Preprocessing Part:
Text Processing using collected Twitter Data

Hydrator provides a json file as well as a CSV after getting information out of the collected Tweet IDs. Using those files there should be a preprocessing part because Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Choosing R or Python to do the Preprocess:

R and Python are both open-source programming languages with a large community. R is mainly used for statistical analysis while Python provides a more general approach to data science. R and Python are state of the art in terms of programming language oriented towards data science. Python is better for data manipulation and repeated tasks, while R is good for ad hoc analysis and exploring datasets.

Python

```
import json

with open('covid19_tweets.json', 'r') as f:
    line = f.readline() # read only the first tweet/line
    tweet = json.loads(line) # load it as Python dict
    print(json.dumps(tweet, indent=4)) # pretty-print
```

The key attributes are the following:

- text: the text of the tweet itself
- created_at: the date of creation
- favorite_count, retweet_count: the number of favorites and retweets
- favorited, retweeted: boolean stating whether the authenticated user (you) have favourited or retweeted this tweet
- lang: acronym for the language (e.g. “en” for english)
- id: the tweet identifier
- place, coordinates, geo: geo-location information if available
- user: the author’s full profile
- entities: list of entities like URLs, @-mentions, hashtags and symbols
- in_reply_to_user_id: user identifier if the tweet is a reply to a specific user
- in_reply_to_status_id: status identifier id the tweet is a reply to a specific status

As you can see there’s a lot of information we can play with. All the *_id fields also have a *_id_str counterpart, where the same information is stored as a string rather than a big int (to avoid overflow problems). We can imagine how these data already allow for some interesting analysis: we can check who is most favorited/retweeted, who’s discussing with who, what are the most popular hashtags and so on. Most of the goodness we’re looking for, i.e. the content of a tweet, is anyway embedded in the text, and that’s where we’re starting our analysis.

We start our analysis by breaking the text down into words. Tokenization is one of the most basic, yet most important, steps in text analysis. The purpose of tokenization is to split a stream of text into smaller units called tokens, usually words or phrases. While this is a well understood problem with several out-of-the-box solutions from popular libraries, Twitter data pose some challenges because of the nature of the language.

Tokenize a Tweet Text

The core component of the tokenizer is the regex_str variable, which is a list of possible patterns. In particular, we try to capture some emoticons, HTML tags, Twitter @usernames (@-mentions), Twitter #hashtags, URLs, numbers, words with and without dashes and apostrophes, and finally "anything else".

```
import re

emoticons_str = r"""
    (?:
        [:=;] # Eyes
        [oO\-]? # Nose (optional)
        [D\)\]\(\]/\\OpP] # Mouth
    )"""

regex_str = [
    emoticons_str,
    r'<[^>]+>', # HTML tags
    r'(?:@[\w_]+)', # @-mentions
    r"(?:\#+[\w_]+[\w\'_\-]*[\w_]+)", # hash-tags
    r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+', # URLs

    r'(?:(?:\d+,?)+(?:\.?\d+)?)', # numbers
    r"(?:[a-z][a-z'\-_]+[a-z])", # words with - and '
    r'(?:[\w_]+)', # other words
    r'(?:\S)' # anything else
]

tokens_re = re.compile(r'('+'|'.join(regex_str)+')', re.VERBOSE | re.IGNORECASE)
emoticon_re = re.compile(r'^'+emoticons_str+'$', re.VERBOSE | re.IGNORECASE)

def tokenize(s):
    return tokens_re.findall(s)

def preprocess(s, lowercase=False):
    tokens = tokenize(s)
    if lowercase:
        tokens = [token if emoticon_re.search(token) else token.lower() for token in
tokens]
    return tokens

tweet = " ABCD "
print(preprocess(tweet))
```

The tokenizer is probably far from perfect, but it gives you the general idea. The tokenization is based on regular expressions (regexp), which is a common choice for this type of problem. Some particular types of tokens (e.g. phone numbers or chemical names) will not be captured, and will be probably broken into several tokens. To overcome this problem, as well as to improve the richness of the pre-processing pipeline, so we can improve the regular expressions, or even employ more sophisticated techniques like Named Entity Recognition.

The regular expressions are compiled with the flags re.VERBOSE, to allow spaces in the regexp to be ignored (see the multi-line emoticons regexp), and re.IGNORECASE to catch both upper and lowercases. The tokenize () function simply catches all the tokens in a string and returns them as a list. This function is used within preprocess (), which is used as a pre-processing chain: in this case we simply add a lowercasing feature for all the tokens that are not emoticons.

R

Text Processing using R

With the increasing importance of computational text analysis in research, many researchers face the challenge of learning how to use advanced software that enables this text analysis. Currently, one of the most popular environments for computational methods and the emerging field of "data science" is the R statistical software. However, for researchers that are not well-versed in programming, learning how to use R can be a challenge, and performing text analysis in particular can seem daunting.

Importance of R

R was specifically designed for statistical analysis, which makes it highly suitable for data science applications. Although the learning curve for programming with R can be steep, especially for people without prior programming experience, the tools now available for carrying out text analysis in R make it easy to perform powerful, cutting-edge text analytics using only a few simple commands. One of the keys to R's explosive growth has been its densely populated collection of extension software libraries, known in R terminology as packages, supplied and maintained by R's extensive user community. Each package extends the functionality of the base R language and core packages, and in addition to functions and data must include documentation and examples, often in the form of vignettes demonstrating the use of the package. The best-known package repository, the Comprehensive R Archive Network (CRAN), currently has over 10,000 packages that are published.

Convert this extracted data to a data frame which makes it more readable and easier to work with.

```
covid19_tweets <- twListToDF(tweets)

View(covid19_tweets)
```

Below is a code to pre-process the data and remove tabs, blank spaces, links etc. This section can be modified according to one's requirements.

```
#convert all text to lower case
covid19_tweets <- tolower(covid19_tweets)

# Replace blank space ("rt")
covid19_tweets <- gsub("rt", "", covid19_tweets)

# Replace @UserName
covid19_tweets <- gsub("@\\w+", "", covid19_tweets)

# Remove punctuation
covid19_tweets <- gsub("[[:punct:]]", "", covid19_tweets)

# Remove links
covid19_tweets <- gsub("http\\w+", "", covid19_tweets)

# Remove tabs
covid19_tweets <- gsub("[ |\t]{2,}", "", covid19_tweets)

# Remove blank spaces at the beginning
covid19_tweets <- gsub("^ ", "", covid19_tweets)

# Remove blank spaces at the end
covid19_tweets <- gsub(" $", "", covid19_tweets)
```

Stop Words

When working with text mining applications, we often hear of the term "stop words" or "stop word list" or even "stop list". Stop words are basically a set of commonly used words in any language, not just English.

The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead.

Stop words are generally thought to be a **"single set of words"**. It really can mean different things to different applications.

```
#clean up by removing stop words
covid19_tweets.text.corpus <- tm_map(covid19_tweets.text.corpus,
function(x)removeWords(x,stopwords()))
```

pre-processing the data is done, and ready to do some analysis.

Word Clouds

Word clouds (also known as text clouds or tag clouds) work in a simple way: the more a specific word appears in a source of textual data (such as a speech, blog post, or database), the bigger and bolder it appears in the word cloud.

Generated some word clouds and found out some of the frequent and important terms being used in the tweets we have extracted.

```
library("wordcloud")
#generate wordcloud
wordcloud(covid19_tweets.text.corpus,min.freq = 10,colors=brewer.pal(8,
"lockdown"),random.color = TRUE,max.words = 500)
```

Discussion & Decision

Many researchers and scholars use R for experimenting with data science. Many popular books and learning resources on data science use R for statistical analysis as well.

Data wrangling is the process of cleaning messy and complex data sets to enable convenient consumption and further analysis. This is a very important and time taking process in data science. R has an extensive library of tools for database manipulation and wrangling.

Data visualization is the visual representation of data in graphical form. This allows analyzing data from angles which are not clear in unorganized or tabulated data. R has many tools that can help in data visualization, analysis, and representation.

R is a language designed especially for statistical analysis and data reconfiguration. All the R libraries focus on making one thing certain – to make data analysis easier, more approachable and detailed. Any new statistical method is first enabled through R libraries. This makes R a perfect choice for data analysis and projection.

R will be the technology which I will use to continue with the data analysis.