Web Scrapping Module:

The web scrapping module built will be different for different websites and has to be revamped for the website which is chosen for data extraction. It is because the way the html pages are structured and laid out is different for different websites.

I am demonstrating how a web Scrapping module will work in general for medHelp website for section Cancer. I suggest using Beautiful soup package for the scrapping of the data. This library lets access the html tags and its contents directly.

Technique suggested:

- First, we take a sample page from the website whose content needs to be extracted eg: (https://medhelp.org/forums/Leukemia-and-Lymphoma-/show/139).
- Try to understand the basic structure that the html page follows. This can be done wither by traversing through the html tags using soup.prettify() or through the inspect section in the chrome browser.
- Each headline of the comments has to be clicked in this case to be able to read the whole content. So, first extract all the headlines text along with the link for the whole message. For this, we figure out that each comment is within a tag <h2 class="subj_title">. We extract all these h2 tags from the html page. Followed by this we extract the <a> within this <h2>. From this <a> the text and the attribute value of href is extracted. Further, we maintain a dictionary with the key value as the headline and the value as the href link. This would facilitate navigating through all the comments.
- We also determine the no of pages associated through the bottom navigation bar and loop through the different pages. We figure out that the pattern of the page url is https://medhelp.org/forums/Leukemia-and-Lymphoma-/show/139?page=1. So, we loop over the number of pages to extract and keep populating the dictionary.
- Once, the dictionary has been populated out task is to make use of the indivisual comment full page link to extract the comment data and the no of responses that comment has received.
- Traverse through the dictionary link pages one by one.

For every page:

- Identify that the main comment content is present in a div with id "subject_msg". Therefore, use beautiful soup soup.find() method to extract the text. Further, the text contains some extra set of characters. Remove these by removing any special characters present in the text.
- Further, we also wish to figure out the no of responses that the person has received on his comment. For this, figure out that the responses button correspond to <a> with id "read_answer_scroll". Extract the span tag within this anchor tag and get its text to find the no of responses.
- Repeat the above process for all the comments present in the dictionary. This way we have extracted the indivisual's headline, his overall comment and the no of interactions involved. We will then convert this data to a pandas dataframe and also save it to disk for future use.

Sample Code for Web Scrapping for MedHelp:

```
from bs4 import BeautifulSoup
from urllib.request import Request, urlopen
import pandas as pd
import numpy as np
import re

#key:header
#value:href
dicta={}
flag=0
def dictionaryCall(soup):
    #get all the anchor tags
    headers=soup.find_all("h2", {"class": "subj_title"})
    if (headers!=[]):
        flag=0
        for all in headers:
            anchor=all.find_all("a")
            for x in anchor:
                dicta[x.text]=x.attrs['href']
    else:
        flag=1
    return (flag)

for x in range(1,5):
    print(x)
    req = Request("https://medhelp.org/forums/Leukemia-and-Lymphoma-/show/139?page="+str(x),
headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(req).read()
    soup = BeautifulSoup(webpage)
    #print (soup.prettify())
    flag=dictionaryCall(soup)
    if(flag==1):
        break

#using the dictionary to navigate to indivisual comment pages

text=[]
responses_list=[]
import re
for headline, link in dicta.items():
    print (headline)
    req = Request("https://medhelp.org"+link, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(req).read()
    soup = BeautifulSoup(webpage)
    divs=soup.find("div",{"id":"subject_msg"}).text
    comment = re.sub('\W+',' ', divs )
    responses=soup.find("a",{"id":"read_answer_scroll"})
    noOfResponses=int(responses.find("span").text)
    text.append(comment)
    responses_list.append(noOfResponses)
df=pd.DataFrame(data={"headline":list(dicta.keys()),"comment":text,"responses":responses_list})
```

NLP Module:

1) **Extracting the Engagement level**: We have already extracted the no of responses received by the individual in our web scrapping module. Using all the values in the responses column we will calculate the percentile values to see the minimum and the maximum no of responses received. We will cap the upper or the lower boundary in case of any outliers. Based on the modified range that we receive after capping we decide thresholds for low, medium and high thresholds. Based on these threshold values we tag each comment as having low, medium or high level of engagement.

   Further, before we start with the NLP Process we would require some amount of data cleaning:

   - Converting all the data to lower case to avoid treat the same word multiple times.
   - Removing the punctuations.
   - Converting the sentences to tokens.
   - Applying spell correction on the tokens.
   - Stemming/Lemmatization.

2) **Identifying the disease name:** The disease name identification can be done using NER (Named Entity Recognition). But the model needs to be trained on the biomedical dataset which requires huge computational power. Thus, we can utilize the concept of transfer learning for the same. There has been a lot of recent work in the area of biomedical text mining eg: https://academic.oup.com/bioinformatics/article/33/14/i37/3953940.

   In this paper, a LSTM-CRF Model has been built to do NER on the disease, gene and species. The model makes use of the word embedding's trained as per Word2Vec but using different corpora more suitable for the biomedical domain. It is important, as the biomedical vocabulary is significantly different from the general English vocabulary.

   In this model, bi-directional LSTM's are used to capture the dependencies from both the forward and the backward direction. As stated in the paper:

   It is comprised of three main layers. The first layer is the embedding layer. It receives the raw sentence S made of the sequence of words $w1,w2,\ldots,wT$ as its input and produces an embedding (i.e. a dense vector representation) $\boldsymbol{x}1,\boldsymbol{x}2,\ldots,\boldsymbol{xT}$ for each word in S. The embedding vector $\boldsymbol{xt}$ of word w_t is a concatenation of a word- and a character-level embedding. The word-level embedding is simply retrieved from a lookup-table of word embeddings trained on a large corpus. The character-level embedding is obtained by applying a bi-directional LSTM to the character sequence of each word and then concatenating the last activations of both directions, The resulting sequence of embeddings $\boldsymbol{x}1,\boldsymbol{x}2,\ldots,\boldsymbol{xT}$ is fed into another bi-directional LSTM-layer that produces a refined representation of the input, which is the input to a final CRF-layer. The classical Viterbi algorithm is used to obtain the final output from this layer. All components together form a single fully differentiable neural network that can be trained by backpropagation.

We can use the weights of this model to train our own NER model. For this we would require pre labelling few 1000's of our sentences with the disease name entities we want to identify. And the trained model can then be used to predict the entities on the unseen dataset.

3) **Identifying the Disease Status:** We can use the text classification model to classify whether the comments classify into the state of discovered, progressing or deceased and NA. For this purpose, we need to prepare the labelled dataset for a few thousand sentences.
   As, we have paragraphs with a lot of comments we will classify each sentence individually and then take the majority voting. Also, we will have to use some word embedding's such as word2Vec or glove. We can simply avoid the words not found in the general English corpus. It is because the words associated with status will be general English words such as growing, discovered, founded . Also, we avoid using lemmatization/stemming for this data as this information will let us discover the time and association of the disease with the patient. Random Forest or Xgboost can be used to do this text classification.

4) **Treatment Effectiveness:** The overall sentiment of the comment can be extracted through sentiment analysis. Each sentence in the comment can be passed to the sentiment analysis module to get the overall expression of the sentence. And then the majority voting can be done for separate sections of the comment eg: the upper section, middle section and the conclusion section where these sections will be divided on the basis of the no of sentences in the comment. This should be done as generally a patient comment starts with negative sentence and if the treatment has been been effective it will converge to positive.

5) **Age of the patient:** In most cases age of the patient has been specified in the comment. This can be extracted by a rule based model. Anything following "age of", preceding "years","yr","year" and is a number can be extracted. Also, look for words like "teenager", "young", "old", "middle-aged man". Using these numbers or discrete categories the data can be parsed a little to get discrete categories of young, old, kid, etc.

Segmentation Module:

Once the features have been extracted from the unstructured data using the scrapping and the NLP module it can be segmented to get the various patient profiles.

One thing that is important in this case is that the features extracted are not numbers but mostly strings eg: disease name, disease status, etc. So, this needs to be handled before we can apply any segmentation algorithm.

1) Engagement level: This feature is already numerical. It can be normalized between 0 to 1 to get smaller valued feature. This will reduce the computation time for the algorithm.
2) Disease Name: Depending upon the cardinality of this column we can adopt a way to convert this feature from discrete to numbers in some way.
   If the cardinality is a smaller number then one hot vectors can be generated for this column.
   If the cardinality is big then we can use the word vectors from the biomedical data to represent the disease (the same, as we will use in the NLP module).
3) Disease Status, treatment Effectiveness, Age: These columns have few categories, so convert these to discrete numbers.

These features can then be passed to some clustering algorithm line K-Means, K-Means ++ to generate clusters. We can use hierarchical or DB-Scan as well but I am assuming that the size of the data will be very large so K-Means++ will be faster. Also, we would want the number of clusters to be moderate, not very large in number otherwise it becomes difficult for the business team to handle.

Once the clusters have been generated we would first figure out the quality of the clusters by some statistical measure eg: within cluster sum of sqauares / silhouette score.
But what is more important is profiling these clusters for some story.
Eg: we got one cluster where mostly the patients had discovered the disease and their number of engagement were on the higher side.
On the other hand, the patients whose treatment has been ineffective there engagement level was less (may be because nobody could help them in this case) . These are just hypothetical examples based on how stories can be once we profile.

Also, sometimes we find that on clustering on multiple features we might not get any story. Then we will have to drop some features and cluster again. But then the profiling will be done on all features till we get a relevant story from the segmentation.