plot(df_dict, x_col, y_col, include=None, title=None,␣
      ↪xlabel=None, ylabel=None, legend=True):
         """
         Plot a line plot of two columns for each dataframe in `df_dict`.

         Uses `sns.lineplot` to plot a line plot of two columns for each
         dataframe in `df_dict`. The keys of `df_dict` are used as entries in
         the legend when `legend` is `True`.

         Parameters
         ----------
             df_dict: dict[str: pd.DataFrame]
                 a dictionary mapping handles to dataframes with the data to plot
             x_col: str
                 the name of a column in each dataframe in `df_dict` to plot on
                 the x-axis
             y_col: str
                 the name of a column in each dataframe in `df_dict` to plot on
                 the y-axis
             include: list[str], optional
                 a list of handles to include in the plot; all keys in `df_dict` not
                 present in `include`, if specified, will *not* be included in the␣
      ↪plot
             title: str, optional
                 a title for the plot
             xlabel: str, optional
                 a label for the x-axis; if unspecified, `x_col` is used
             ylabel: str, optional
                 a label for the y-axis; if unspecified, `y_col` is used
```

```python
        legend: bool, optional
            whether to include a legend with each key in `df_dict`
    """
    import matplotlib.pyplot as plt
    import seaborn as sns

    if include is not None:
        df_dict = {k: v for k, v in df_dict.items() if k in include}

    plt.figure(figsize=[10,6])
    for handle, df in df_dict.items():
        sns.lineplot(x=x_col, y=y_col, data=df, label=handle)
    if title:
        plt.title(title)
    if xlabel:
        plt.xlabel(xlabel)
    if ylabel:
        plt.ylabel(ylabel)
    if not legend:
        plt.gca().get_legend().remove()

tweets = {handle: add_hour(df, "converted_time", "converted_hour") for handle, df in tweets.items()}
binned_hours = {handle: bin_df(df, hour_bins, "converted_hour") for handle, df in tweets.items()}

make_line_plot(binned_hours, "bin", "counts", title="Distribution of Tweets by Time of Day (timezone-corrected)",
               xlabel="Hour", ylabel="Number of Tweets")
```



2.2 Question 4: Sentiment

In the past few questions, we have explored the sources of the tweets and when they are posted. Although on their own, they might not seem particularly intricate, combined with the power of regular expressions, they could actually help us infer a lot about the users. In this section, we will continue building on our past analysis and specifically look at the sentiment of each tweet – this would lead us to a much more direct and detailed understanding of how the users view certain subjects and people.

How do we actually measure the sentiment of each tweet? In our case, we can use the words in the text of a tweet for our calculation! For example, the word "love" within the sentence "I love America!" has a positive sentiment, whereas the word "hate" within 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 AOC'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:

[ ]:
```
# just run this cell
print(''.join(open("vader_lexicon.txt").readlines()[:10]))
```

```
$:      -1.5    0.80623 [-1, -1, -1, -1, -3, -1, -3, -1, -2, -1]
%)      -0.4    1.0198  [-1, 0, -1, 0, 0, -2, -1, 2, -1, 0]
```

```
%-)     -1.5    1.43178 [-2, 0, -2, -2, -1, 2, -2, -3, -2, -3]
&-:     -0.4    1.42829 [-3, -1, 0, 0, -1, -1, -1, 2, -1, 2]
&:      -0.7    0.64031 [0, -1, -1, -1, 1, -1, -1, -1, -1, -1]
( '}{' )        1.6     0.66332 [1, 2, 2, 1, 1, 2, 2, 1, 3, 1]
(%      -0.9    0.9434  [0, 0, 1, -1, -1, -1, -2, -2, -1, -2]
('-:    2.2     1.16619 [4, 1, 4, 3, 1, 2, 3, 1, 2, 1]
(':     2.3     0.9     [1, 3, 3, 2, 2, 4, 2, 3, 1, 2]
((-:    2.1     0.53852 [2, 2, 2, 1, 2, 3, 2, 2, 3, 2]
```

As you can see, the lexicon contains emojis too! Each row contains a word and the *polarity* of that word, measuring how positive or negative the word is.

2.2.1 VADER Sentiment Analysis

The creators of VADER describe the tool's assessment of polarity, or "compound score," in the following way:

"The compound score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive). This is the most useful metric if you want a single unidimensional measure of sentiment for a given sentence. Calling it a 'normalized, weighted composite score' is accurate."

As you can see, VADER doesn't "read" sentences, but works by parsing sentences into words, assigning a preset generalized score from their testing sets to each word separately.

VADER relies on humans to stabilize its scoring. The creators use Amazon Mechanical Turk, a crowdsourcing survey platform, to train its model. Its training data consists of a small corpus of tweets, New York Times editorials and news articles, Rotten Tomatoes reviews, and Amazon product reviews, tokenized using the natural language toolkit (NLTK). Each word in each dataset was reviewed and rated by at least 20 trained individuals who had signed up to work on these tasks through Mechanical Turk.

2.2.2 Question 4a

Please score the sentiment of one of the following words, using your own personal interpretation. No code is required for this question!

- police
- order
- Democrat
- Republican
- gun
- dog
- technology
- TikTok
- security
- face-mask

- science
- climate change
- vaccine

What score did you give it and why? Can you think of a situation in which this word would carry the opposite sentiment to the one you've just assigned?

- police: -1
- order: +1
- Democrat: +1
- Republican: -1
- gun: -1
- dog: +1
- technology: +1
- TikTok: -1
- security: +1
- face-mask: +1
- science: +1
- climate change: -1
- vaccine: +1

I gave extreme scores for this exercise based on the general notion of the public. Typically when we see some of these words in context, the context tends to carry a connotation which is tough to scale without any textbody at the moment.

2.2.3 Question 4b

Let's first load in the data containing all the sentiments. Read `vader_lexicon.txt` into a dataframe called `sent`. The index of the dataframe should be the words in the lexicon and should be named `token`. `sent` should have one column named `polarity`, storing the polarity of each word.

Hint: The `pd.read_csv` function may help here. Since the file is tab-separated, be sure to set `sep='\t'` in your call to `pd.read_csv`.

[ ]:
```
sent = pd.read_csv("vader_lexicon.txt", sep="\t", header=None, names=["token",␣
↪"polarity", "sd", "raw"], index_col=0)
sent.head()
```

[ ]:
```
       polarity       sd                                        raw
token
$:         -1.5  0.80623  [-1, -1, -1, -1, -3, -1, -3, -1, -2, -1]
%)         -0.4  1.01980       [-1, 0, -1, 0, 0, -2, -1, 2, -1, 0]
%-)        -1.5  1.43178    [-2, 0, -2, -2, -1, 2, -2, -3, -2, -3]
&-:        -0.4  1.42829      [-3, -1, 0, 0, -1, -1, -1, 2, -1, 2]
&:         -0.7  0.64031    [0, -1, -1, -1, 1, -1, -1, -1, -1, -1]
```

2.2.4 Question 4c

Before further analysis, we will need some more tools that can help us extract the necessary information and clean our data.

Complete the following regular expressions that will help us match part of a tweet that we either (i) want to remove or (ii) are interested in learning more about.

Question 4c Part (i) Assign a regular expression to a new variable `punct_re` that captures all of the punctuations within a tweet. We consider punctuation to be any non-word, non-whitespace character.

Note: A word character is any character that is alphanumeric or an underscore. A whitespace character is any character that is a space, a tab, a new line, or a carriage return.

[ ]:
```
punct_re = r"[^\w]|\s"
re.sub(punct_re, " ", tweets["AOC"].iloc[0]["full_text"])
```

[ ]:
```
'RT  RepEscobar  Our country has the moral obligation and responsibility to
reunite every single family separated at the southern border   T '
```

Question 4c Part (ii) Assign a regular expression to a new variable `mentions_re` that matches any mention in a tweet. Your regular expression should use a capturing group to extract the user's username in a mention.

Hint: a user mention within a tweet always starts with the `@` symbol and is followed by a series of word characters (with no space in between).

[ ]:
```
mentions_re = r"@\w+"
re.findall(mentions_re, tweets["AOC"].iloc[0]["full_text"])
```

[ ]:
```
['@RepEscobar']
```

2.2.5 Tweet Sentiments and User Mentions

As you have seen in the previous part of this question, there are actually a lot of interesting components that we can extract out of a tweet for further analysis! For the rest of this question though, we will focus on one particular case: the sentiment of each tweet in relation to the users mentioned within it.

To calculate the sentiments for a sentence, we will follow this procedure:

1. Remove the punctuation from each tweet so we can analyze the words.
2. For each tweet, find the sentiment of each word.
3. Calculate the sentiment of each tweet by taking the sum of the sentiments of its words.

2.2.6 Question 4d

Let's use our `punct_re` regular expression from the previous part to clean up the text a bit more! The goal here is to remove all of the punctuations to ensure words can be properly matched with those from VADER to actually calculate the full sentiment score.

Complete the following function `sanitize_texts` that takes in a table `df` and adds a new column `clean_text` by converting all characters in its original `full_text` column to lower case and replace all instances of punctuations with a space character.

```
[ ]: def sanitize_texts(df):
         df["clean_text"] = df["full_text"].str.replace(punct_re, " ",regex=True).
      ↪str.lower()
         return df

     tweets = {handle: sanitize_texts(df) for handle, df in tweets.items()}
     tweets["AOC"]["clean_text"].head()
```

```
[ ]: id
     1358149122264563712
     rt  repescobar  our country has the moral obligation and responsibility to
     reunite every single family separated at the southern border   t
     1358147616400408576
     rt  rokhanna  what happens when we guarantee  15 hour     31  of black workers
     and 26  of latinx workers get raises    a majority of essent
     1358145332316667909
     source  https   t co 3o5jer6zpd
     1358145218407759875                                                joe cunningham
     pledged to never take corporate pac money  and he never did  mace said she ll
     cash every check she gets  yet another way this is a downgrade  https   t co
     dytsqxkxgu
     1358144207333036040    what s even more gross is that mace takes corporate pac
     money   she s already funded by corporations  now she s choosing to swindle
     working people on top of it   peak scam artistry  caps for cash   https   t co
     ccvxgdf6id
     Name: clean_text, dtype: object
```

2.2.7 Question 4e

With the texts sanitized, we can now extract all the user mentions from tweets.

Complete the following function `extract_mentions` that takes in the `full_text` (not `clean_text`!) column from a tweets dataframe and uses `mentions_re` to extract all the mentions in a dataframe. The returned dataframe is: * single-indexed by the IDs of the tweets * has one row for each mention * has one column named `mentions`, which contains each mention in all lower-cased characters

Hint: There are several ways to approach this problem. Here is documentation for potentially useful functions: `str.extractall` (link) and `str.findall` (link), `dropna` (link), and `explode` (link).

```
[ ]: def extract_mentions(full_texts):
         mentions = pd.DataFrame({"mentions": full_texts.str.findall(mentions_re)})
```

```
    mentions = mentions.explode("mentions")
    mentions["mentions"] = mentions["mentions"].str.lower()
    mentions.dropna(inplace=True)
    return mentions

# uncomment this line to help you debug
# display(extract_mentions(tweets["AOC"]["full_text"]).head())

# # do not modify the below code
mentions = {handle: extract_mentions(df["full_text"]) for handle, df in tweets.
 ↪items()}
horiz_concat_df(mentions).head()
```

[ ]:

	AOC	Cristiano	elonmusk
	mentions	mentions	mentions
0	@repescobar	@sixpadhomegym	@dumdin7
1	@rokhanna	@globe_soccer	@grimezsz
2	@jaketapper	@pestanacr7	@grimezsz
3	@repnancymace	@goldenfootofficial	@kristennetten
4	@aoc	@herbalife	@kristennetten

2.2.8 Tidying Up the Data

Now, let's convert the tweets into what's called a *tidy format* to make the sentiments easier to calculate. The `to_tidy_format` function implemented for you uses the `clean_text` column of each tweets dataframe to create a tidy table, which is:

- single-indexed by the IDs of the tweets, for every word in the tweet.
- has one column named `word`, which contains the individual words of each tweet.

Run the following cell to convert the table into the tidy format. Take a look at the first 5 rows from the "tidied" tweets dataframe for AOC and see if you can find out how the structure has changed.

Note: Although there is no work needed on your part, we have referenced a few more advanced pandas methods you might have not seen before – you should definitely look them up in the documentation when you have a chance, as they are quite powerful in restructuring a dataframe into a useful intermediate state!

[ ]:

```
# just run this cell
def to_tidy_format(df):
    tidy = (
        df["clean_text"]
        .str.split()
        .explode()
        .to_frame()
        .rename(columns={"clean_text": "word"})
    )
    return tidy
```

```
tidy_tweets = {handle: to_tidy_format(df) for handle, df in tweets.items()}
tidy_tweets["AOC"].head()
```

```
[ ]:                            word
id
1358149122264563712           rt
1358149122264563712   repescobar
1358149122264563712          our
1358149122264563712      country
1358149122264563712          has
```

2.2.9 Adding in the Polarity Score

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.

The following `add_polarity` function adds a new `polarity` column to the `df` table. The `polarity` column contains the sum of the sentiment polarity of each word in the text of the tweet.

Note: Again, though there is no work needed on your part, it is important for you to go through how we set up this method and actually understand what each method is doing. In particular, see how we deal with missing data.

```
[ ]: # just run this cell
def add_polarity(df, tidy_df):
    df["polarity"] = (
        tidy_df
        .merge(sent, how='left', left_on='word', right_index=True)
        .reset_index()
        .loc[:, ['id', 'polarity']]
        .fillna(0)
        .groupby('id')
        .sum()
    )
    return df

tweets = {handle: add_polarity(df, tidy_df) for (handle, df), tidy_df in \
          zip(tweets.items(), tidy_tweets.values())}
tweets["AOC"][["clean_text", "polarity"]].head()
```

```
[ ]: clean_text  \
id
1358149122264563712
rt  repescobar  our country has the moral obligation and responsibility to
reunite every single family separated at the southern border   t
1358147616400408576
rt  rokhanna  what happens when we guarantee  15 hour     31  of black workers
and 26  of latinx workers get raises    a majority of essent
1358145332316667909
```

```
source  https   t co 3o5jer6zpd
1358145218407759875                                          joe cunningham
pledged to never take corporate pac money  and he never did  mace said she ll
cash every check she gets  yet another way this is a downgrade  https   t co
dytsqxkxgu
1358144207333036040  what s even more gross is that mace takes corporate pac
money   she s already funded by corporations  now she s choosing to swindle
working people on top of it   peak scam artistry  caps for cash   https   t co
ccvxgdf6id

                     polarity
id
1358149122264563712       0.0
1358147616400408576       1.0
1358145332316667909       0.0
1358145218407759875       0.0
1358144207333036040      -6.4
```

2.2.10 Question 4f

Finally, with our polarity column in place, we can finally explore how the sentiment of each tweet relates to the user(s) mentioned in it.

Complete the following function `mention_polarity` that takes in a mentions dataframe `mentions` and the original tweets dataframe `df` and returns a series where the mentioned users are the index and the corresponding mean sentiment scores of the tweets mentioning them are the values.

Hint: You should consider joining tables together in this question.

[ ]:
```python
def mention_polarity(df, mention_df):
    temp = df.merge(mention_df, how='left', left_index=True, right_index=True).
 ↪set_index("mentions")

    return temp.groupby("mentions")["polarity"].mean()


aoc_mention_polarity = mention_polarity(tweets["AOC"],mentions["AOC"]).
 ↪sort_values(ascending=False)
aoc_mention_polarity
```

[ ]:
```
mentions
@booker4ky          15.4
@texasaflcio        12.8
@davidscottjaffe    12.6
@teamwarren         12.6
@padmalakshmi       12.3
                    …
```

```
@meggiebaer         -8.6
@manhattanda       -10.8
@scotthech         -10.8
@repmarktakano     -10.8
@repchuygarcia     -10.8
Name: polarity, Length: 1182, dtype: float64
```

2.2.11 Question 4g

When grouping by mentions and aggregating the polarity of the tweets, what aggregation function should we use? What might be one drawback of using the mean?

The method of aggregation chosen will depend on the distribution of the data. A good aggregation function that we can choose is the median as it is not skewed by outliers. We are also unsure if the data is symmetrically distributed and thus the mean might not be the best.

As the distribution is not yet plotted, we cannot be sure that there are no outliers that might skew the mean. This could cause the mean to not reflect the true center of the distribution.

2.3 Question 5: You Do EDA!

Congratulations! You have finished all of the preliminary analysis on AOC, Cristiano, and Elon Musk's recent tweets.

As you might have recognized, there is still far more to explore within the data and build upon what we have uncovered so far. In this open-ended question, we want you to come up with a new perspective that can expand upon our analysis of the sentiment of each tweet.

For this question, you will perform some text analysis on our `tweets` dataset. Your analysis should have two parts:

1. a piece of code that manipulates `tweets` in some way and produces informative output (e.g. a dataframe, series, or plot)
2. a short (4-5 sentence) description of the findings of your analysis: what were you looking for? What did you find? How did you go about answering your question?

Your work should involve text analysis in some way, whether that's using regular expressions or some other form.

To assist you in getting started, here are a few ideas for this you can analyze for this question:

- dig deeper into when devices were used
- how sentiment varies with time of tweet
- expand on regexes from 4b to perform additional analysis (e.g. hashtags)
- examine sentiment of tweets over time

In general, try to combine the analyses from earlier questions or create new analysis based on the scaffolding we have provided.

This question is worth 4 points and will be graded based on this rubric:

	2 points	1 point	0 points
Code	Produces a mostly informative plot or pandas output that addresses the question posed in the student's description and uses at least one of the following pandas DataFrame/Series methods: `groupby`, `agg`, `merge`, `pivot_table`, `str`, `apply`	Attempts to produce a plot or manipulate data but the output is unrelated to the proposed question, or doesn't utilize at least one of the listed methods	No attempt at writing code
Description	Describes the analysis question and procedure comprehensively and summarizes results correctly	Attempts to describe analysis and results but description of results is incorrect or analysis of results is disconnected from the student's original question	No attempt at writing a description

---

2.3.1 Question 5a

Use this space to put your EDA code.

let us explore how the sentiment of the tweets from the 3 different users.

We first exttract the tweets from

```
[ ]: hashtag_re = r"#\w+"
tweets["AOC"].full_text.str.findall(hashtag_re).groupby(level=0).sum().
 ↪value_counts().head(6)
```

```
[ ]: full_text
[]                    3061
[#GreenNewDeal]         18
[#MedicareForAll]       12
[#TeamAOC]               8
[#coronavirus]           5
[#COVID19]               4
Name: count, dtype: int64
```

It seems like most tweets do not contain tags as seen from how many of the tweets conatain an empty array. We want to explore if there exists tweets that contain more than 1 hashtag.

[ ]:
```python
temp = tweets["AOC"].full_text.str.findall(hashtag_re)
first_index = temp.apply(lambda x: len(x)).sort_values(ascending=False).index[0]
print(f"There are at most {len(temp.loc[first_index])} items in a row")
print(f"The items are {temp.loc[first_index]}")
```

```
There are at most 3 items in a row
The items are ['#COVID', '#Patient31', '#socialdi']
```

Now we see that one tweet can contain multiple tweets. lets use the function `explode` to get each hashtag into a row with its corresponding id.

We then count the hashtags again now by its name to see if there are any changes to the ranking.

We also drop the NA rows so that we can merge the dataframe back later.

[ ]:
```python
hashtags = temp.explode().dropna()
hashtags.groupby(level=0).sum().value_counts().head(5)

hashtags = hashtags.to_frame()
hashtags.head(2)
```

[ ]:
```
                     full_text
id
1355363792545263617  #COVID19
1354577938767818764   #inners
```

It seems that there are not changes to top few hashtags and infact it is surprising to see so little tags for the dataset of tweets which contains thousands rows.

[ ]:
```python
hashtags.columns = ["hashtag"]
hashtags_tweet = hashtags.merge(tweets["AOC"], how='left', left_index=True,␣
 ↪right_index=True)
aoc_hashtag = hashtags_tweet.groupby("hashtag")["polarity"].mean().
 ↪sort_values(ascending=False)
```

[ ]:
```python
import plotly.express as px
fig1 = px.bar(aoc_hashtag, title="Average Polarity of Tweets by Hashtag",␣
 ↪labels={"value": "Average Polarity"})
fig1.show()

fig2 = px.bar(hashtags_tweet.value_counts("hashtag"), title="Count of Tweets by␣
 ↪Hashtag", labels={"value": "Count"})
fig2.show()
```

[ ]:
```python
temp = tweets["elonmusk"].full_text.str.findall(hashtag_re)
hashtags = temp.explode().dropna()
hashtags.groupby(level=0).sum().value_counts().head(5)
hashtags = hashtags.to_frame()
hashtags.columns = ["hashtag"]
```

```
hashtags_tweet = hashtags.merge(tweets["elonmusk"], how='left',␣
 ↪left_index=True, right_index=True)
elonmusk_hashtag = hashtags_tweet.groupby("hashtag")["polarity"].mean().
 ↪sort_values(ascending=False)

fig1 = px.bar(elonmusk_hashtag, title="Average Polarity of Tweets by Hashtag",␣
 ↪labels={"value": "Average Polarity"})
fig1.show()

fig2 = px.bar(hashtags_tweet.value_counts("hashtag"), title="Count of Tweets by␣
 ↪Hashtag", labels={"value": "Count"})
fig2.show()
```

Let us also explore how the sentiment of the tweet changes over tiime. We use the polarity found previously and we group them to find out the average sentiment of each bin and plot of graph of it

[ ]:
```
hour_bins = np.arange(0, 24.5, .5)
tweets["elonmusk"]['binned_hour'] = pd.cut(tweets["elonmusk"].converted_hour,␣
 ↪hour_bins, labels=hour_bins[:-1])
tweets["elonmusk"].head(2)
```

[ ]:
```
                                     created_at               id_str  \
id
1357991946082418690 2021-02-06 09:58:04+00:00  1357991946082418688
1357973565413367808 2021-02-06 08:45:02+00:00  1357973565413367808

                                                                  full_text  \
id
1357991946082418690        The Second Last Kingdom https://t.co/Je4EI88HmV
1357973565413367808  @DumDin7 @Grimezsz Haven't heard that name in years …

                     truncated display_text_range  \
id
1357991946082418690      False            [0, 23]
1357973565413367808      False           [19, 53]

                                                    entities  \
id
1357991946082418690  {'hashtags': [], 'symbols': [], 'user_mentions': [],
'urls': [], 'media': [{'id': 1357991942471094275, 'id_str':
'1357991942471094275', 'indices': [24, 47], 'media_url':
'http://pbs.twimg.com/media/EtiOegrVEAMCgZE.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtiOegrV…
1357973565413367808  {'hashtags': [], 'symbols': [], 'user_mentions':
[{'screen_name': 'DumDin7', 'name': 'Dum Din', 'id': 1279896279733145601,
'id_str': '1279896279733145601', 'indices': [0, 8]}, {'screen_name': 'Grimezsz',
'name': '         ', 'id': 276540738, 'id_str': '276540738', 'indi…
```

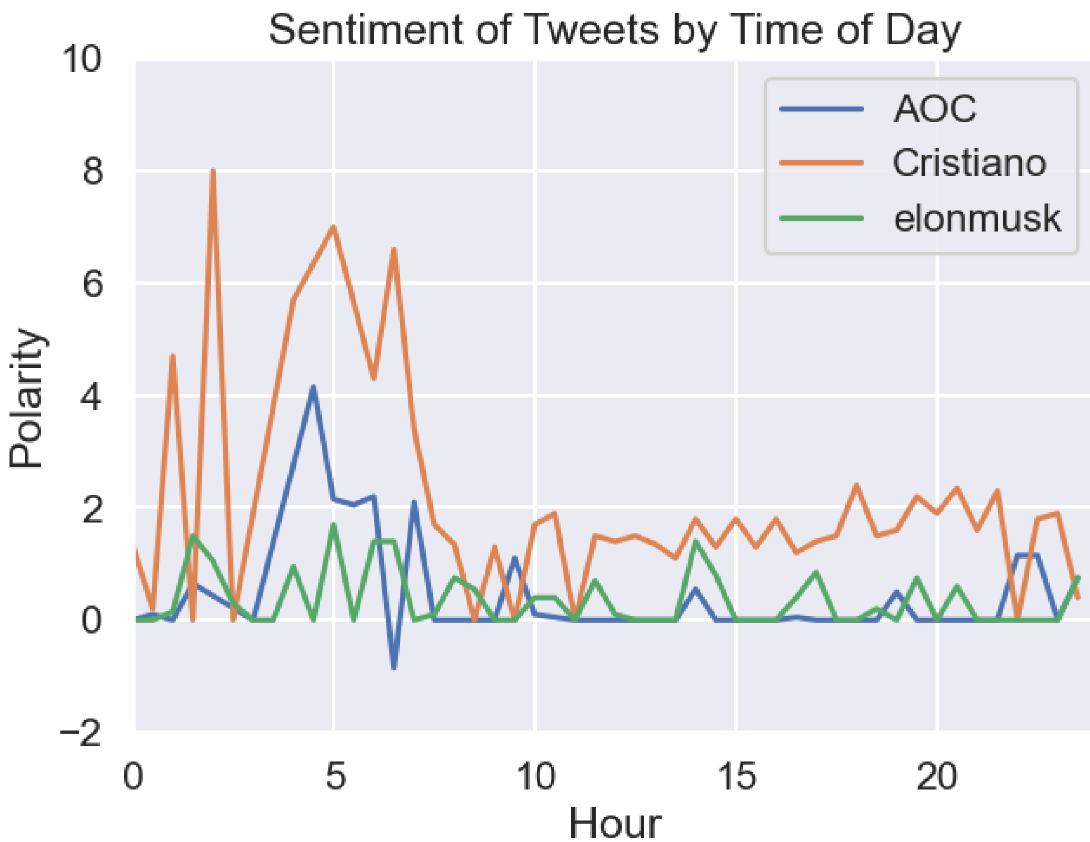
```
                                            extended_entities  \
id
1357991946082418690  {'media': [{'id': 1357991942471094275, 'id_str':
'1357991942471094275', 'indices': [24, 47], 'media_url':
'http://pbs.twimg.com/media/EtiOegrVEAMCgZE.jpg', 'media_url_https':
'https://pbs.twimg.com/media/EtiOegrVEAMCgZE.jpg', 'url':
'https://t.co/Je4EI88HmV', 'display_url': '…
1357973565413367808
NaN

                 source  \
id
1357991946082418690  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>
1357973565413367808  <a href="http://twitter.com/download/iphone"
rel="nofollow">Twitter for iPhone</a>

                     in_reply_to_status_id  in_reply_to_status_id_str  …  \
id                                                                     …
1357991946082418690                    NaN                        NaN  …
1357973565413367808           1.357973e+18               1.357973e+18  …

                     quoted_status_id_str  quoted_status_permalink  \
id
1357991946082418690                   NaN                      NaN
1357973565413367808                   NaN                      NaN

                    quoted_status              device      hour  \
id
1357991946082418690           NaN  Twitter for iPhone  9.967778
1357973565413367808           NaN  Twitter for iPhone  8.750556

                               converted_time  converted_hour  \
id
1357991946082418690 2021-02-06 01:58:04-08:00        1.967778
1357973565413367808 2021-02-06 00:45:02-08:00        0.750556

                                                                clean_text  \
id
1357991946082418690        the second last kingdom https   t co je4ei88hmv
1357973565413367808   dumdin7  grimezsz haven t heard that name in years

                     polarity  binned_hour
id
1357991946082418690       0.0          1.5
1357973565413367808       0.0          0.5
```

```
[2 rows x 37 columns]
```

[ ]:
```python
from traitlets import observe


hour_bins = np.arange(0, 24.5, .5)
for item in tweets:
    tweets[item]['binned_hour'] = pd.cut(tweets[item].converted_hour,␣
 ↪hour_bins, labels=hour_bins[:-1])

for item in tweets:
    sns.lineplot(tweets[item].groupby(tweets[item]['binned_hour']).polarity.
 ↪median(), label=item)
plt.title(f'Sentiment of Tweets by Time of Day')
plt.xlabel("Hour")
plt.ylabel("Polarity")
plt.ylim(-2,10)
plt.xlim(0,24)
```

```
/var/folders/_m/glx_2hfd76x1hdgjm__cz8mw0000gn/T/ipykernel_11923/1877393207.py:9
: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.

/var/folders/_m/glx_2hfd76x1hdgjm__cz8mw0000gn/T/ipykernel_11923/1877393207.py:9
: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.

/var/folders/_m/glx_2hfd76x1hdgjm__cz8mw0000gn/T/ipykernel_11923/1877393207.py:9
: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
```

[ ]:
```
(0.0, 24.0)
```



2.3.2 Question 5b

Use this space to put your EDA description.

Overall it seems like the hashtags have an interesting mean polarity where we can see how hashtag issues are viewed by the public in general. This would be interesting to analyse along with the mentions. That is how the polarity changes for the same hashtag with a different mention.

We can also see how different users have different hashtag uses as well. For example, elonmusk uses a smaller varity of hashtags compared to AOC and that his average polarity across tweets is generally more positive than AOC.

Ploting the sentiment over time for the 3 groups we can see that the sentiment past midnight to early morning is higher than that of the hour after 8. This could be because people are working during this hours and thus are not having fun, while people tend to play past working hours which could explain the higher polarity.

2.4 Congratulations! You have finished Lab 3!