DATA301 Project Individual Progress Report

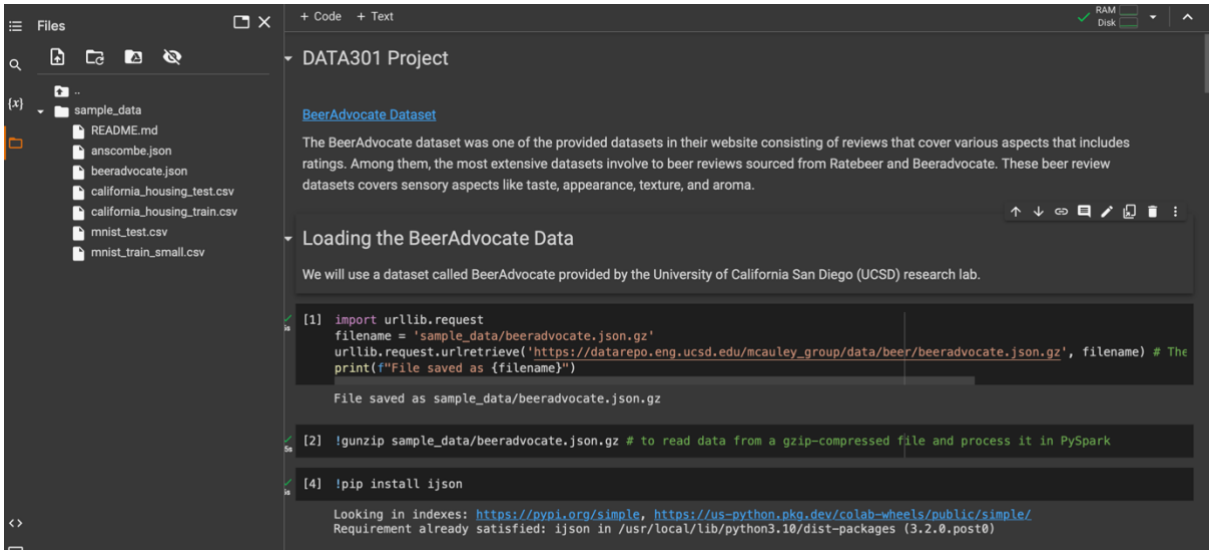
Student Name: Moses Bernard Velano
Student ID: 83396373

Start here.
The progression in my project focuses on working along with what I have planned on my Project Proposal. Some of the data that I have attempted was Amazon review data (2018) and Food.com Recipe & Review Data (McAuley, n.d.). But due to personal interest of knowing the best beer, I have decided to stay with what I have originally planned for.

Some tasks that I have worked on throughout Week 9 and 10 were conducting a research about the dataset that I am going to work on. For example, I have used the PDF document (Singh, 2015) and (McAuley et al., n.d.) to understand the variables in the BeerAdvocate dataset. When loading the BeerAdvocate dataset I have followed the process of loading a larger dataset from the Sample Project and the Labs that we have worked on for past few months. The second block of code:

```
!gunzip sample_data/beeradvocate.json.gz
```

This is helpful to decompress a ".json.gz" file, where it reads data from a gzip-compressed file and process it in PySpark as a normal ".json" file format. I was able to get help in this part by a fellow student in DATA301 class named Mike and from our Lecturer.

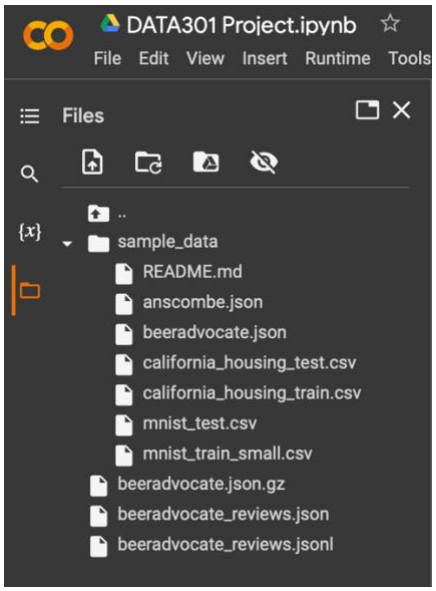


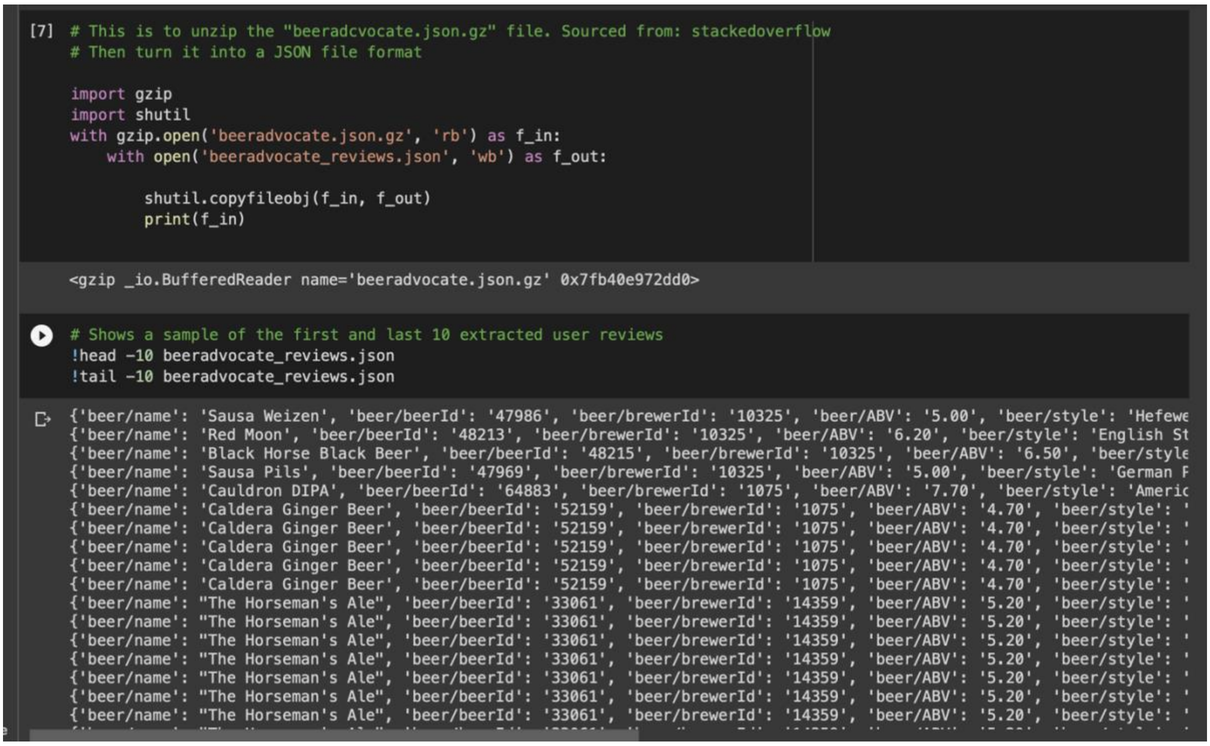
But then, when I was working along with a fellow student named Justin, he recommended me to look at the collab notebook documentation that was provided in the Amazon review data 2018 (Ni, 2019). This collab notebook showed a sample of a dataset that also had a file format of '.json.gz'.

```
[5] # Importing pyhton functions and modules. Sourced from: https://colab.research.google.com/drive/1Zv6MARGQcrBbLHyjPVVMZVr

    import os
    import json
    import gzip
    import pandas as pd
    from urllib.request import urlopen
```

```
[6] # Sourced from: https://colab.research.google.com/drive/1Zv6MARGQcrBbLHyjPVVMZVnRWsRnVMpV#scrollTo=feWoOrmt4Tja

    !wget https://datarepo.eng.ucsd.edu/sharknado/data/beer/beeradvocate.json.gz

    --2023-05-15 07:13:52--  https://datarepo.eng.ucsd.edu/sharknado/data/beer/beeradvocate.json.gz
    Resolving datarepo.eng.ucsd.edu (datarepo.eng.ucsd.edu)... 132.239.8.30
    Connecting to datarepo.eng.ucsd.edu (datarepo.eng.ucsd.edu)|132.239.8.30|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 453029619 (432M) [application/x-gzip]
    Saving to: 'beeradvocate.json.gz'

    beeradvocate.json.g 100%[===================>] 432.04M  41.5MB/s    in 9.8s

    2023-05-15 07:14:02 (43.9 MB/s) - 'beeradvocate.json.gz' saved [453029619/453029619]
```

And when I tried it on my own collab notebook it worked perfectly fine. Hence, I will be using this style of loading/downloading the BeerAdvocate dataset into my collab notebook. Afterwards, I proceeded to downloading and editing the name of the BeerAdvocate dataset inside the files of my collab notebook. This is what it looked like after running the first part of the code below.





```
[7] # This is to unzip the "beeradcvocate.json.gz" file. Sourced from: stackedoverflow
    # Then turn it into a JSON file format

    import gzip
    import shutil
    with gzip.open('beeradvocate.json.gz', 'rb') as f_in:
        with open('beeradvocate_reviews.json', 'wb') as f_out:

            shutil.copyfileobj(f_in, f_out)
            print(f_in)
```

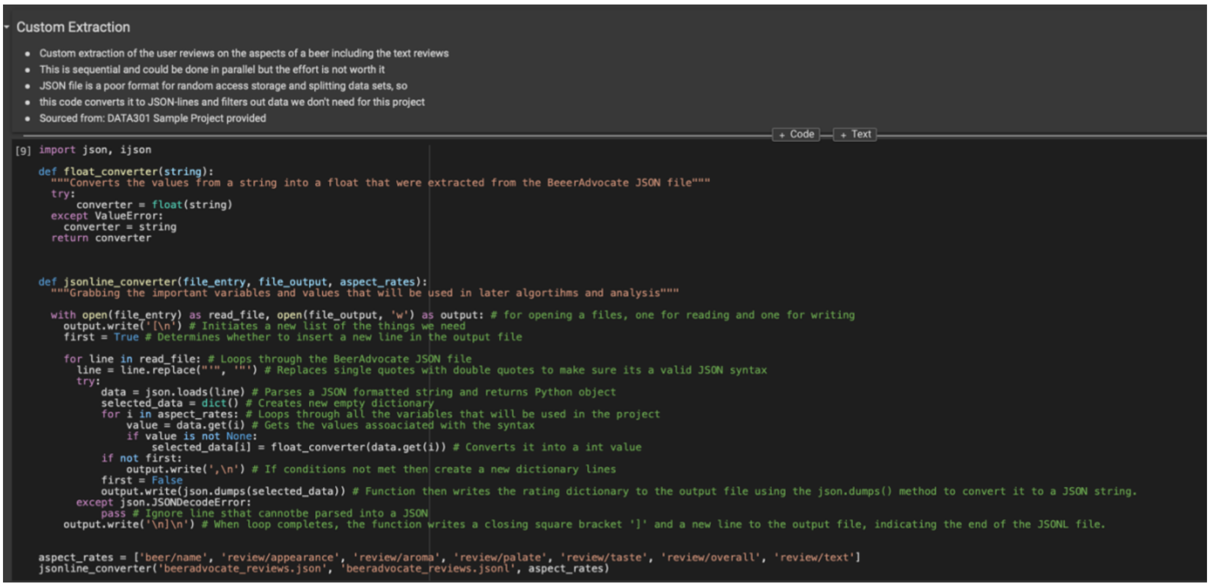
```
<gzip _io.BufferedReader name='beeradvocate.json.gz' 0x7fb40e972dd0>
```

```
# Shows a sample of the first and last 10 extracted user reviews
!head -10 beeradvocate_reviews.json
!tail -10 beeradvocate_reviews.json
```

```
{'beer/name': 'Sausa Weizen', 'beer/beerId': '47986', 'beer/brewerId': '10325', 'beer/ABV': '5.00', 'beer/style': 'Hefewe
{'beer/name': 'Red Moon', 'beer/beerId': '48213', 'beer/brewerId': '10325', 'beer/ABV': '6.20', 'beer/style': 'English St
{'beer/name': 'Black Horse Black Beer', 'beer/beerId': '48215', 'beer/brewerId': '10325', 'beer/ABV': '6.50', 'beer/style
{'beer/name': 'Sausa Pils', 'beer/beerId': '47969', 'beer/brewerId': '10325', 'beer/ABV': '5.00', 'beer/style': 'German P
{'beer/name': 'Cauldron DIPA', 'beer/beerId': '64883', 'beer/brewerId': '1075', 'beer/ABV': '7.70', 'beer/style': 'Americ
{'beer/name': 'Caldera Ginger Beer', 'beer/beerId': '52159', 'beer/brewerId': '1075', 'beer/ABV': '4.70', 'beer/style': '
{'beer/name': 'Caldera Ginger Beer', 'beer/beerId': '52159', 'beer/brewerId': '1075', 'beer/ABV': '4.70', 'beer/style': '
{'beer/name': 'Caldera Ginger Beer', 'beer/beerId': '52159', 'beer/brewerId': '1075', 'beer/ABV': '4.70', 'beer/style': '
{'beer/name': 'Caldera Ginger Beer', 'beer/beerId': '52159', 'beer/brewerId': '1075', 'beer/ABV': '4.70', 'beer/style': '
{'beer/name': 'Caldera Ginger Beer', 'beer/beerId': '52159', 'beer/brewerId': '1075', 'beer/ABV': '4.70', 'beer/style': '
{'beer/name': "The Horseman's Ale", 'beer/beerId': '33061', 'beer/brewerId': '14359', 'beer/ABV': '5.20', 'beer/style': '
{'beer/name': "The Horseman's Ale", 'beer/beerId': '33061', 'beer/brewerId': '14359', 'beer/ABV': '5.20', 'beer/style': '
{'beer/name': "The Horseman's Ale", 'beer/beerId': '33061', 'beer/brewerId': '14359', 'beer/ABV': '5.20', 'beer/style': '
{'beer/name': "The Horseman's Ale", 'beer/beerId': '33061', 'beer/brewerId': '14359', 'beer/ABV': '5.20', 'beer/style': '
{'beer/name': "The Horseman's Ale", 'beer/beerId': '33061', 'beer/brewerId': '14359', 'beer/ABV': '5.20', 'beer/style': '
{'beer/name': "The Horseman's Ale", 'beer/beerId': '33061', 'beer/brewerId': '14359', 'beer/ABV': '5.20', 'beer/style': '
{'beer/name': "The Horseman's Ale", 'beer/beerId': '33061', 'beer/brewerId': '14359', 'beer/ABV': '5.20', 'beer/style': '
```

The chunk of code above helps unzip a ‘gzip-compressed’ file and lets it rewrite it into any file format. In my case, I want it as a normal ‘json’ file format. This was sourced from Stackoverflow: ‘How to unzip gz file using Python’.
(https://stackoverflow.com/questions/31028815/how-to-unzip-gz-file-using-python)
The second chunk of code in that image visualizes what the dataset looks like in a JSON Lines text.

During Week 9, I started extracting all of the important variables that I will later use in my Project to meet the results I want. This includes extracting each of the user reviews on the aspects of a singular beer along with the text reviews. I have shared code and worked with Sudarshan and Daniel in this part of the project as we are all working on the same dataset.



Custom Extraction

- Custom extraction of the user reviews on the aspects of a beer including the text reviews
- This is sequential and could be done in parallel but the effort is not worth it
- JSON file is a poor format for random access storage and splitting data sets, so
- this code converts it to JSON-lines and filters out data we don't need for this project
- Sourced from: DATA301 Sample Project provided

```
[9] import json, ijson

    def float_converter(string):
      """Converts the values from a string into a float that were extracted from the BeeerAdvocate JSON file"""
      try:
          converter = float(string)
      except ValueError:
        converter = string
      return converter



    def jsonline_converter(file_entry, file_output, aspect_rates):
      """Grabbing the important variables and values that will be used in later algortihms and analysis"""

      with open(file_entry) as read_file, open(file_output, 'w') as output: # for opening a files, one for reading and one for writing
        output.write('[\n') # Initiates a new list of the things we need
        first = True # Determines whether to insert a new line in the output file

        for line in read_file: # Loops through the BeerAdvocate JSON file
          line = line.replace("'", '"') # Replaces single quotes with double quotes to make sure its a valid JSON syntax
          try:
              data = json.loads(line) # Parses a JSON formatted string and returns Python object
              selected_data = dict() # Creates new empty dictionary
              for i in aspect_rates: # Loops through all the variables that will be used in the project
                  value = data.get(i) # Gets the values assoaciated with the syntax
                  if value is not None:
                      selected_data[i] = float_converter(data.get(i)) # Converts it into a int value
              if not first:
                  output.write(',\n') # If conditions not met then create a new dictionary lines
              first = False
              output.write(json.dumps(selected_data)) # Function then writes the rating dictionary to the output file using the json.dumps() method to convert it to a JSON string.
          except json.JSONDecodeError:
              pass # Ignore line sthat cannotbe parsed into a JSON
        output.write('\n]\n') # When loop completes, the function writes a closing square bracket ']' and a new line to the output file, indicating the end of the JSONL file.


    aspect_rates = ['beer/name', 'review/appearance', 'review/aroma', 'review/palate', 'review/taste', 'review/overall', 'review/text']
    jsonline_converter('beeradvocate_reviews.json', 'beeradvocate_reviews.jsonl', aspect_rates)
```

To visualize the resulting in JSON-Lines text format of the variables that we need from BeerAdvocate dataset.

```
# Shows a sample of the first and last 10 extracted user reviews
!head -10 beeradvocate_reviews.jsonl
!tail -10 beeradvocate_reviews.jsonl
```

```
[
{"beer/name": "Sausa Weizen", "review/appearance": 2.5, "review/aroma": 2.0, "review/palate": 1.5, "review/taste": 1.5, "review/overall": 1.5, "review/text":
{"beer/name": "Red Moon", "review/appearance": 3.0, "review/aroma": 2.5, "review/palate": 3.0, "review/taste": 3.0, "review/overall": 3.0, "review/text": "D
{"beer/name": "Black Horse Black Beer", "review/appearance": 3.0, "review/aroma": 2.5, "review/palate": 3.0, "review/taste": 3.0, "review/overall": 3.0, "re
{"beer/name": "Sausa Pils", "review/appearance": 3.5, "review/aroma": 3.0, "review/palate": 2.5, "review/taste": 3.0, "review/overall": 3.0, "review/text":
{"beer/name": "Caldera Ginger Beer", "review/appearance": 4.0, "review/aroma": 4.0, "review/palate": 3.5, "review/taste": 4.0, "review/overall": 4.0, "revie
{"beer/name": "Caldera Oatmeal Stout", "review/appearance": 2.5, "review/aroma": 1.5, "review/palate": 2.5, "review/taste": 2.0, "review/overall": 2.0, "rev
{"beer/name": "Rauch \u00dcr Bock", "review/appearance": 3.0, "review/aroma": 4.5, "review/palate": 4.0, "review/taste": 4.5, "review/overall": 4.5, "review/
{"beer/name": "Rauch \u00dcr Bock", "review/appearance": 4.0, "review/aroma": 4.0, "review/palate": 3.0, "review/taste": 4.0, "review/overall": 4.0, "review/
{"beer/name": "Rauch \u00dcr Bock", "review/appearance": 4.0, "review/aroma": 4.5, "review/palate": 3.5, "review/taste": 4.0, "review/overall": 4.0, "review/
{"beer/name": "Four Horsemen #4 War", "review/appearance": 4.5, "review/aroma": 4.0, "review/palate": 3.0, "review/taste": 4.5, "review/overall": 4.0, "revi
{"beer/name": "2007 Resolution #1", "review/appearance": 4.0, "review/aroma": 4.0, "review/palate": 4.0, "review/taste": 4.0, "review/overall": 4.0, "review/
{"beer/name": "2007 Resolution #1", "review/appearance": 4.0, "review/aroma": 4.5, "review/palate": 4.0, "review/taste": 4.5, "review/overall": 4.5, "review/
{"beer/name": "Big Thumper Ale", "review/appearance": 3.0, "review/aroma": 4.0, "review/palate": 4.0, "review/taste": 4.5, "review/overall": 4.5, "review/te
{"beer/name": "Big Thumper Ale", "review/appearance": 4.0, "review/aroma": 4.0, "review/palate": 4.0, "review/taste": 4.0, "review/overall": 4.5, "review/te
{"beer/name": "Big Thumper Ale", "review/appearance": 3.5, "review/aroma": 3.5, "review/palate": 3.5, "review/taste": 3.5, "review/overall": 3.5, "review/te
{"beer/name": "Bear Mountain Ale", "review/appearance": 3.0, "review/aroma": 2.5, "review/palate": 2.0, "review/taste": 2.5, "review/overall": 2.0, "review/
{"beer/name": "Irish Amber", "review/appearance": 4.0, "review/aroma": 3.5, "review/palate": 4.0, "review/taste": 4.5, "review/overall": 4.5, "review/text":
{}
]
```

This part is setting up a local spark cluster

Set up a local spark cluster

Sourced from: Sample Project Code

```
[14] %env PYTHONHASHSEED 3
     !apt-get install openjdk-8-jdk-headless -qq > /dev/null
     !pip install -q pyspark
```

```
env: PYTHONHASHSEED=3
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 310.8/310.8 MB 3.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for pyspark (setup.py) ... done
```

```
[15] from math import sqrt
     import pyspark
     from pyspark import SparkConf, SparkContext
     from pyspark.sql import *
     spark = SparkSession.builder.master("local[*]").appName('SparkExample').config(
         "spark.executor.memory", "1g").config("spark.ui.port", "4050"
             ).getOrCreate()
     sc = spark.sparkContext
```

To load the data 'BeerAdvocate' into an Resilient Distributed Dataset (RDD)

Loading the data (BeerAdvocate) into an Resilient Distributed Dataset (RDD)

Sourced from: Sample Project Code

Add code cell ⌘/Ctrl+M B + Code + Text

```
[ ] user_reviews_df = spark.read.json("beeradvocate_reviews.jsonl", multiLine=True)
    user_reviews_df.printSchema()
    print(user_reviews_df.count())
    print(user_reviews_df.rdd.take(1))
    user_reviews_df_ids = user_reviews_df.rdd.zipWithUniqueId()
    print(user_reviews_df_ids.take(5))
```

```
root
 |-- beer/name: string (nullable = true)
 |-- review/appearance: double (nullable = true)
 |-- review/aroma: double (nullable = true)
 |-- review/overall: double (nullable = true)
 |-- review/palate: double (nullable = true)
 |-- review/taste: double (nullable = true)
 |-- review/text: string (nullable = true)

589528
[Row(beer/name='Sausa Weizen', review/appearance=2.5, review/aroma=2.0, review/overall=1.5, review/palate=1.5, review/taste=1.5, review/text='A lot of foam.
[(Row(beer/name='Sausa Weizen', review/appearance=2.5, review/aroma=2.0, review/overall=1.5, review/palate=1.5, review/taste=1.5, review/text='A lot of foam.
```

The code below will be later used in the text reviews of each user on a specific beer. What this code basically do is, it helps us remove any insignificant words that do not really add any meaningful explanation about a text reviews on a beer. This is allowed by importing the 'nltk' which provides collection of 'stopwords' for various languages. Hence, in the result

output we can see some examples of stop words that are very common e.g., few, only, me, this, the, and, etc. This can be useful in our project when performing text analysis to prevent any commonalities of words from each user text review. Sourced from the Sample project code and from the article made by Chuahan, (2019).

```
[17] # download a set of stop words that we can ignore because they are not interesting
     # Sourced from: DATA301 Sample Project Code
     !pip install nltk

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
     Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.3)
     Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.2.0)
     Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2022.10.31)
     Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.65.0)
```

The code below imports/download a Natural Language Toolkit (nltk) library and the English stopwords corpus. Then prints out lists of stopwords in English.

Sourced from: geeksforgeeks.org and DATA301 Sample Project Code

```
import nltk
nltk.download('stopwords')

from nltk.corpus import stopwords

STOP_WORDS = set(stopwords.words('english'))
print(STOP_WORDS)

{'few', 'only', 's', 'own', 'me', 'this', 'wasn', 'we', 'on', 'be', "didn't", 'up', 'ours', 'by', 'd', 'your', 'until', 'my', 'yours', 'won', 'the', 'before
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

This process is to clean and remove any unnecessary things in the RDD dataset. I have sourced this code from the DATA3012 Sample Project Code, because I need it for removing non-letters, stopwords, and perform a word count to return a dictionary (result) containing the count of each words.

Cleaning the dataset stored in JSON-line

The section below performs word count within a document first (we consider a single review text to be a document so we aren't going to parallelize the per document word count since we have so many documents, it is much more efficient to parallelize the documents as a whole)

The section below parallizes the documents by performing word count within a singular document.

Sourced from: re — Regular expression operations and DATA301 Sample Project

```
[15] # Perform word count within a document first (we consider a single review text to be a document so we aren't going to parallelize the per document word count since we have so many documents, it is much m

     import re

     # Takes a string input and returns a new string without all 'non-letter'
     # The re.sub() function replaces non-letters specified into an empty string
     def remove_nonletters(word):
       """Removes any nonletter words"""
       return re.sub(r'[^a-zA-Z]', '', word)

     # This function helps remove any insignificant words that do not really add any meaningful explanation about a text reviews on a beer
     def split_remove_nonletters(line):
       """Removes any insignificant words that do not add any meaning to the text reviews"""
       result = []
       for word in line.split(" "):
         removed_token = remove_nonletters(word.lower())
         if removed_token != '':
           result.append((removed_token, 1))
       return result

     # Parses the review_str and generates a dictionary containing word:count entries
     # The funciton takes a string as an input and then perform a word count and return a dictionary (result) containing the count of each words.
     def wc(review_str):
       """Takes a string as an input and performs a word count on the input strings and returns a dictionary result containing counts of each word"""
       result = {}
       for word in review_str.split():
         removed_token = remove_nonletters(word.lower())
         if removed_token != '' and removed_token not in STOP_WORDS:
           if removed_token not in result:
             result[removed_token] = 0
           result[removed_token] += 1
       return result
```

This code is for testing and visualizing my current RDD data. What happens in the first chunk of code below is it maps through the RDD and just apply the word count function 'wc()' in the RDD. Which counts only the significant words that is meaningful in the review text.

The second chunk of code is another testing to see whether it would make sense to put the 'review/overall' and 'review/text' in a tuple. And then put that tuple in another tuple which contains that and the index of each users. Basically it looks like this:

[(user_index, (overall rating of a beer, {word count of significant words}))]

```
### Applying word count of the significant words (that were meaningful) in the reviews text of each individuals on a beer
### per User Document Count (UDC)
per_UDC = user_reviews_df_ids.map(lambda x: (x[1], wc(x[0]['review/text'])))
print(per_UDC.take(5))

[(0, {'lot': 2, 'foam': 2, 'smell': 1, 'banana': 2, 'lactic': 2, 'tart': 1, 'good': 1, 'start': 1, 'quite': 1, 'dark': 1, 'orange': 1, 'color': 1, 'lively': 1, 'carbonation': 1, 'visibl
```

```
[63] ### Grabbing variables that will be used in TF-IDF algorithm
     ### Applying word count of the significant words (that were meaningful) in the reviews text of each individuals on a beer
     ### per User Document Count (UDC)
     per_UDC = user_reviews_df_ids.map(lambda x: (x[1], (x[0]['review/overall'], wc(x[0]['review/text']))))
     print(per_UDC.take(5))

     ### NOT sure if working
     #per_UDC_ratings = per_UDC.filter(lambda x: overall_splitter(x[0]['review/overall']))

     [(0, (1.5, {'lot': 2, 'foam': 2, 'smell': 1, 'banana': 2, 'lactic': 2, 'tart': 1, 'good': 1, 'start': 1, 'quite': 1, 'dark': 1, 'orange': 1, 'color': 1, 'lively': 1, 'carbonation': 1, '
```

This is the part was a road block and I was planning on discontinuing the process of splitting the 'review/overall' into three categories of negative (0-6), neutral (7-13), and positive (14-20) because I was struggling on imagining of what would the RDD would look like if I continue to proceed with it. This was a hard progress in my project due to difficulty in coming up of how the RDD would look like or whether it would make sense.

But I got help from a friend (Sudarshan) on the second chunk of code in the picture below. Initially I was trying to loop through 'review/overall' when I did not need to. And after that, I was able to map through the RDD and got the three categories working (as shown in the 3rd chunk of code below).

```
[45] ### TESTING
     num = 1.5
     string = (f"neagtive {num}")
     print(string)

     neagtive 1.5

### This function help split the overall review ratings by a user on a beer into three categories of negative (0-6), neutral (7-13), and positive (14-20)
def overall_splitter(ratings):
  """Splits the overall review ratings made by a user into a three categoried. That is negative (score from 0-6), neutral (score from 7-13), and positive (score from 14-20)"""
  # for i in range(ratings): ### don't need this
  if ratings <= 6:
    return f'negative {ratings}'
  elif ratings > 6 and ratings <= 13:
    return f'neutral {ratings}'
  elif ratings > 13 and ratings <= 20:
    return f'positive {ratings}'
  else:
    return f'Exceeded the overall rating range of 20 by {ratings - 20}'

[65] ### Displaying to make sure we have everything that we need before we start doing TF-IDF Algorithm

     per_UDC = user_reviews_df_ids.map(lambda x: (x[1], (overall_splitter(x[0]['review/overall']), wc(x[0]['review/text']))))
     print(per_UDC.take(5))

     #print(per_UDC_ratings.collect())

     #per_UDC_ratings = user_reviews_df_ids.map(lambda x: (x[1], (overall_splitter(x[0]['review/overall'], wc(x[0]['review/text']))))))

     #per_UDC = user_reviews_df_ids.map(lambda x: (x[1], wc(x[0]['review/text'])))
     #print(per_UDC.take(5))

     [(0, ('negative 1.5', {'lot': 2, 'foam': 2, 'smell': 1, 'banana': 2, 'lactic': 2, 'tart': 1, 'good': 1, 'start': 1, 'quite': 1, 'dark': 1, 'orange': 1, 'color': 1, 'lively': 1, 'carbona
```

At this point, I am in the process of starting my Term Frequency – Inverse Document Frequency (TF-IDF) algorithm. But along the way I am constantly finding small things that needed to be done before then. For example, we need to double check whether the loaded RDD BeerAdvocate data has the right variables and in the right format.

Right now I am having trouble with my TF-IDF algorithm. The first part that I'm doing is 'idf' but I been getting an error every now and then every time I try to change the number inside the '.take()' function. So what I tend to do is 'disconnect and delete runtime' and rerun the whole thing again.

```
## TESTING
import math

def idf(user_count):
  """Measures the importance of a term (words) in a collection of documents (per user reviews)"""
  sample1 = user_count.map(lambda x: x[1][1])
  #print(sample1.take(10))

  #sample2 = sample1.flatMap(lambda x: [(word, 1) for word in x.keys()])
  #print(sample2.take(5))

  #sample11 = user_count.flatMap(lambda x: [(word, 1) for word in x[1].keys()])

  #sample12 = sample11.reduceByKey(lambda a, b: a + b)

  #sample3 = sample2.reduceByKey(lambda a, b: a + b)

  #n = per_UDC.count()
  #print(n)

  #sample4 = sample3.map(lambda x: (x[0], math.log(n/x[1],2)))

  return sample1

  # MIGHT NOT NEED THIS FROM HERE
  #count_reviews = user_count.flatMap(lambda x: [(word, 1) for word in x[1].keys()])
  #print(count_reviews.take(5))

  #overall_per_count_reviews = count_reviews.reduceByKey(lambda a, b: a + b)

  #number = overall_per_count_reviews.count()

  #return overall_per_count_reviews.map(lambda x: (x[0], math.log(number/x[1],2)))

  #print(overall_per_count_reviews.take(5))
  #return count_reviews
  #print(count_reviews.take(5))
  # TO HERE
```

```
  #print(count_reviews.take(5))
  # TO HERE

word_idf = idf(per_UDC)
print(word_idf.take(5))

---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-29-1f49ee82afb0> in <cell line: 40>()
     38   # TO HERE
     39
---> 40 word_idf = idf(per_UDC)
     41 print(word_idf.take(5))

                              5 frames
/usr/local/lib/python3.10/dist-packages/py4j/protocol.py in get_return_value(answer, gateway_client, target_id, name)
    324             value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
    325             if answer[1] == REFERENCE_TYPE:
--> 326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
    328                     format(target_id, ".", name), value)

Py4JJavaError: An error occurred while calling z:org.apache.spark.api.python.PythonRDD.runJob.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 19.0 failed 1 times, most recent failure: Lost task 0.0 in stage 19.0 (TID 18) (00fa521add21
executor driver): org.apache.spark.api.python.PythonException: Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/pyspark/python/lib/pyspark.zip/pyspark/worker.py", line 830, in main
    process()
  File "/usr/local/lib/python3.10/dist-packages/pyspark/python/lib/pyspark.zip/pyspark/worker.py", line 822, in process
    serializer.dump_stream(out_iter, outfile)
  File "/usr/local/lib/python3.10/dist-packages/pyspark/python/lib/pyspark.zip/pyspark/serializers.py", line 274, in dump_stream
    vs = list(itertools.islice(iterator, batch))
  File "/usr/local/lib/python3.10/dist-packages/pyspark/rdd.py", line 2830, in takeUpToNumLeft
    yield next(iterator)
  File "/usr/local/lib/python3.10/dist-packages/pyspark/python/lib/pyspark.zip/pyspark/util.py", line 81, in wrapper
    return f(*args, **kwargs)
  File "<ipython-input-29-1f49ee82afb0>", line 9, in <lambda>
AttributeError: 'tuple' object has no attribute 'keys'

        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonException(PythonRunner.scala:561)
        at org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:767)
        at org.apache.spark.api.python.PythonRunner$$anon$3.read(PythonRunner.scala:749)
        at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRunner.scala:514)
        at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at org.apache.spark.InterruptibleIterator.foreach(InterruptibleIterator.scala:28)
        at scala.collection.generic.Growable.$plus$plus$eq(Growable.scala:62)
```

Another rode block that is preventing me from making progress is trying to get my TF-IDF working. Hence, I need to go to the labs to get help from tutors.

References

Singh, A. (2015). *CSE 225: Assignment Winter 2015*. (A53035244). https://cseweb.ucsd.edu/classes/wi15/cse255-a/reports/wi15/Alok_Singh.pdf

McAuley, J., Leskovec, J., Standford, D. J. (n.d.). *Learning Attitudes and Attributes from Multi-Aspect Reviews.* https://cseweb.ucsd.edu/~jmcauley/pdfs/icdm12.pdf

Ni, J., Li, J., McAuley, J. (2019). *Justifying recommendations using distantly-labelled reviews and fined-grained aspects.* Empirical Methods in Natural Language Processing (EMNLP). https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/

Chauhan, N., S. (2020). *Natural language processing in Apache Spark using* NLTK (part 1/2). https://www.theaidream.com/post/natural-language-processing-in-apache-spark-using-nltk-part-1-2

McAuley, J. (n.d.). Recommender Systems and Personalization Datasets. https://cseweb.ucsd.edu/~jmcauley/datasets.html#foodcom