Assignment 10.1c: Capstone - Social Media Analytics

Learning Outcome Addressed

- Recognize best industry practices for writing and managing large programs.
- Write a medium-level program (300-500 lines) with some level of guidance.

1. Introduction

In this project, you will analyze a dataset that contains social media messages (tweets and facebook posts) made by United States politicians in 2013 over a two-day period. You will parse the labels and text of the messages to add additional data on hashtag use, message sentiment, and poster identification, which can be used to compare politicians across several vectors.

Note: this project uses the pandas library to perform data analysis. You will need to install several libraries and learn about pandas DataFrames to complete this project. You can get started by watching the Bonus video from Module 9 about External Libraries for Data Analysis.

In the first section, you'll organize the data by adding several additional columns to the dataframe, to include information on name, location, position, sentiment, and hashtags. In the second section, you'll do a deeper analysis on the data by doing different kinds of comparisons on the hashtags included in the messages.

Data source used

2. Instructions

In the first stage, you will organize the data for the project by parsing some of the text included in the original CSV into smaller data points, and adding new columns containing this new data to the DataFrame. This reformatting will make analysis easier in the following stage.

Note: to keep the starter file from becoming too crowded, all the tests have been moved to the file social_tests.py, which you can open and read while debugging. The functions in this file are called at the bottom of social.py. If you change the filename of social.py, you will need to change the filename imported by social_tests.py in the first line of the file too.

3. [10 pts] Parse Label

One of the columns in the dataset contains the message’s label, which describes the person who posted the message. We’ll use this column to gather information for four columns in the database: Name, State, Position, and Region.

Write a function **parse_label(label)** which takes a string (the label) and returns a dictionary mapping three keys to three pieces of information. The key “name” maps to the name of the poster; the key “position” maps to the position of the poster; and the key “state” maps to the state of the poster.

How can we derive this information? We’re guaranteed that the label takes the format:

"From: [Name] ([Position] from [State])"

So we can use the text that is guaranteed to come before/after each piece of information to isolate the text (using string methods). However, we have to be careful- any of these three pieces of information could be any length, and some may contain more than one word!

For example:
parseName("From: Steny Hoyer (Representative from Maryland)") ->
{ “name” : "Steny Hoyer", “position” : "Representative", “state” : “Maryland” }

4. [5 pts] Find Region

We can get the poster’s name, position, and state from the label, but to get their region we’ll need to take an extra step. Write a function **get_region_from_state(state_df, state)** that takes state_df, a DataFrame, and a string of a state name to search for, and returns the corresponding region. state_df has rows where each state is in the column “state” and its corresponding region of the US, like Northeast or South, is in the “region” column.

The pandas way to do this is to create a subset of the DataFrame containing only rows where the “state” column holds the state. Then you can index into the first (and only) row in the resulting DataFrame and return its “region” value.

Alternatively, you can iterate over each row in the DataFrame and check whether the state in the “state” column matches the provided state. Once you’ve found the row with the matching state, index into the “region” column and return the value at that position immediately.

5. [15 pts] Parse Message

We’ve finished analyzing the label, but we can still gather data from the posted message itself! Given a social media message, what hashtags are included inside that message? Write the message **find_hashtags(message)** which takes a string (a social media message) and returns a list of strings- the hashtags included in that message, in order of appearance.

To do this, you’ll need to find all the starting places of hashtags (designated by the ‘#’ character), and iterate over the characters that follow that starting index until you reach an ‘end’ character. We’ve provided a list of end characters in the global variable end_chars; just check for whether each character is in that list, and stop when you reach one of them.

For example:
find_hashtags(“I am so #excited to watch #TheMandalorian! #starwars”) -> ["#excited", "#TheMandalorian", “#starwars”]

6. [5 pts] Identify Message Sentiment

We can also analyze the message to determine it’s sentiment - whether the message is generally positive, negative, or neutral. Write the function **find_sentiment(classifier, message)** so that it returns a tuple containing the sentiment score of a message as well as a string descriptor of the sentiment.

First, to get the actual sentiment score of the message, run the method polarity_scores on the classifier object with the message as the argument. This method returns a dictionary with multiple results, but you only need to use one - index into the key “compound” to get the message score.

To get a descriptor, you’ll need to map the score to a string. The string should be “positive” if the classifier predicts the message is positive (greater than 0.1), “negative” if it is negative (less than -0.1), and “neutral” if it is neutral (between -0.1 and 0.1). If that score is less than -0.1, return "negative"; if greater than 0.1, return "positive"; and otherwise return "neutral".

For example, given the message "Had opportunity to participate in panel on importance of disaster mitigation", the function returns (0.0516, “neutral”)

Note: you might be curious about how the sentiment classifier works. Sentiment Analysis infers whether the statement has a positive, negative, or neutral connotation from the words in the sentence. This is a really hard problem. Think about the statement “That’s great!” You could use it to mean it really is great, or you can use it in response to something that is really horrible. The sentiment classifier is trained using thousands of example sentences in many different contexts in order to help it understand whether particular words or phrases are generally positive, negative, neutral (or could be used both positively and negatively). This classifier generates a number for the text given to it, where negative numbers are associated with negative sentiment, and positive numbers are associated with positive sentiment.

7. [15 pts] Add Columns to DataFrame

Finally, we need to take all the new information we can extract and add it to the DataFrame. Write a function **add_columns(data, state_df)** that takes in a dataframe data and a dataframe state_df, and destructively adds seven new columns to data based on the functions you wrote above. Return None at the end of the function.

To do this, you should iterate through each index and row in the DataFrame. Use the helper function parse_label on the “label” column to extract the poster’s name, position, and state. Then use get_region_from_state to map that state to a region. Then call find_hashtags and find_sentiment on the “text” column to get the message’s hashtag list and sentiment. You’ll need to add each of these data points to an individual list.

When you’re done collecting data, you can create the new columns by setting data[“col_name”] directly equal to the list of data points. This will destructively change the DataFrame to contain the new information. You should add the columns "name", "position", "state", and “region” from the label, and the columns "hashtags", "score", and “sentiment” from the text.

8. Instructions

Congratulations, you’re halfway there!

In this stage, you will dive deeper into actually analyzing the social media messages in the CSV. You'll be able to filter data by information about the poster and do different analyses on hashtags and sentiments to compare the positive and negative feelings on different topics between different politicians.

9. [10 pts] Filter Data for Sentiment Quantiles

First, we'll write a few functions to gather some basic but interesting data on the dataset. For example - what are the quantiles for the sentiment scores across all the messages in the dataset?

Write the function **get_sentiment_quantiles(data, col_name, col_value)** which takes a DataFrame and two strings (a column name and targeted column value) and returns a list holding five numbers - the min score, 25% quantile, 50% quantile, 75% quantile, and max score across all sentiment scores that match the given column value.

First, you'll need to filter the data to get only the rows that match the given column value. Start by filtering the DataFrame down to just the rows where the value in the column col_name matches the value col_value. You can do this with a Boolean operation index, or by iterating over all rows and checking their values.

Once you've filtered the data down to just the rows you want to analyze, getting the min, max, and quantiles is easy! You can just use built-in pandas methods. Check out the documentation for min, max, and quantile, which can all be called directly on the "score" column. Gather the results from each of these function calls and organize them into a single one-dimensional list. (Note that you'll need to cast the result of the quantile function to a list to get the right type).

For example, get_sentiment_quantiles(df, "state", "Pennsylvania") returns [-0.9196, -0.1779, 0.1779, 0.7424, 0.9793].

This function should also handle one special case. If both col_name and col_value are empty strings (""), just run the analysis on the entire dataset instead of a filtered version.

10. [10 pts] Filter Data for Hashtags

We may also want to filter data to determine which hashtags were used by a specific subset of the population. This is trickier, though, as hashtags are held in lists; we'll need to combine those lists together to get a final result.

Write the function **get_hashtag_subset(data, col_name, col_value)** which takes a DataFrame and two strings (a column name and targeted column value) and returns a set of all the hashtags that occurred in the rows of the DataFrame that matched the given column value. To do this, filter the dataset (as was done in the previous function), then iterate over all rows in the filtered dataset and add all the hashtags from the list in the "hashtags" column to a result set.

For example, get_hashtag_subset(df, "name", "Mitch McConnell") returns { '#budget', '#Obamacare', '#ISIL', '#Senate', '#Sequester', '#StudentLoans', '#Kentucky', '#SavingCoalJobsAct' }

11. [10 pts] Get Hashtag Counts

The final three steps will build towards one central goal: determining what the most popular hashtags in the dataset are, and whether they are used in an overall positive or negative sentiment.

First, write the function **get_hashtag_rates(data)** which takes the dataframe and returns a dictionary mapping hashtags (strings) to counts of how many times they're used in the entire dataset (integers).

To do this, you should loop over the "hashtags" column, then iterate over the hashtags in the list (if there are any) to update the number of times each hashtag has been seen in the overall dictionary.

12. [10 pts] Find Most Common Hashtags

Next, write the function **most_common_hashtags(hashtags, count)** which takes the hashtag dictionary generated in the previous function and the number of top hashtags to find (integer) and returns a new dictionary mapping just the top-count hashtags to how often they each occur. For example, most_common_hashtags(hashtags, 5) would return a dictionary of five key-value pairs, the five most popular hashtags in the dataset.

One way to do this is to create an empty dictionary which will hold the known most-common hashtags. Then repeatedly search for the most common hashtag that is not already a key in that dictionary. Once you've gone through all the hashtags and found the one with the highest appearance rate, add it to the dictionary. Continue the process until the length of the dictionary is equal to count.

For example, using the rates determined from the main dataset, most_common_hashtags(df, 7) should return
{ “#Obamacare” : 61, “#IRS” : 26, “#RenewUI” : 21, “#jobs” : 20, “#Benghazi” : 20, “#ObamaCare” : 20, “#SOTU” : 20 }

13. [10 pts] Get a Hashtag's Sentiment Score

Finally, write the function **get_hashtag_sentiment(data, hashtag)** which takes the dataset and a hashtag (string) and returns an average sentiment score (float) for that hashtag.

You will need to iterate over the rows of the DataFrame to find every row that contains the given hashtag in its “hashtags” column. We could average the direct sentiment scores of these messages, but that would put a lot of emphasis on messages that are heavily negative / heavily positive vs. more neutral. Instead, let’s map the sentiment categories (from the row “sentiment”) to numbers.

For each of these, get its sentiment category. If it is "positive", that maps to 1; “negative” maps to -1; and “neutral” maps to 0. To get the average score, average all the individual hashtag-message scores together and divide them by the total number of messages that contained the hashtag.

If this number is positive and close to 1, the messages were generally very positive; if it is negative and close to -1, they were generally very negative. A score close to 0 might indicate a neutral hashtag, or one which has a mix of positive and negative sentiments.

14. Congratulations!

Congratulations - you’re done! Try entering in the name of your own state or senator to see what kinds of hashtags they used. You can also explore which hashtags are most popular and what their average sentiments are.

social.py

```python
import social_tests as test

### PHASE 1 ###

import pandas as pd
import nltk
nltk.download('vader_lexicon', quiet=True)
from nltk.sentiment.vader import SentimentIntensityAnalyzer

def parse_label(label):
    result = { }
    name_start = label.find(" ")
    open_paren = label.find("(")
    result["name"] = label[name_start:open_paren].strip()

    position_end = label.find(" from")
    result["position"] = label[open_paren+1:position_end].strip()

    state_start = position_end + len(" from")
    close_paren = label.find(")")
    result["state"] = label[state_start:close_paren].strip()

    return result

def get_region_from_state(state_df, state):
    row = state_df[state_df["state"] == state]
    return row.iloc[0]["region"]

end_chars = [ " ", "\n", "#", ".", ",", "?", "!", ":", ";", ")" ]
def find_hashtags(message):
    import string
    hashtag_list = [ ]
    for i in range(len(message)):
        if message[i] == "#":
            j = i+1
            while j < len(message) and message[j] not in end_chars:
                j = j + 1
            hashtag_list.append(message[i:j])
    return hashtag_list

def find_sentiment(classifier, message):
    score = classifier.polarity_scores(message)['compound']
    if score < -0.1:
        return (score, "negative")
    elif score > 0.1:
        return (score, "positive")
    else:
        return (score, "neutral")

def add_columns(data, state_df):
    classifier = SentimentIntensityAnalyzer()
    names, positions, states, regions = [], [], [], []
    for label in data["label"]:
        label_result = parse_label(label)
        names.append(label_result["name"])
        positions.append(label_result["position"])
        state = label_result["state"]
        states.append(state)
        regions.append(get_region_from_state(state_df, state))
    data["name"] = names
    data["position"] = positions
    data["state"] = states
    data["region"] = regions

    hashtags, scores, sentiments = [], [], []
    for text in data["text"]:
        hashtags.append(find_hashtags(text))
        (score, category) = find_sentiment(classifier, text)
        scores.append(score)
        sentiments.append(category)
    data["hashtags"] = hashtags
    data["score"] = scores
    data["sentiment"] = sentiments

### PHASE 2 ###

def get_sentiment_quantiles(data, col_name, col_value):
    if col_name != "":
        data = data[data[col_name] == col_value]

    result = [ data["score"].min() ]
    result.extend(list(round(data["score"].quantile([0.25, 0.5, 0.75]),5)))
    result.append(data["score"].max())
    return result

def get_hashtag_subset(data, col_name, col_value):
    data = data[data[col_name] == col_value]
    all_hashtags = set()
```

```python
    for hashtags in data["hashtags"]:
        for tag in hashtags:
            all_hashtags.add(tag)
    return all_hashtags

def get_hashtag_rates(data):
    d = { }
    for hashtags in data["hashtags"]:
        for tag in hashtags:
            if tag not in d:
                d[tag] = 0
            d[tag] += 1
    return d

def most_common_hashtags(hashtags, count):
    best_only = { }
    while len(best_only) < count:
        curr_best = None
        curr_count = 0
        for k in hashtags:
            if hashtags[k] > curr_count and k not in best_only:
                curr_best = k
                curr_count = hashtags[k]
        best_only[curr_best] = curr_count
    return best_only

def get_hashtag_sentiment(data, hashtag):
    total = 0
    count = 0
    for index, row in data.iterrows():
        hashtags = row['hashtags']
        sent = row['sentiment']
        if hashtag in hashtags:
            count += 1
            if sent == 'positive':
                total += 1
            elif sent == 'negative':
                total -= 1
    return total / count

### RUN CODE ###
# This code runs the test cases to check your work
if __name__ == "__main__":
    test.test_all()
    test.run()
```

social_tests.py

```
import sys
sys.path.append("code/")
from social import *

def test_parse_label():
    print("Testing parse_label()...", end="")
    assert(parse_label("From: Steny Hoyer (Representative from Maryland)") == {
"name" : "Steny Hoyer", "position" : "Representative", "state" : "Maryland" })
    assert(parse_label("From: Mitch (Senator from Kentucky)") == { "name" :
"Mitch", "position" : "Senator", "state" : "Kentucky" })
    assert(parse_label("From: Heidi Heitkamp (Senator from North Dakota)") == {
"name" : "Heidi Heitkamp", "position" : "Senator", "state" : "North Dakota" })
    assert(parse_label("From: Chris Collins (Representative from New York)") == {
"name" : "Chris Collins", "position" : "Representative", "state" : "New York" })
    assert(parse_label("From: Kelly (Professor from PA)") == { "name" : "Kelly",
"position" : "Professor", "state" : "PA" })
    print("... done!")

def test_get_region_from_state():
    print("Testing get_region_from_state()...", end="")
    state_df = pd.read_csv("code/data/statemappings.csv")
    assert(str(get_region_from_state(state_df, "California")) == "West")
    assert(str(get_region_from_state(state_df, "Maine")) == "Northeast")
    assert(str(get_region_from_state(state_df, "Nebraska")) == "Midwest")
    assert(str(get_region_from_state(state_df, "Texas")) == "South")
    print("... done!")

def test_find_hashtags():
    print("Testing find_hashtags()...", end="")
    assert(find_hashtags("I am so #excited to watch #TheMandalorian! #starwars")
== [ "#excited", "#TheMandalorian", "#starwars" ])
    assert(find_hashtags("#CMUCarnival will be amazing as long as it doesn't rain
#weatherchannel") == [ "#CMUCarnival", "#weatherchannel" ])
    assert(find_hashtags("#Whatif, #everything #is: #hashtags?") ==  [ "#Whatif",
"#everything", "#is", "#hashtags" ])
    assert(find_hashtags("I don't like hashtags, I think they're overused") == [
])
    assert(find_hashtags("So excited for #registration!Let's go CMU!") == [
"#registration" ])
    assert(find_hashtags("I'm nervous-#registration but I think it should work
out") == [ "#registration" ])
```

```
    assert(find_hashtags("I'm waitlisted for everything #registration...") == [
"#registration" ])
    assert(find_hashtags("Not sure what to take #110#112") == [ "#110", "#112" ])
    assert(find_hashtags("Uh oh#") == ["#"])
    print("... done!")

def test_find_sentiment():
    print("Testing find_sentiment()...", end="")
    classifier = SentimentIntensityAnalyzer()
    assert(find_sentiment(classifier, "great") == (0.6249, "positive"))
    assert(find_sentiment(classifier, "bad") == (-0.5423, "negative"))
    assert(find_sentiment(classifier, "hello") == (0.0, "neutral"))
    assert(find_sentiment(classifier, "Being a senator means getting votes on
what you want and also having to take tough votes on what you don't") == (-
0.0516, "neutral"))
    assert(find_sentiment(classifier, "Had opportunity to participate in panel on
importance of disaster mitigation") == (0.0516, "neutral"))
    assert(find_sentiment(classifier, "If you're in the area tomorrow make sure
to stop by my office! I will be hold office hours from 3-4pm CST.") == (0.1007,
"positive"))
    assert(find_sentiment(classifier, "The House will pass a bill to pay federal
workers for their time in furlough once the shutdown ends.") == (-0.1027,
"negative"))
    print("...done!")

def test_add_columns():
    print("Testing add_columns()...", end="")
    df = pd.read_csv("code/data/politicaldata.csv")
    state_df = pd.read_csv("code/data/statemappings.csv")
    add_columns(df, state_df)
    assert(df["name"][1] == "Mitch McConnell")
    assert(df["name"][4] == "Mark Udall")
    assert(df["name"][4979] == "Ted Yoho")
    assert(df["position"][1] == "Senator")
    assert(df["position"][4] == "Senator")
    assert(df["position"][4979] == "Representative")
    assert(df["state"][1] == "Kentucky")
    assert(df["state"][4] == "Colorado")
    assert(df["state"][4979] == "Florida")
    assert(df["region"][1] == "South")
    assert(df["region"][4] == "West")
    assert(df["region"][4979] == "South")
    assert(df["hashtags"][1] == [ "#Obamacare" ])
    assert(df["hashtags"][4] == [ "#drones", "#innovation", "#privacy", "#UAS" ])
    assert(df["hashtags"][4979] == [ ])
    assert(df["sentiment"][0] == "neutral")
    assert(df["sentiment"][1] == "negative")
    assert(df["sentiment"][4978] == "positive")
    assert(df["score"][0] == 0.0)
    assert(df["score"][1] == -0.128)
    assert(df["score"][4978] == 0.3595)
    print("... done!")

def test_get_sentiment_quantiles(df):
    print("Testing get_sentiment_quantiles()...", end="")
    assert(get_sentiment_quantiles(df, "state", "Pennsylvania") == [-0.9196, -
0.1779, 0.1779, 0.7424, 0.9793])
    assert(get_sentiment_quantiles(df, "name", "Mitch McConnell") == [-0.5994, -
0.2484, 0.0, 0.35045, 0.9042])
    assert(get_sentiment_quantiles(df, "", "") == [-0.9852, 0.0, 0.3678, 0.7003,
0.9981])
    print("... done!")

def test_get_hashtag_subset(df):
    print("Testing get_hashtag_subset()...", end="")
    assert(get_hashtag_subset(df, "state", "Pennsylvania") == {
    '#SOTU', '#ABetterWay', '#PTA', '#WeAre', '#September11',
    '#Forestry', '#military', '#IRSscandal', '#Youth',
    '#whitenosesyndrome', '#Constitution', '#engineering',
    '#cropinsurance', '#Obamacare', '#NationalAdoptionDay', '#4Jobs',
    '#FF', '#EndTrafficking', '#floodinsurance', '#NoDealNoBreak',
    '#OAM2014chat', '#FathersDay', '#spellingbee', '#StopTheSequester',
    '#9', '#EarthDay', '#RateShock', '#MarcellusFest',
    '#corporatewelfare', '#Hezbollah', '#Agriculture',
    '#MarchOnWashington', '#RenewUI-', '#Ebola', '#Sellersville',
    '#manufacturing', '#shalegas', '#tcot', '#BorderCrisis', '#Benghazi',
    '#RestoreTrust', '#Iran', '#NIH', '#CombatSuicide', '#LaborDay',
    '#OpEd', '#House', '#DontDoubleMyRate', '#MedicareAdvantage',
    '#MOW50', '#30Days30Ways', '#MemorialDay', '#America',
    '#WashingtonMonument', '#TrainWreck', '#Vets', '#School', '#Delco',
    '#MentalHealth', '#earmark', '#weather-related', '#FTW', '#THON14',
    '#CBWest', '#BCTHS', '#Safety', '#business', '#humantrafficking',
    '#budget', '#MLKï¿½Ûªs', '#ErieCounty', '#ACA', '#ISIS',
    '#SenateMustAct', '#FortHood', '#SaveSarah', '#HR3717', '#FTK',
    '#PA', '#teens', '#USFS', '#SWPA', '#CoffeeWithKeith',
    '#CareerOneStop', '#IRS', '#DaNicaShirey', '#2013GC', '#energy',
    '#Waterford', "#PA8's", '#jobs', '#Traffic', '#LetsTalk',
    '#ACARepeal', '#USDA' })
    assert(get_hashtag_subset(df, "name", "Mitch McConnell") == {
    '#budget', '#Obamacare', '#ISIL', '#Senate', '#Sequester',
    '#StudentLoans', '#Kentucky', '#SavingCoalJobsAct' })
    assert(len(get_hashtag_subset(df, "region", "West")) == 470) # too long to
check all the hashtags here - just check the length instead
    print("... done!")

def test_get_hashtag_rates(df):
    print("Testing get_hashtag_rates()...", end="")
    d = get_hashtag_rates(df)
    assert(len(d) == 1529)
    assert(d["#TrainWreck"] == 8)
    assert(d["#jobs"] == 20)
    assert(d["#STEM"] == 5)
    assert(d["#ObamaCare"] == 20)
    print("... done!")

def test_most_common_hashtags(df):
    print("Testing most_common_hashtags()...", end="")
    d1 = { "#CMU" : 10, "#TheMandalorian" : 15, "#tgif" : 3, "#homework" : 20,
"#hashtag" : 1, "#programming" : 7, "#testcase" : 1, "#WorldPeace" : 9, "#coffee"
: 18, "#naptime" : 2 }
    assert(most_common_hashtags(d1, 1) == { "#homework" : 20 })
    assert(most_common_hashtags(d1, 2) == { "#homework" : 20, "#coffee" : 18 })
    assert(most_common_hashtags(d1, 5) == { "#homework" : 20, "#coffee" : 18,
"#TheMandalorian" : 15, "#CMU" : 10, "#WorldPeace" : 9 })

    d2 = get_hashtag_rates(df)
    assert(most_common_hashtags(d2, 1) == { "#Obamacare" : 61 })
    assert(most_common_hashtags(d2, 7) == { "#Obamacare" : 61, "#IRS" : 26,
"#RenewUI" : 21, "#jobs" : 20, "#Benghazi" : 20, "#ObamaCare" : 20, "#SOTU" : 20
})
    print("... done!")

def test_get_hashtag_sentiment(df):
    # Note - we're comparing floats here, so we'll check if they're
    # almost equal instead of exactly equal
    print("Testing get_hashtag_sentiment()...", end="")
    import math
    assert(math.isclose(get_hashtag_sentiment(df, "#TrainWreck"), -0.125))
    assert(math.isclose(get_hashtag_sentiment(df, "#jobs"), 0.7894736842105263))
    assert(math.isclose(get_hashtag_sentiment(df, "#STEM"), 0.6))
    assert(math.isclose(get_hashtag_sentiment(df, "#ObamaCare"), 0))
    print("... done!")

def test_all():
    test_parse_label()
    test_get_region_from_state()
    test_find_hashtags()
    test_find_sentiment()
    test_add_columns()

    df = pd.read_csv("code/data/politicaldata.csv")
    add_columns(df, pd.read_csv("code/data/statemappings.csv"))

    test_get_sentiment_quantiles(df)
    test_get_hashtag_subset(df)
    test_get_hashtag_rates(df)
    test_most_common_hashtags(df)
    test_get_hashtag_sentiment(df)

def run():
    print("\n-----\n")
    print("Now let's look at the general trends!")
    print("\n-----\n")

    df = pd.read_csv("code/data/politicaldata.csv")
    state_df = pd.read_csv("code/data/statemappings.csv")
    add_columns(df, state_df)

    print("Overall Sentiment Quantiles:")
    print(get_sentiment_quantiles(df, "", ""))
    print()

    hashtags = get_hashtag_rates(df)
    print("Total # Hashtags:")
    print(len(get_hashtag_rates(df)))
    print()

    freq_hashtags = most_common_hashtags(hashtags, 10)
    print("Top 10 Hashtags:")
    for hashtag in freq_hashtags:
        print(hashtag, "[", hashtags[hashtag], "uses, average score:",
get_hashtag_sentiment(df, hashtag), "]")
```