STA323 Project 1 report

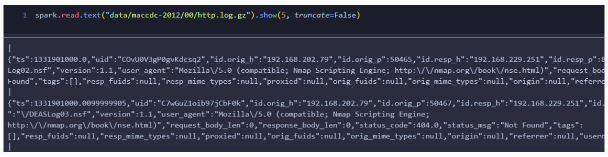
SID: 12110821
Name: ZHANG Chi

Solution for Q1

The data given is about an an existing network infrastructure, which is stored in multiple directories.

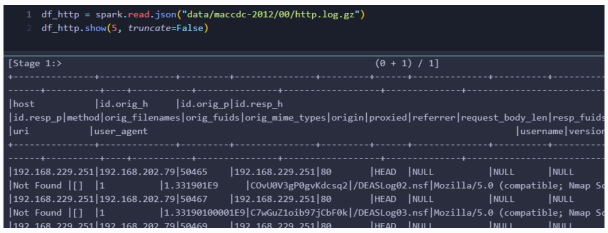
(1)

Before we read the whole data, we need to know the structure of the data. I use `spark.read.text` to read one `http.log.gz` of the first directory, and we can find that each record is in `json` format. So we can use `spark.read.json` to read the data.



```
spark.read.text("data/maccdc-2012/00/http.log.gz").show(5, truncate=False)
```

```
|
{"ts":1331901000.0,"uid":"COvU0V3gP0gvKdcsq2","id.orig_h":"192.168.202.79","id.orig_p":50465,"id.resp_h":"192.168.229.251","id.resp_p":8
Log02.nsf","version":1.1,"user_agent":"Mozilla\/5.0 (compatible; Nmap Scripting Engine; http:\/\/nmap.org\/book\/nse.html)","request_bod
Found","tags":[],"resp_fuids":null,"resp_mime_types":null,"proxied":null,"orig_fuids":null,"orig_mime_types":null,"origin":null,"referre
|
{"ts":1331901000.0099999905,"uid":"C7wGuZ1oib97jCbF0k","id.orig_h":"192.168.202.79","id.orig_p":50467,"id.resp_h":"192.168.229.251","id.
":"\/DEASLog03.nsf","version":1.1,"user_agent":"Mozilla\/5.0 (compatible; Nmap Scripting Engine;
http:\/\/nmap.org\/book\/nse.html)","request_body_len":0,"response_body_len":0,"status_code":404.0,"status_msg":"Not Found","tags":
[],"resp_fuids":null,"resp_mime_types":null,"proxied":null,"orig_fuids":null,"orig_mime_types":null,"origin":null,"referrer":null,"usern
|
```



```
df_http = spark.read.json("data/maccdc-2012/00/http.log.gz")
df_http.show(5, truncate=False)
```

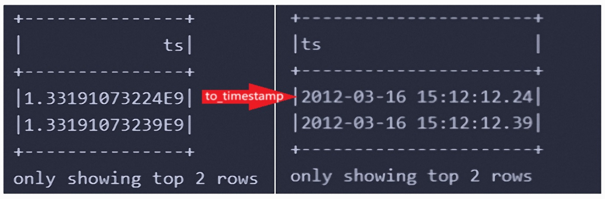
```
[Stage 1:>                                                          (0 + 1) / 1]
+---------------+--------------+--------+---------------+--------+------+--------------+----------+--------
-----+---------+----+-----------+---------------+------------------+--------------+--------------------
|host           |id.orig_h     |id.orig_p|id.resp_h
|id.resp_p|method|orig_filenames|orig_fuids|orig_mime_types|origin|proxied|referrer|request_body_len|resp_fuids
|uri           |user_agent                                                                  |username|version
+---------------+--------------+--------+---------------+--------+------+--------------+----------+--------
-----+---------+----+-----------+---------------+------------------+--------------+--------------------
|192.168.229.251|192.168.202.79|50465    |192.168.229.251|80       |HEAD  |NULL          |NULL      |NULL
|Not Found |[]  |1          |1.331901E9     |COvU0V3gP0gvKdcsq2|/DEASLog02.nsf|Mozilla/5.0 (compatible; Nmap Sc
|192.168.229.251|192.168.202.79|50467    |192.168.229.251|80       |HEAD  |NULL          |NULL      |NULL
|Not Found |[]  |1          |1.33190100001E9|C7wGuZ1oib97jCbF0k|/DEASLog03.nsf|Mozilla/5.0 (compatible; Nmap Sc
|192.168.229.251|192.168.202.79|50469    |192.168.229.251|80       |HEAD  |NULL          |NULL      |NULL
```

Given a path list, the function can also read the data from the each path from path list and concat them together to a single dataframe.

```python
folders = ["00", "01", "02", "03", "04", "05"]
df_http = spark.read.json([f"data/maccdc-2012/{folder}/http.log.gz" for folder in folders])
df_dns = spark.read.json([f"data/maccdc-2012/{folder}/dns.log.gz" for folder in folders])
```

After checking the column `ts`, we can feel free to convert it to a `Timestamp` data type by `to_timestamp` with `withColumn` function. And the temp view is created for further analysis.

```python
df_http = df_http.withColumn("ts", to_timestamp(col("ts")))
df_dns = df_dns.withColumn("ts", to_timestamp(col("ts")))
df_http.createOrReplaceTempView("http_log")
df_dns.createOrReplaceTempView("dns_log")
```



```
+---------------+
|             ts|
+---------------+
|1.33191073224E9|
|1.33191073239E9|
+---------------+
only showing top 2 rows
```

```
+----------------------+
|ts                    |
+----------------------+
|2012-03-16 15:12:12.24|
|2012-03-16 15:12:12.39|
+----------------------+
only showing top 2 rows
```

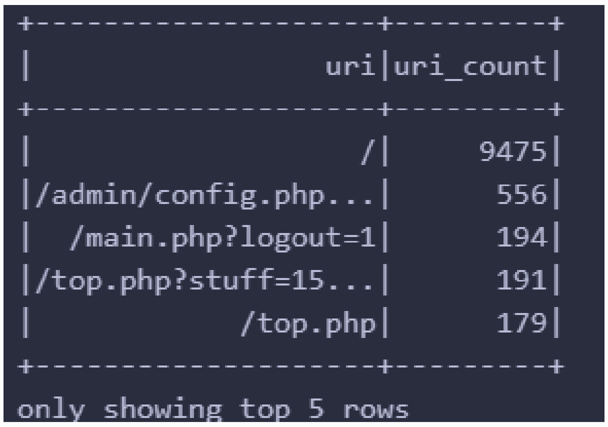
(2)

In this task, we need to find the top uri that has been accessed most among the records whose `uid` is `status_code` is 200 and `method` used is `GET`.

We can use `filter` to filter the records and then group by the `uri` and count the records. Finally, we can sort the result by the count in descending order and show the top records. The code is as follows:

```
# In Spark SQL API
spark.sql("SELECT uri, COUNT(uri) AS uri_count FROM http_log WHERE status_code = 200 AND method = 'GET' GROUP BY uri order by uri_count desc").show(5)

# In Spark DataFrame API
df_http.select(col("uri")).filter((col("status_code")==200) & (col("method")=='GET')).groupBy(col("uri")).agg(count(col("uri")).alias("uri_count")).orderBy(col("uri_count").desc()).show(5)
```

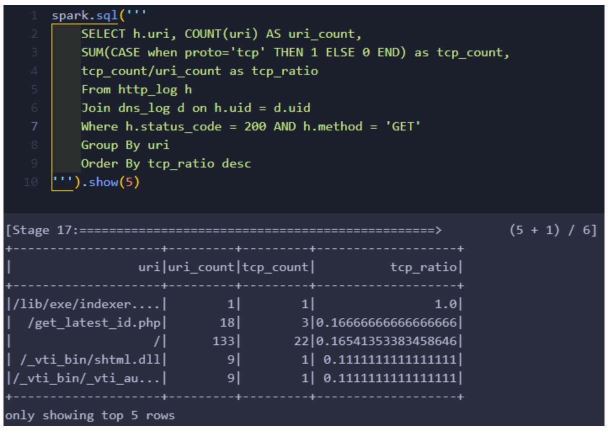
And they get the same result:



```
+--------------------+---------+
|                 uri|uri_count|
+--------------------+---------+
|                   /|     9475|
|/admin/config.php...|      556|
|   /main.php?logout=1|      194|
|/top.php?stuff=15...|      191|
|            /top.php|      179|
+--------------------+---------+
only showing top 5 rows
```

(3)

After inner joining two tables by `uid`, filtering the records in the same way as in the last task, the number of record whose `proto` is `tcp` for each `uri` can be calulated by `SUM(CASE when proto='tcp' THEN 1 ELSE 0 END) Group By uri`. Then results are sorted by the `tcp_ratio` in descending order and top records are showed below.

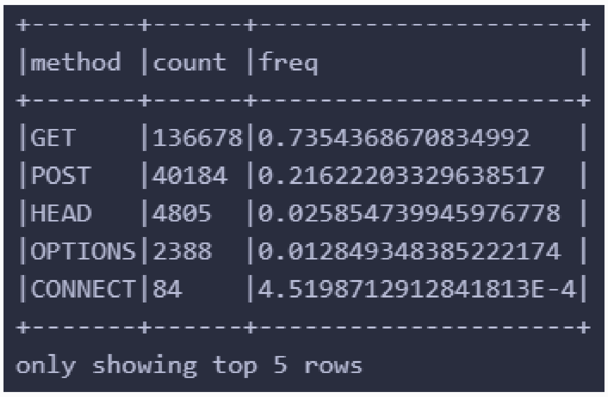


```
spark.sql('''
    SELECT h.uri, COUNT(uri) AS uri_count,
    SUM(CASE when proto='tcp' THEN 1 ELSE 0 END) as tcp_count,
    tcp_count/uri_count as tcp_ratio
    From http_log h
    Join dns_log d on h.uid = d.uid
    Where h.status_code = 200 AND h.method = 'GET'
    Group By uri
    Order By tcp_ratio desc
''').show(5)
```

```
[Stage 17:==============================================>          (5 + 1) / 6]
+--------------------+---------+---------+------------------+
|                 uri|uri_count|tcp_count|         tcp_ratio|
+--------------------+---------+---------+------------------+
|/lib/exe/indexer....|        1|        1|               1.0|
|  /get_latest_id.php|       18|        3|0.1666666666666666|
|                   /|      133|       22|0.1654135338345864|
| /_vti_bin/shtml.dll|        9|        1| 0.111111111111111|
|/_vti_bin/_vti_au...|        9|        1| 0.111111111111111|
+--------------------+---------+---------+------------------+
only showing top 5 rows
```

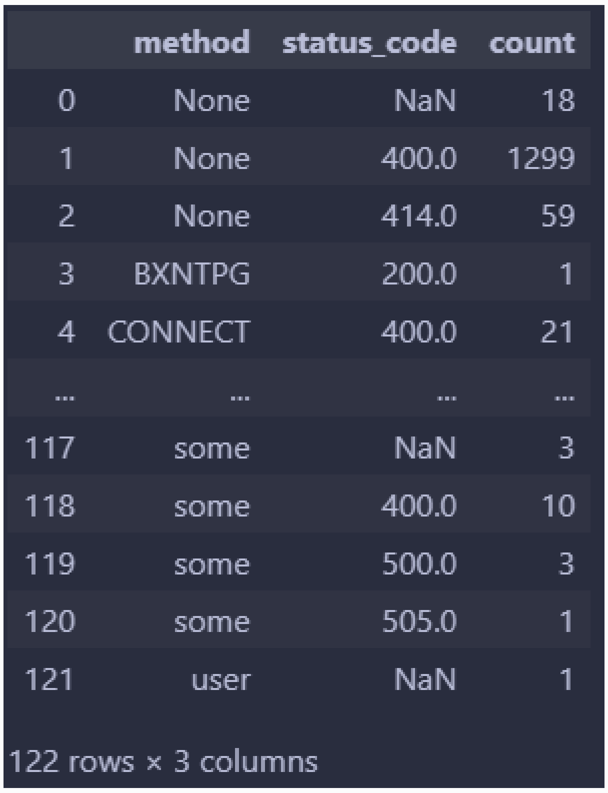
(4)

According to the requirement, we need to calculate the use frequency of each method. Then `groupBy` and `agg` can be used directly. The result are showing below:



```
+-------+------+--------------------+
|method |count |freq                |
+-------+------+--------------------+
|GET    |136678|0.7354368670834992  |
|POST   |40184 |0.21622203329638517 |
|HEAD   |4805  |0.025854739945976778|
|OPTIONS|2388  |0.012849348385222174|
|CONNECT|84    |4.5198712912841813E-4|
+-------+------+--------------------+
only showing top 5 rows
```

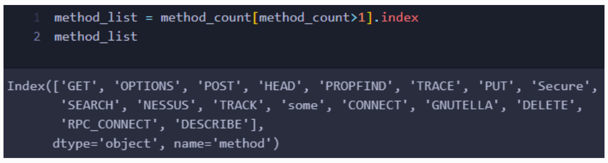
To get the number of records in different status code for each method, two columns have been put into `GroupBy`. Before we plot the pie chart, we need to convert the result to a pandas dataframe.



	method	status_code	count
0	None	NaN	18
1	None	400.0	1299
2	None	414.0	59
3	BXNTPG	200.0	1
4	CONNECT	400.0	21
...	...	...	...
117	some	NaN	3
118	some	400.0	10
119	some	500.0	3
120	some	505.0	1
121	user	NaN	1

122 rows × 3 columns

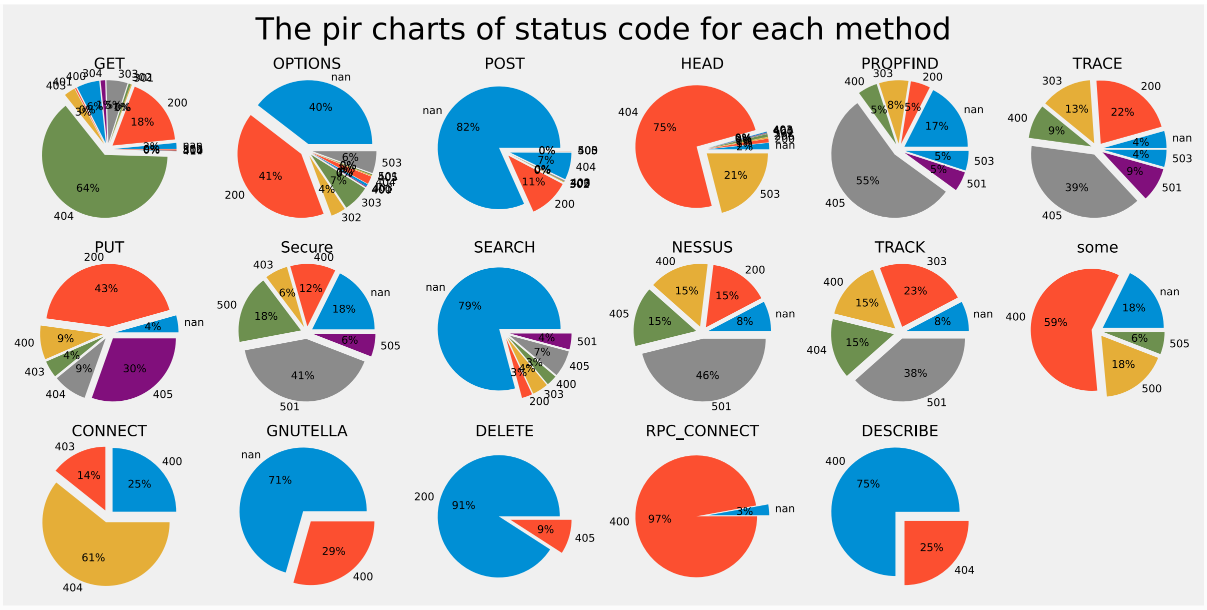
It is notable that some values in column `method` and column `status_code` are missing. And I will drop `None` but keep `NaN` as we want to get distribution of status code regarding to a specific method. Besides, the method emerging once (only appearing with one kind of status code) will also be scrapped, and selected methods are shown below.



```
method_list = method_count[method_count>1].index
method_list
```

```
Index(['GET', 'OPTIONS', 'POST', 'HEAD', 'PROPFIND', 'TRACE', 'PUT', 'Secure',
       'SEARCH', 'NESSUS', 'TRACK', 'some', 'CONNECT', 'GNUTELLA', 'DELETE',
       'RPC_CONNECT', 'DESCRIBE'],
      dtype='object', name='method')
```

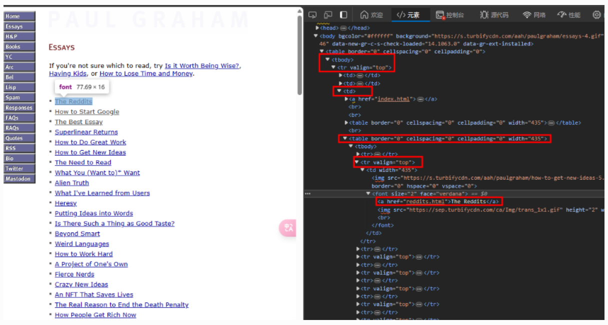
Then we can use `matplotlib` to plot the pie chart.



Solution for Q2

(1)

To scratch the data, I use `request` to get the data from the given website and use `BeautifulSoup` to parse the data. In the main page, there is a list of links to the detailed information of each article. I used `find_all` to get a link list and then request the page related to each link to get the article. Here is a shortcut of the website structure.



ESSAYS

If you're not sure which to read, try Is it Worth Being Wise?, Having Kids, or How to Lose Time and Money.

font 77.69 × 16

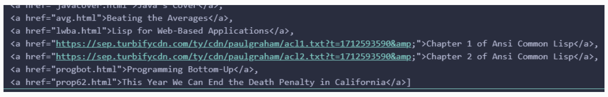
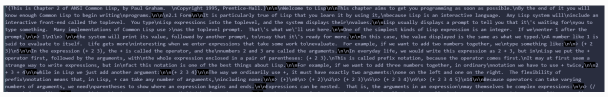
- The Reddits
- How to Start Google
- The Best Essay
- Superlinear Returns
- How to Do Great Work
- How to Get New Ideas
- The Need to Read
- What You (Want to)* Want
- Alien Truth
- What I've Learned from Users
- Heresy
- Putting Ideas into Words
- Is There Such a Thing as Good Taste?
- Beyond Smart
- Weird Languages
- How to Work Hard
- A Project of One's Own
- Fierce Nerds
- Crazy New Ideas
- An NFT That Saves Lives
- The Real Reason to End the Death Penalty
- How People Get Rich Now

```
▶<head>⋯</head>
▼<body bgcolor="#ffffff" background="https://s.turbifycdn.com/aah/paulgraham/essays-4.gif"
 46" data-new-gr-c-s-check-loaded="14.1063.0" data-gr-ext-installed>
  ▼<table border="0" cellspacing="0" cellpadding="0">
    ▼<tbody>
      ▼<tr valign="top">
        ▶<td>⋯</td>
        ▶<td>⋯</td>
        ▼<td>
          ▶<a href="index.html">⋯</a>
           <br>
           <br>
          ▶<table border="0" cellspacing="0" cellpadding="0" width="435">⋯</table>
           <br>
          ▼<table border="0" cellspacing="0" cellpadding="0" width="435">
            ▼<tbody>
              ▶<tr>⋯</tr>
              ▼<tr valign="top">
                ▼<td width="435">
                   <img src="https://s.turbifycdn.com/aah/paulgraham/how-to-get-new-ideas-5.
                   border="0" hspace="0" vspace="0">
…                 ▼<font size="2" face="verdana"> == $0
                     <a href="reddits.html">The Reddits</a>
                     <img src="https://sep.turbifycdn.com/ca/Img/trans_1x1.gif" height="2" w
                     <br>
                   </font>
                 </td>
               </tr>
              ▶<tr>⋯</tr>
              ▶<tr valign="top">⋯</tr>
              ▶<tr>⋯</tr>
              ▶<tr valign="top">⋯</tr>
              ▶<tr>⋯</tr>
              ▶<tr valign="top">⋯</tr>
              ▶<tr>⋯</tr>
              ▶<tr valign="top">⋯</tr>
              ▶<tr>⋯</tr>
```

Then there is a `for` loop to get the data from each page. In each paragraph of one article in a page, texts are seperated into multiple lines under unknown resons. In this case, I first convert the soup object into a string, replace `'\n'` into `""` and split the string by `<br/><br/>`, which is a kind of paragraph seperator in html. Now each sentence has been grouped into their own paragraph. However, there are also some annoying html tags in the text. I use regular expression to remove them.

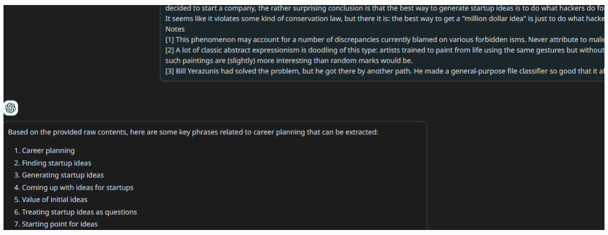
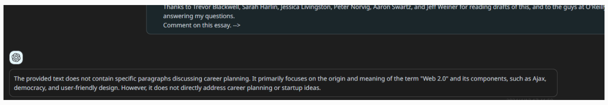
Besides, I also replace some special characters in the title of the article to avoid the error of reading files by `spark.sparkContext.textFile`.

```python
for i in tqdm(article_list):
    try:
        href = i['href']
        if "http" in href:
            article_page = requests.get(href)
            split_text = re.split('\n{2,}', article_page.text)
            text = ("\n").join([i.replace("\n", " ").strip("-").strip() for i in split_text if i != ""]).strip("\n")
        else:
            article_page = requests.get(f"{suburl}{href}")
            article_soup = BeautifulSoup(article_page.text, 'html.parser')
            article_text = article_soup.find("font", face = "verdana")
            para_list = str(article_text).replace("\n", " ").split("<br/><br/>")
            text = ("\n").join([re.sub('<.*?>', '', i).strip() for i in para_list if i != ""]).strip("\n")

        txt_file = i.text.replace("/", "or").replace("?", "").replace(",","").replace(":", "")

        with open(f"data/paul_articles/{txt_file}.txt", "w") as f:
            f.write(text)
    except Exception as e:
        print(f"Error with {i.text}: {e}")
        continue
```

> Two url links are not in the format of previous links, and even their related pages are not consistent with others. I use a `if` statement to deal with them separately.
>
> 
>
> ```
> <a href="avg.html">Beating the Averages</a>,
> <a href="lwba.html">Lisp for Web-Based Applications</a>,
> <a href="https://sep.turbifycdn.com/ty/cdn/paulgraham/acl1.txt?t=1712593590&amp;">Chapter 1 of Ansi Common Lisp</a>,
> <a href="https://sep.turbifycdn.com/ty/cdn/paulgraham/acl2.txt?t=1712593590&amp;">Chapter 2 of Ansi Common Lisp</a>,
> <a href="progbot.html">Programming Bottom-Up</a>,
> <a href="prop62.html">This Year We Can End the Death Penalty in California</a>]
> ```
>
> Same as the previous step, `\n` and `\n+` (more `\n` like `\n\n`, `\n\n\n`) are used simultaneously in one page. Luckily, I find that `\n+` is used to seperate paragraphs. So I use `re.split` by `\n+` to split the whole page. Then I can let every paragraph to be a single line.
>
> 
>
> ```
> '(This is Chapter 2 of ANSI Common Lisp, by Paul Graham.  \nCopyright 1995, Prentice-Hall.)\n\n\nWelcome to Lisp\n\nThis chapter aims to get you programming as soon as possible.\nBy the end of it you will
> know enough Common Lisp to begin writing\nprograms.\n\n\n2.1 Form\n\nIt is particularly true of Lisp that you learn it by using it,\nbecause Lisp is an interactive language.  Any Lisp system will\ninclude an
> interactive front-end called the toplevel.  You type\nLisp expressions into the toplevel, and the system displays their\nvalues.\n\nLisp usually displays a prompt to tell you that it\'s waiting for\nyou to
> type something.  Many implementations of Common Lisp use >\nas the toplevel prompt.  That\'s what we\'ll use here.\n\nOne of the simplest kinds of Lisp expression is an integer.  If we\nenter 1 after the
> prompt,\n\n> 1\n1\n> \n\nthe system will print its value, followed by another prompt, to\nsay that it\'s ready for more.\n\nIn this case, the value displayed is the same as what we typed.\nA number like 1 is
> said to evaluate to itself.  Life gets more\ninteresting when we enter expressions that take some work to\nevaluate.  For example, if we want to add two numbers together, we\ntype something like:\n\n> (+ 2
> 3)\n5\n\nIn the expression (+ 2 3), the + is called the operator, and the\nnumbers 2 and 3 are called the arguments.\n\nIn everyday life, we would write this expression as 2 + 3, but in\nLisp we put the +
> operator first, followed by the arguments, with\nthe whole expression enclosed in a pair of parentheses: (+ 2 3).\nThis is called prefix notation, because the operator comes first.\nIt may at first seem a
> strange way to write expressions, but in\nfact this notation is one of the best things about Lisp.\n\nFor example, if we want to add three numbers together, in ordinary\nnotation we have to use + twice,\n\n2
> + 3 + 4\n\nwhile in Lisp we just add another argument:\n\n(+ 2 3 4)\n\nThe way we ordinarily use +, it must have exactly two arguments:\none on the left and one on the right.  The flexibility of
> prefix\nnotation means that, in Lisp, + can take any number of arguments,\nincluding none:\n\n> (+)\n0\n> (+ 2)\n2\n> (+ 2 3)\n5\n> (+ 2 3 4)\n9\n> (+ 2 3 4 5)\n14\n\nBecause operators can take varying
> numbers of arguments, we need\nparentheses to show where an expression begins and ends.\n\nExpressions can be nested.  That is, the arguments in an expression\nmay themselves be complex expressions:\n\n> (/
> ```

(2)

In this task, the main problem is to get key phrases in high quality. To finish it, I ask `chatgpt` for help, and my prompts as well as corresponding reponses are as follows:

> In my Pyspark task, I have a rdd that contains texts (each element is a paragraph) written by a author. Now I want to find paragraphs talking about 'career planning', could you help me find some key phrases from raw contents which I will give you to help me filter elemetns in rdd? If Ok I will give you the raw contents. And you will give me the key phrases.
>
> 
>
> decided to start a company, the rather surprising conclusion is that the best way to generate startup ideas is to do what hackers do fo
> It seems like it violates some kind of conservation law, but there it is: the best way to get a "million dollar idea" is just to do what hacke
> Notes
> [1] This phenomenon may account for a number of discrepancies currently blamed on various forbidden isms. Never attribute to mali
> [2] A lot of classic abstract expressionism is doodling of this type: artists trained to paint from life using the same gestures but without
> such paintings are (slightly) more interesting than random marks would be.
> [3] Bill Yerazunis had solved the problem, but he got there by another path. He made a general-purpose file classifier so good that it a
>
> Based on the provided raw contents, here are some key phrases related to career planning that can be extracted:
>
> 1. Career planning
> 2. Finding startup ideas
> 3. Generating startup ideas
> 4. Coming up with ideas for startups
> 5. Value of initial ideas
> 6. Treating startup ideas as questions
> 7. Starting point for ideas
>
> 
>
> Thanks to Trevor Blackwell, Sarah Harlin, Jessica Livingston, Peter Norvig, Aaron Swartz, and Jeff Weiner for reading drafts of this, and to the guys at O'Reill
> answering my questions.
> Comment on this essay. -->
>
> The provided text does not contain specific paragraphs discussing career planning. It primarily focuses on the origin and meaning of the term "Web 2.0" and its components, such as Ajax, democracy, and user-friendly design. However, it does not directly address career planning or startup ideas.

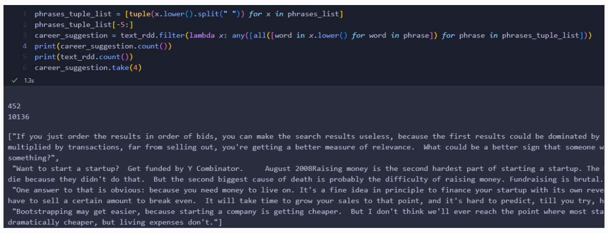
The raw pages I give the gpt are "Ideas for Startups", "Web 2.0", "Why TV lost" and so on. Sometimes it can give me the key phrases directly, and sometimes it can find that the raw contents are not related to the career. Including phrases Induced by my self or by online information, I summarize all these phrases (listed in Appendix A) into the `phrases_list` and use them to filter the paragraphs in the rdd.

There are ways I think can be used:

- For each phrase in the `phrases_list`, I will tokenize it to a small phrase_tuple. If one paragraph contains all words of any phrase_tuple (even in disorder or in different morphologys), I will consider it as a paragraph talking about career planning, which will be kept in the final result.
- Let all tokens in the paragraph be the nodes of a graph, and the edges are the co-occurrence of two words in the same paragraph. Then I will use a graph algorithm to find the subgraph that contains all nodes in the phrase. If the subgraph is connected, the paragraph is considered to be talking about career planning.

> Word tenses and case should be ignored because authors may use different tenses and case to convey the same meaning. So I will convert words into lower cases as well as normalize each of them. What I used to normalize words is deleting letters. For example, If `coming` is converted to `come`, then `coming` may not be pointed out, so I use `com` to represent `coming`. But this may relate to some unwanted words like `common`. In my opinion, more is better then less.

Taking the complexity and time efficiency into account, I use the first way to filter the paragraphs. Here is a shortcut of the selected paragraphs.

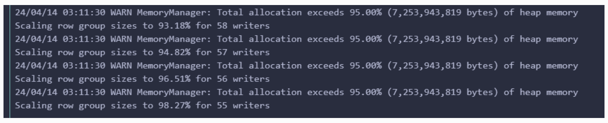


```
phrases_tuple_list = [tuple(x.lower().split(" ")) for x in phrases_list]
phrases_tuple_list[-5:]
career_suggestion = text_rdd.filter(lambda x: any([all([word in x.lower() for word in phrase]) for phrase in phrases_tuple_list]))
print(career_suggestion.count())
print(text_rdd.count())
career_suggestion.take(4)
```

```
✓ 1.3s

452
10136

["If you just order the results in order of bids, you can make the search results useless, because the first results could be dominated by
multiplied by transactions, far from selling out, you're getting a better measure of relevance.  What could be a better sign that someone w
something?",
 "Want to start a startup?  Get funded by Y Combinator.     August 2008Raising money is the second hardest part of starting a startup. The
die because they didn't do that.  But the second biggest cause of death is probably the difficulty of raising money. Fundraising is brutal.
 "One answer to that is obvious: because you need money to live on. It's a fine idea in principle to finance your startup with its own reve
have to sell a certain amount to break even.  It will take time to grow your sales to that point, and it's hard to predict, till you try, h
 "Bootstrapping may get easier, because starting a company is getting cheaper.  But I don't think we'll ever reach the point where most sta
dramatically cheaper, but living expenses don't."]
```

To save the result into a `parquet` file, I first convert the rdd into a dataframe and then save it.

```
spark.createDataFrame(career_suggestion.map(lambda x: Row(text=x))).write.format("parquet").mode("overwrite").save("output/career_suggestion.parquet")
```

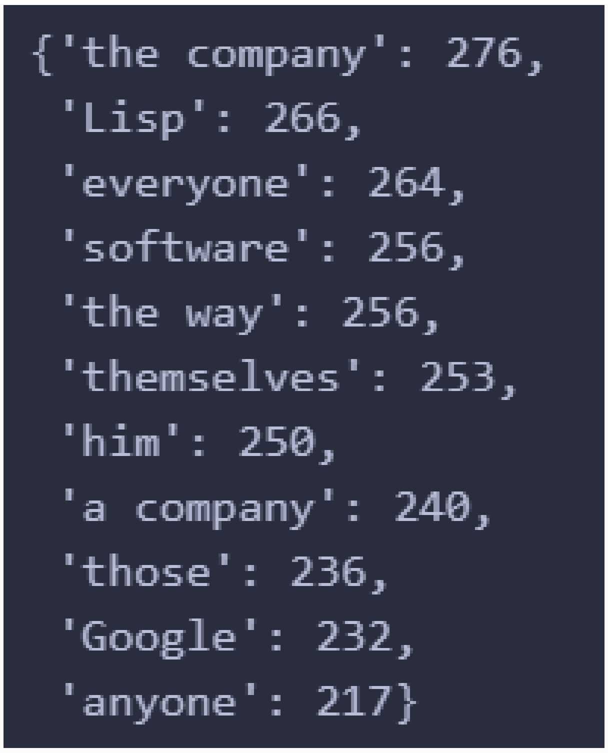
> To save the file successfully, we need set the configure of the spark session at its creation to avoid the error of exceeds of heap memory (shown below). Here my configuration are `("spark.executor.memory", "8g")` and `("spark.driver.memory", "8g")`
>
> 
>
> ```
> 24/04/14 03:11:30 WARN MemoryManager: Total allocation exceeds 95.00% (7,253,943,819 bytes) of heap memory
> Scaling row group sizes to 93.18% for 58 writers
> 24/04/14 03:11:30 WARN MemoryManager: Total allocation exceeds 95.00% (7,253,943,819 bytes) of heap memory
> Scaling row group sizes to 94.82% for 57 writers
> 24/04/14 03:11:30 WARN MemoryManager: Total allocation exceeds 95.00% (7,253,943,819 bytes) of heap memory
> Scaling row group sizes to 96.51% for 56 writers
> 24/04/14 03:11:30 WARN MemoryManager: Total allocation exceeds 95.00% (7,253,943,819 bytes) of heap memory
> Scaling row group sizes to 98.27% for 55 writers
> ```

(3)

In order to extract noun phrases from the paragraphs, referring to the `Spacy` given by the question, I define a `UDF` function called `extract_noun_phrases`, which can extract noun phrases by model `en_core_web_sm`.

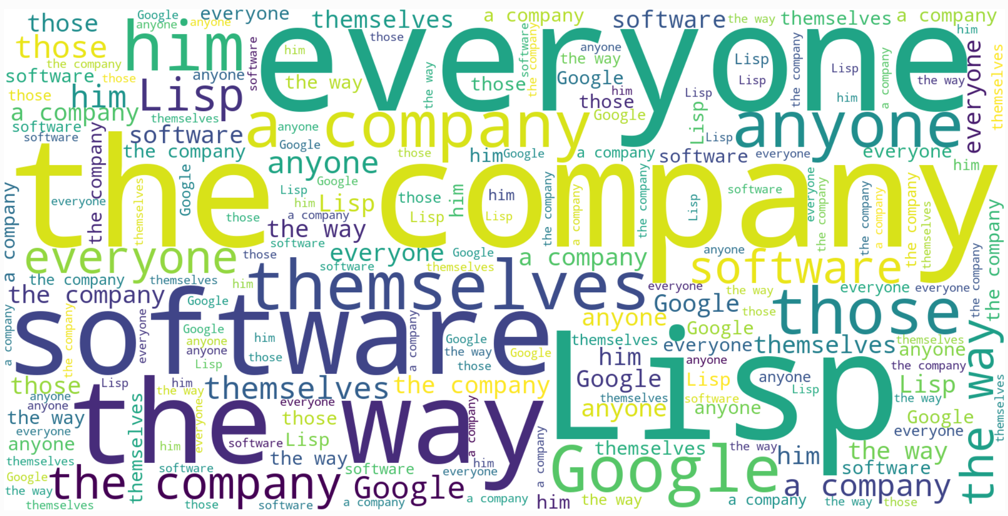
```python
nlp = spacy.load("en_core_web_sm")

def extract_noun_phrases(text):
    doc = nlp(text)
    noun_phrases = [chunk.text for chunk in doc.noun_chunks]
    return noun_phrases

spark.udf.register("extract_noun_phrases", extract_noun_phrases, ArrayType(StringType()))
```

After registering the `UDF`, I can use it in `flatMap` and get all noun phrases in one rdd. Then `CountByValue` and `sorted` by the count can be used to get top 40~50 noun phrases.



```
{'the company': 276,
 'Lisp': 266,
 'everyone': 264,
 'software': 256,
 'the way': 256,
 'themselves': 253,
 'him': 250,
 'a company': 240,
 'those': 236,
 'Google': 232,
 'anyone': 217}
```

Then the word cloud map can be plotted by `wordcloud`. As the parameter `max_words` exceeds the total number of words, the `repeat` is set to `True`.

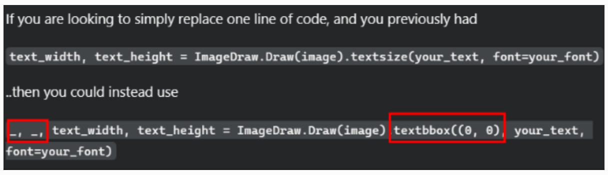


However, the plot generated by `wordcloud` is not that satifying. So I use `stylecloud` to plot the word cloud map, which is an advanced encapsulation API of `wordcloud`. The result is shown below.

SOFTWARE
THE WAY
LISP
THE COMPANY
EVERYONE
ANYONE
A COMPANY
GOOGLE

In fact, the api has some bugs, because it has not been maintained for many years, so I jump directly to his api source code and change it.

- if encountering an error that `'ImageDraw' object has no attribute 'textsize'`, that is because `ImageDraw.textsize()` called by it has been updated, and two places are needed to be changed in the source code, which is shown in the figure below (ref: **python - 'ImageDraw' object has no attribute 'textbbox' - Stack Overflow**)



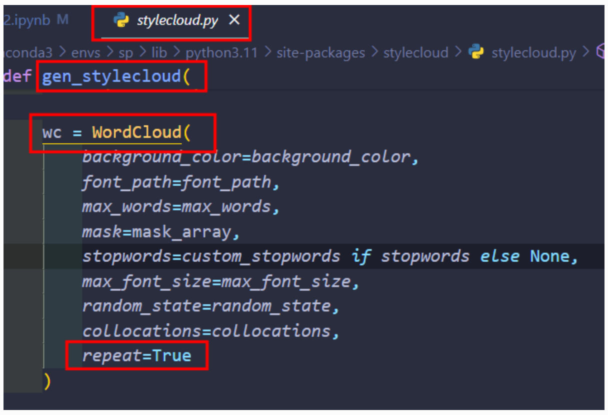
If you are looking to simply replace one line of code, and you previously had

```
text_width, text_height = ImageDraw.Draw(image).textsize(your_text, font=your_font)
```

..then you could instead use

```
_, _, text_width, text_height = ImageDraw.Draw(image).textbbox((0, 0), your_text,
font=your_font)
```

- If there are not enough words in the list, you want to repeat them when plotting. Just modify the `repeat` parameter in the source code, as it call the `wordcloud` function in the `wordcloud` package.



```
def gen_stylecloud(

    wc = WordCloud(
        background_color=background_color,
        font_path=font_path,
        max_words=max_words,
        mask=mask_array,
        stopwords=custom_stopwords if stopwords else None,
        max_font_size=max_font_size,
        random_state=random_state,
        collocations=collocations,
        repeat=True
    )
```

Appendix

A

All key phrases I used to match the paragraphy talking about career planning are as follows:

```
phrases_list = [
   "Career plan",
   "suggestion for",
   "suggestion of",
   "suggestion about",
   "Career development",
    "Skill development",
    "Career goal",
    "Job market",
    "Career mentor",
    "Career counsel",
    "Career transition",
    "Self-assessment",
    "Career prospect",
    "Industry research",
    "Resume and cover letter",
    "Interview preparation",
    "Professional networking",
    "Continuing education",
    "Job satisfaction",
    "Work life balance",
    "Career success",
    "Professional ethic",
    "Career growth"
    "Find startup",
    "Generat startup",
    "Com up zip_with",
    "Value of initial",
    "Start point",
    "Partial solution",
    "Max future option",
    "Work on new technolog",
    "Conversation with friend",
    "University research startup",
    "Importance of collaboration and friendship",
   "Mind wander and idea generation",
    "Mak thing eas to use",
   "Mak something people want",
    "Redefin problem",
    "Mak thing cheaper and commoditiz",
   "Mak thing easier",
   "Design for exit strateg",
   "Product development on spec",
   "Counteract monopol",
   "Accident startup",
   "Do what hacker enjoy"
]
```