Lab 3: Text Analysis (20 Pts)

In [ ]:
```
# 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
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 visualizatio
    Each DataFrame must have the same columns.
    """
    df = pd.concat([df.reset_index(drop=True) for df in dict_of_df.values
    if head is None:
        return df
    return df.head(head)
```

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:

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

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`:

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

In [ ]:
```
with open("data/BernieSanders_recent_tweets.txt", "r") as f:
    tweets = f.read(1000)
```

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 (SONJ)
D. Excel XML

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

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

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.

In [ ]:
```
tweets = {
    "AOC": pd.read_json("data/AOC_recent_tweets.txt").set_index("id"),
    "Cristiano": pd.read_json("data/Cristiano_recent_tweets.txt").set_ind
    "elonmusk": pd.read_json("data/elonmusk_recent_tweets.txt").set_index
}
```

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).

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

Out[ ]:

	created_at	id_str	
id			
1357991946082418690	2021-02-06 09:58:04+00:00	1357991946082418688	The Second La https://t.co/J
1357973565413367808	2021-02-06 08:45:02+00:00	1357973565413367808	@DumDin7 @Grime heard that nam
1357972904663687173	2021-02-06 08:42:25+00:00	1357972904663687168	@Grimezs
1357970517165182979	2021-02-06 08:32:55+00:00	1357970517165182976	YOLT\n\nhttps://t.co/
1357964347813687296	2021-02-06 08:08:24+00:00	1357964347813687296	@Kristennetten Th

5 rows × 30 columns

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.

Gpt is used to find open

Social media marketing researcher will be likely to do data analysis on tweets.
Reason: They can analyze tweets to understand audience sentiment, trends, and engagement, often for brand management. It is also important for them to plan their ad targeting, and social media strategies.

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 :(

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

Out[ ]:

id	source
1358137564587319299	Twitter for iPhone
1357379984399212545	Twitter for iPhone
1356733030962987008	Twitter for iPhone
1355924395064233986	Twitter for iPhone
1355599316300292097	Twitter for iPhone
...	...
32514882561638401	WhoSay
32513604662071296	WhoSay
32511823722840064	WhoSay
32510294081146881	WhoSay
32508748819857410	WhoSay

3198 rows × 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`.

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.

In [ ]:

```
q2a_pattern = r'<[^>]+>'
test_str = '<a href="http://twitter.com/download/iphone" rel="nofollow">'
```

```
            'Twitter for iPhone</a>'
re.sub(q2a_pattern, "", test_str)
```

Out[ ]: 'Twitter for iPhone'

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']`.

In [ ]:
```
#gpt is used to learn more about the re.findall()

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

Out[ ]: ['Twitter for iPhone']

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).

In [ ]:
```
#gpt is used to learn str.replace and str.extract

for username,df in tweets.items():
    df.loc[:, "device"] = df["source"].str.replace(q2a_pattern, '',
                                                   regex=True)

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

Out[ ]:

	created_at	id_str	full_
id			
1358149122264563712	2021-02-06 20:22:38+00:00	1358149122264563712	RT @RepEscobar: country has the n obligation responsibility to reu every single fa separated a southern border.\n
1358147616400408576	2021-02-06 20:16:39+00:00	1358147616400408576	RT @RoKhanna: W happens whe guarantee $15/hour' 💰 31% of Black wor and 26% of La workers get raises. A majority of esse
1358145332316667909	2021-02-06 20:07:35+00:00	1358145332316667904	(Sou https://t.co/3o5JEr6
1358145218407759875	2021-02-06 20:07:07+00:00	1358145218407759872	Joe Cunning pledged to never corporate PAC mo and he never did. M said she'll cash e check she gets another way this downgr https://t.co/DytsQXK
1358144207333036040	2021-02-06 20:03:06+00:00	1358144207333036032	What's even more g is that Mace t corporate money.\n\nShe's alre funded by corporat Now she's choosir swindle working pe on top of it.\n\nI scam artistry. Cap cas https://t.co/CcVxgD

5 rows × 31 columns

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.

In [ ]:
```python
def most_freq(series, k = 5):
    return series.value_counts().head(k)

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

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

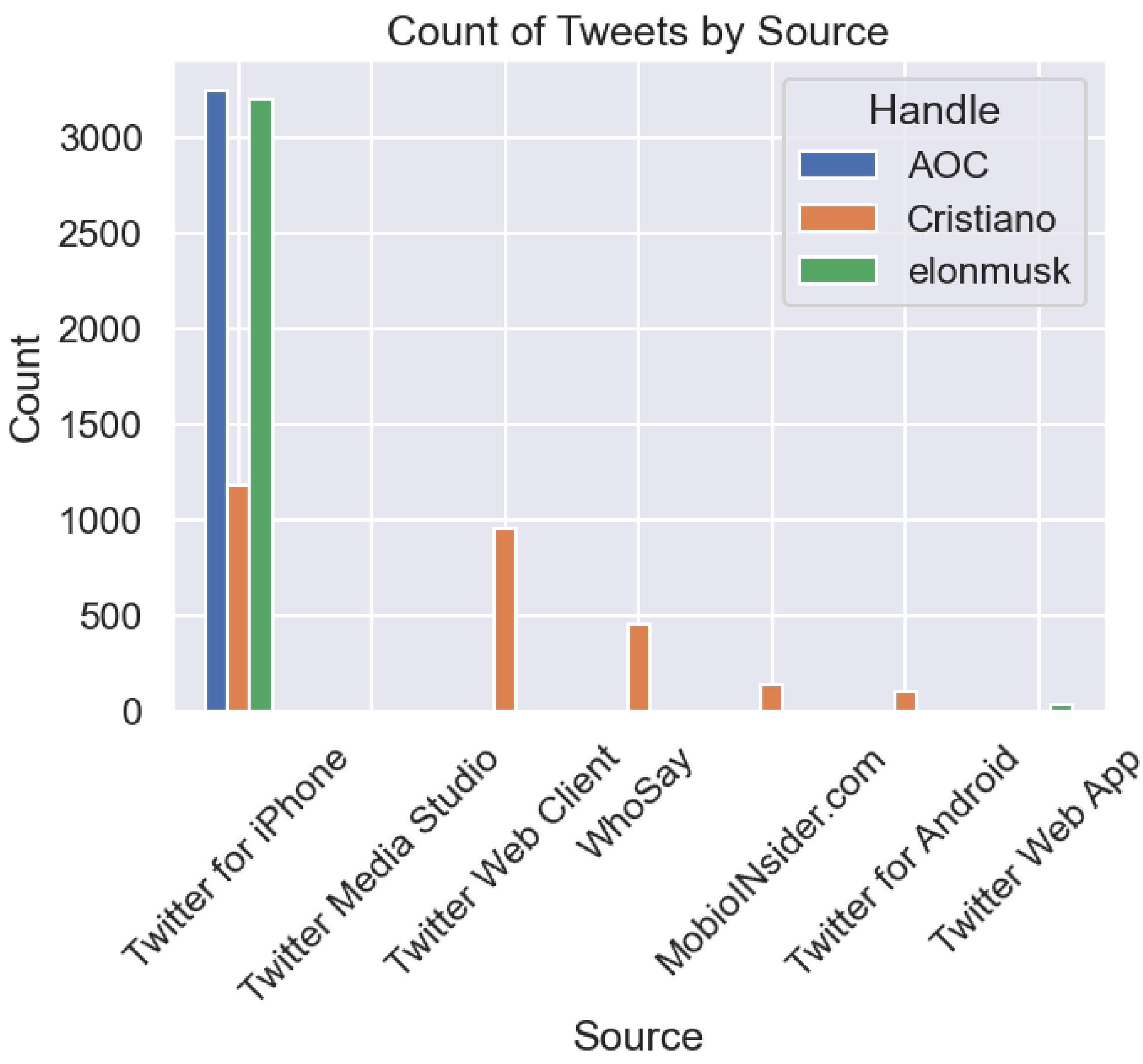
Run the following two cells to compute a table and plot describing the top 5 most commonly used devices for each user.

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

Out[ ]:

device	Twitter for iPhone	Twitter Media Studio	Twitter Web Client	WhoSay	MobioINsider.com	Twitter for Android	Twitter Web App
AOC	3245.0	2.0	0.0	0.0	0.0	0.0	0.0
Cristiano	1183.0	0.0	959.0	453.0	144.0	108.0	0.0
elonmusk	3202.0	0.0	0.0	0.0	0.0	0.0	37.0

In [ ]:
```python
# 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");
```



Question 2e

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

We can investigate the topic realated to each tweets by accounts "AOC", "elonmusk", and "Cristiano". We can also investigate the topic realated from each device posted by "Cristiano" account

---

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?

Comparing these distribution by camparing proportions of tweets can shows the device preference of each users, regardless of their tweets count. This might be able to shows the usage patterns and minimizes the impact of outliers.

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.

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.

In [ ]:

```
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 / pow(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()
```

Out[ ]:
```
id
1358149122264563712    20.377222
1358147616400408576    20.277500
1358145332316667909    20.126389
1358145218407759875    20.118611
1358144207333036040    20.051667
Name: hour, dtype: float64
```

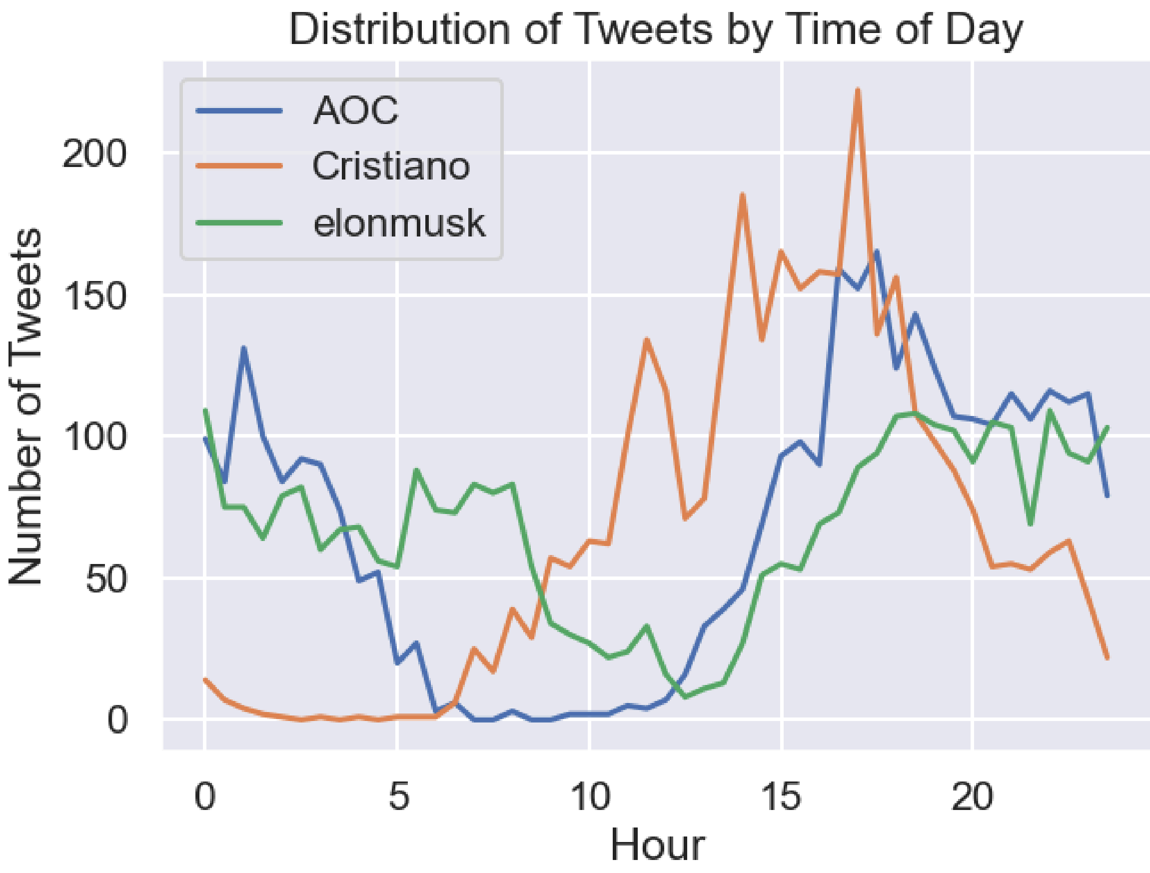
With our new `hour` column, let's take a look at the distribution of tweets for each user by time of day. The following cell helps create a density plot on the number of tweets based on the hour they are posted.

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.

In [ ]:
```
# just run this cell
def bin_df(df, bins, colname):
    binned = (pd.cut(df[colname],bins=bins,include_lowest=True)
              .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 t

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()
```

Out[ ]: <matplotlib.legend.Legend at 0x33e8e5e20>



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?

Cristiano has higher number of tweet post after the hour 6, compared to elonmusk and AOC. This might be because of the different timezone

---

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.

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

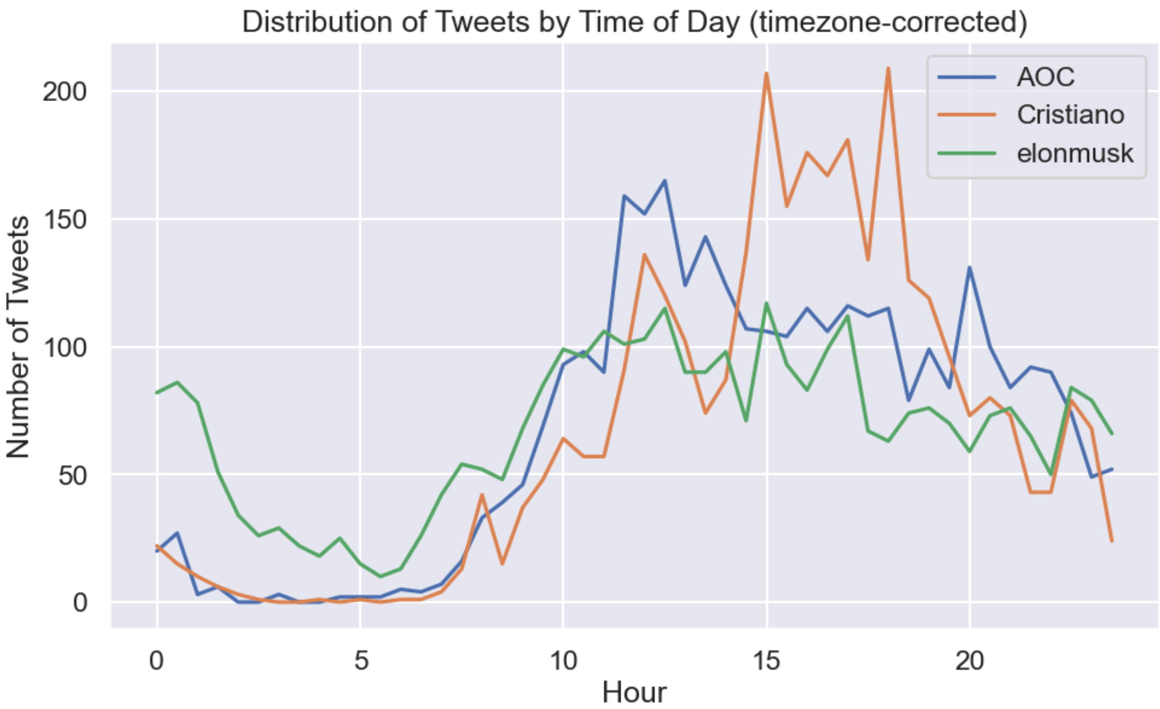
```
timezones = {"AOC": "EST", "Cristiano": "Europe/Lisbon", "elonmusk": "Ame

tweets = {handle: convert_timezone(df, tz) for (handle, df), tz in zip(tw
```

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.

In [ ]:

```
# just run this cell
def make_line_plot(df_dict, x_col, y_col, include=None, title=None, xlabe
    """
    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 p
        x_col: str
            the name of a column in each dataframe in `df_dict` to plot o
            the x-axis
        y_col: str
            the name of a column in each dataframe in `df_dict` to plot o
            the y-axis
        include: list[str], optional
            a list of handles to include in the plot; all keys in `df_dic
            present in `include`, if specified, will *not* be included in
        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
        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 ha
binned_hours = {handle: bin_df(df, hour_bins, "converted_hour") for handl
```

```
make_line_plot(binned_hours, "bin", "counts", title="Distribution of Twee
                xlabel="Hour", ylabel="Number of Tweets")
```



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:

In [ ]:
```
# 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.

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.

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 : 0.5
- order : 0
- Democrat : 1
- Republican : 1
- gun : -1
- dog : 1
- technology : 1
- TikTok : 0.5
- security : 0.8
- face-mask : -0.5
- science : 1.5
- climate change : 0.5
- vaccine : -1

My prediction score for gun is -1 because gun word are usually used during negative event.

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`.

In [ ]:
```
sent = pd.read_csv("vader_lexicon.txt", sep='\t')
sent.columns = ['token', 'polarity', 'std_dev', 'raw_score']
sent.set_index('token', inplace=True)
sent.head()
```

Out[ ]:

token	polarity	std_dev	raw_score
%)	-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]
('}{')	1.6	0.66332	[1, 2, 2, 1, 1, 2, 2, 1, 3, 1]

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.

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

Out[ ]:
```
'RT  RepEscobar  Our country has the moral obligation and responsibility
to reunite every single family separated at the southern border \n\nT '
```

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).

In [ ]:
```
mentions_re = r'@(\w+)'

re.findall(mentions_re, tweets["AOC"].iloc[0]["full_text"])
```

Out[ ]:
```
['RepEscobar']
```

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.

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.

In [ ]:
```
def sanitize_texts(df):
    df["clean_text"] = (df["full_text"].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()
```

Out[ ]:

```
id
1358149122264563712
rt  repescobar  our country has the moral obligation and responsibility
to reunite every single family separated at the southern border \n\nt
1358147616400408576
rt  rokhanna  what happens when we guarantee  15 hour \n\n  31  of black
workers and 26  of latinx workers get raises \n  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 do
wngrade  https   t co dytsqxkxgu
1358144207333036040    what s even more gross is that mace takes corpora
te pac money \n\nshe s already funded by corporations  now she s choosin
g to swindle working people on top of it \n\npeak scam artistry  caps fo
r cash   https   t co ccvxgdf6id
Name: clean_text, dtype: object
```

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).

In [ ]:

```python
#gpt is used to learn reset_index()

def extract_mentions(full_texts):
    mentioned_user = full_texts.str.extractall(mentions_re)
    mentioned_user = mentioned_user.reset_index(level=1, drop=True)

    mentions = pd.DataFrame({"id":mentioned_user.index,
                             "mentions":mentioned_user[0]})
    return mentions[["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 t
horiz_concat_df(mentions).head()
```

	mentions
id	
1358149122264563712	RepEscobar
1358147616400408576	RoKhanna
1358130063963811840	jaketapper
1358130063963811840	RepNancyMace
1358130063963811840	AOC

Out[ ]:

	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

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!

In [ ]:

```
# 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()
```

Out[ ]:

	word
id	
1358149122264563712	rt
1358149122264563712	repescobar
1358149122264563712	our
1358149122264563712	country
1358149122264563712	has

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.

In [ ]:

```
# 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()
```

Out[ ]:

	clean_text	polarity
id		
1358149122264563712	rt repescobar our country has the moral obligation and responsibility to reunite every single family separated at the southern border \n\nt	0.0
1358147616400408576	rt rokhanna what happens when we guarantee 15 hour \n\n 31 of black workers and 26 of latinx workers get raises \n a majority of essent	1.0
1358145332316667909	source https t co 3o5jer6zpd	0.0
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	0.0
1358144207333036040	what s even more gross is that mace takes corporate pac money \n\nshe s already funded by corporations now she s choosing to swindle working people on top of it \n\npeak scam artistry caps for cash https t co ccvxgdf6id	-6.4

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.

In [ ]:

```
#gpt is used to merge the dataframes

def mention_polarity(df, mention_df):
    data = mention_df.merge(df[["polarity"]],  how='left',
                            left_index=True,
                            right_index=True)
    data = data.groupby("mentions").mean()
    return data["polarity"]

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

Out[ ]:
```
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
RepChuyGarcia      -10.8
RepMarkTakano      -10.8
Name: polarity, Length: 1188, dtype: float64
```

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?

gpt is used to learn what are the possible drawback of each aggregation function.

Several aggregation function that can be used includes, mean, median, and mode. Median could be suitble for aggregating the polarity of the tweets mitigates the influence of outliers.

The drawback for using mean is that it is sensitive the oulier value, which might skew the polarity to one side.

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:

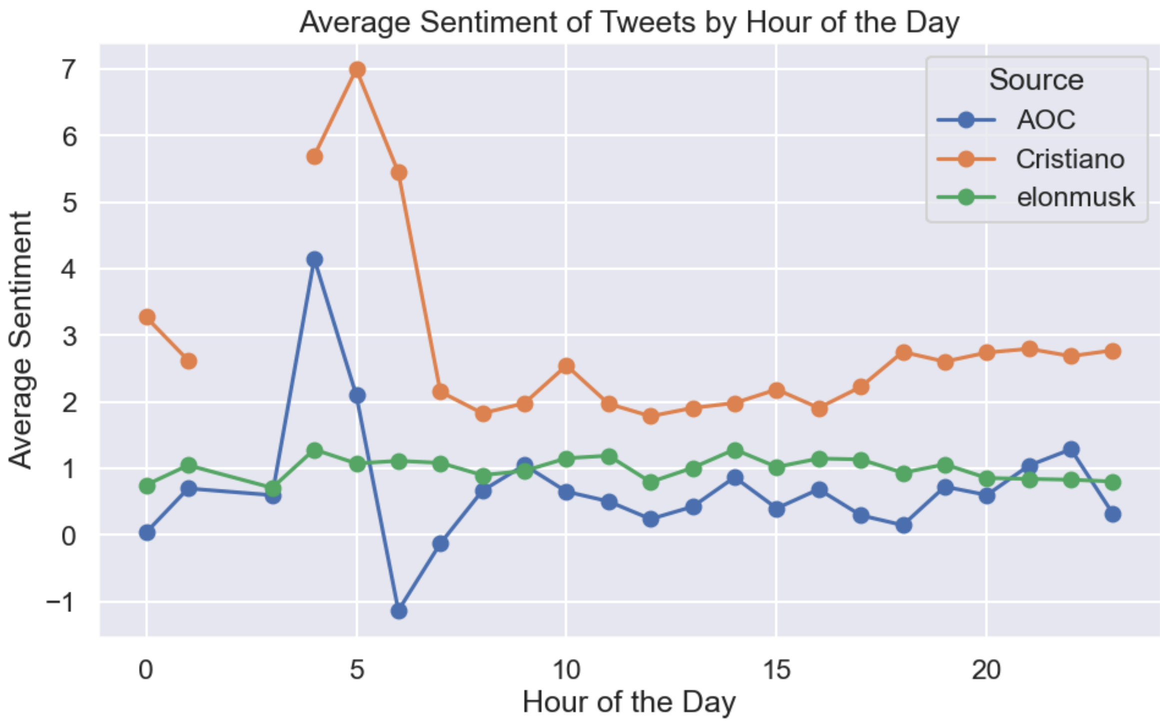
	2 points	1 point	0 points
Code	Produces a mostly informative plot or pandas output that addresses the question posed in the student's description and uses at least one of the following pandas DataFrame/Series methods: `groupby`, `agg`, `merge`, `pivot_table`, `str`, `apply`	Attempts to produce a plot 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

---

Question 5a

Use this space to put your EDA code.

In [ ]:
```
#gpt is usd to help debug the the code and find the possibility of the re

all_hourly_sentiment = pd.DataFrame()

for username, df in tweets.items():
    # copy the necessary columns
    tweets_data = df[["converted_time", "polarity"]].copy()

    #Create a new column with the 'converted_time' to hour of the day
    tweets_data["hour"] = tweets_data["converted_time"].dt.hour

    # Group by hour and calculate mean sentiment
    hourly_sentiment = tweets_data.groupby("hour")["polarity"].mean()

    # Add to the collected data with the dataframe name as a label
    all_hourly_sentiment[username] = hourly_sentiment

# Plot the data for each dataframe in the dictionary
plt.figure(figsize=(10, 6))

# Plot each sentiment line
for name in all_hourly_sentiment.columns:
    plt.plot(all_hourly_sentiment.index, all_hourly_sentiment[name],
             marker='o', label=name)

# Set plot title and labels
plt.title("Average Sentiment of Tweets by Hour of the Day")
plt.xlabel("Hour of the Day")
plt.ylabel("Average Sentiment")
plt.legend(title="Source")  # Legend to show which line belongs to which
plt.grid(True)
plt.show()
```



Question 5b

Use this space to put your EDA description.

The goal of this analysis is to examine the average sentiment of tweets by hour of the day for user AOC, Cristiano, and elonmusk. By plotting the average sentiment polarity of each user’s tweets across different hours. The plot average sentiment of each user is base on the mean of the calculated polarity agaisnt the hour of the day (converted time).

From the plot, we can notice that:

1. AOC user has huge flunctuation in the early hour of the day. AOC’s tweets in the early hours may carry more extreme or diverse sentiment, (GPT) suggest that it is related to the timing of significant political events or responses.

2. Cristiano has a more positive sentiment throughout the day, with a notable peak around hours 4 to 6. His sentiment is relatively high and stable, indicating a predominantly positive tone across his tweets, this could be due to the motivational content he shares.

3. elonmusk has a more neutral sentiment throughout the day, with slight variations throughout the day. His sentiment is generally stable, (GPT) suggesting a balanced or business-oriented tone, possibly reflecting his focus on tech-related or neutral topics

Congratulations! You have finished Lab 3!