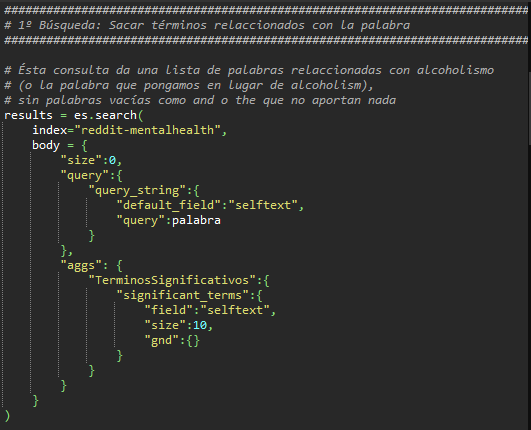
**Ejercicio 1**

El objetivo del ejercicio 1 es obtener documentos que traten sobre un determinado tema que introducirá el usuario (obviamente sin tener que contener esa palabra necesariamente). Para las pruebas de esta práctica se ha utilizado el alcoholismo (alcoholism) aunque el programa soporta cualquier término.

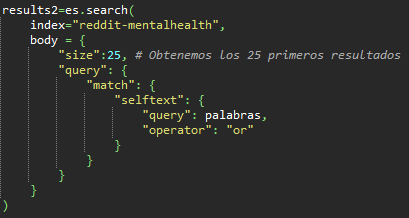
El script comienza haciendo una serie de consultas utilizando el parámetro gnd (Google normalized distance) para obtener palabras que se almacenarán en un set (se ha utilizado esa estructura de datos para que no se repitan las palabras). Las palabras más relacionadas con la palabra que ha introducido el usuario es lo que contendrá el set, que se imprimirá por pantalla al ejecutar el script.





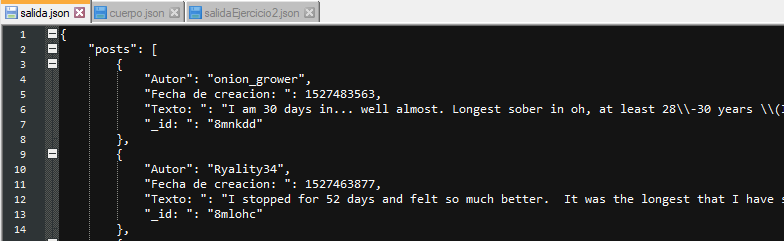
El resultado (results) está en formato JSON.

Una vez sacados los términos, lo que se hace es simplemente pasarlos a un string (palabras) para poder introducirlos en una consulta que nos devolverá los 25 posts que más de esas palabras coincidan.



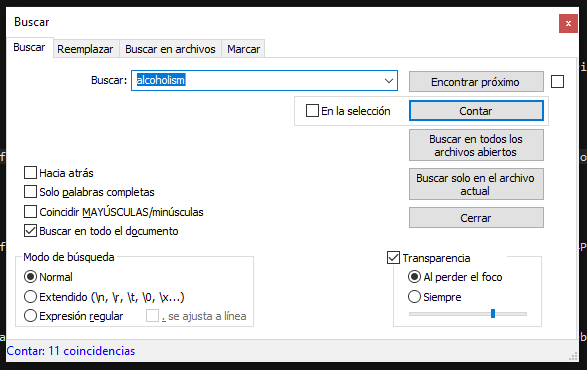
Una vez que ya tenemos los términos ya es cuestión de utilizar la librería json para crear el fichero de salida.

**Análisis del fichero de salida y su precisión:**



Una vez examinados los 25 documentos del fichero de salida, se comprueba que la precisión es del 100% ya que no hay ningún documento que no hable sobre un problema con la bebida.

Cabe destacar que hay bastantes documentos que no contienen la palabra que introdujo el usuario (alcoholism). De hecho, sólo hay 11 veces que aparece la palabra en el documento:



Para obtener la precisión del 100% de documentos relevantes, se han utilizado los siguientes valores:

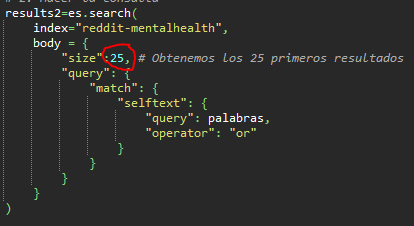
* Número de palabras que se obtienen en la primera extracción de términos relacionados con el alcoholismo: 10



(En las siguientes dos expansiones también se utiliza 10. Al final no salen 30 porque obviamente se repiten términos y acaban saliendo unos 20 distintos para el término alcoholism).

Si se aumenta el número de términos a 100 por ejemplo empiezan a salir palabras que no tienen realmente mucho que ver con el alcoholismo, y por tanto los posts que se recuperan no tratan sobre el alcoholismo en su totalidad.

* Número de documentos que se retornan: 25



Se ha probado a retornar 100 documentos pero ya había alguno que no trataba de alguien que está hablando del alcoholismo, así que se dejó en 25 para lograr una precisión del 100% de documentos relevantes

**Ejercicio 2**

Las consultas MLT (More like this) por debajo funcionan de la siguiente manera:

* Se les pasa como parámetros una serie de documentos (ya sea por ejemplo en forma de un \_id de un documento que se encuentra en el índice, en este caso reddit-mentalhealth, o en forma de texto plano)
* Obtienen de ese documento o serie de documentos una serie de palabras representativas de esos documentos.
* A partir de esa serie de palabras, hacen una consulta que devuelve los documentos que más de esas palabras coincidan.

Por tanto, MLT lo que hace por debajo es una expansión de términos, pero en lugar de hacerla a partir de un término como en el ejercicio 1, lo hace a partir de una serie de documentos.

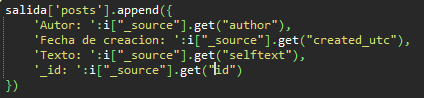
Para emular la expansión de términos del ejercicio 1 con MLT se puede coger uno o varios documentos obtenidos en la salida del ejercicio 1 (hablan sobre el alcoholismo, o la palabra que introduzca el usuario) y pasárselos como entrada a la consulta MLT para que saque otra serie de documentos relacionados.

Los términos que la consulta MLT ha considerado representativos no aparecen porque es un proceso interno, lo que nos devuelve la consulta MLT son los documentos relacionados.

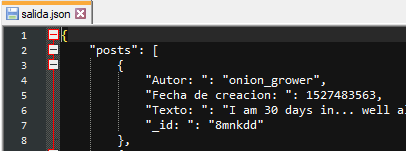
No hay problema porque el objetivo del ejercicio no es saber qué términos cree la consulta MLT que son representativos de los documentos de entrada, sino simplemente obtener los documentos de salida.

Por tanto, los pasos a seguir en este ejercicio son los siguientes:

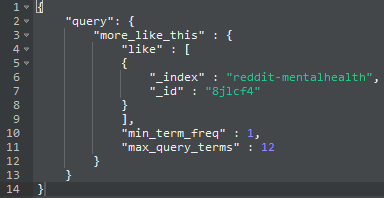
* Obtener del archivo JSON generado por el ejercicio 1 los \_id de los documentos (Para ello hay que modificar ligeramente el ejercicio 1 para que también añada los \_id de cada documento, el objetivo es poder hacer el ejercicio 1 y el 2 para cualquier término, no solo para alcoholismo.)



Muestra del archivo generado por el ejercicio 1:

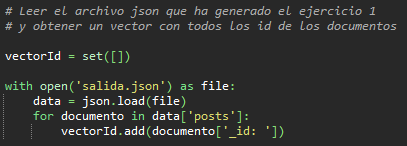


* Pasarle los \_id a la consulta MLT, un pequeño ejemplo:

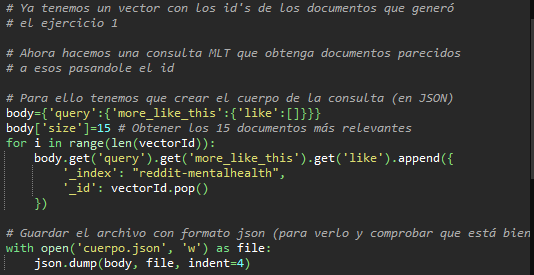


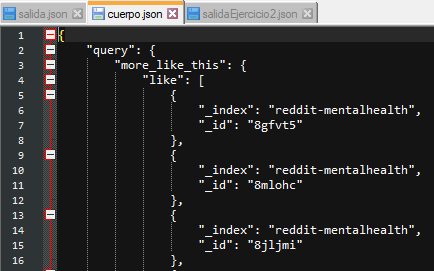
Esa consulta devuelve documentos parecidos al “8jlcf4”

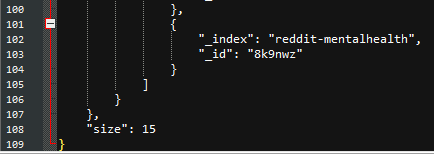
En el script lo que se hace es leer los id’s del fichero generado por el ejercicio1 mediante la librería json:



Una vez ya tenemos los id’s de los documentos, hay que crear el cuerpo de una consulta parecida a la que hay más arriba de ejemplo en cerebro. El cuerpo es en json:

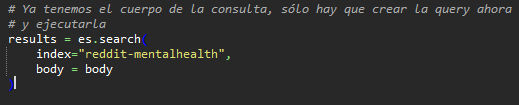






(Se le añade el parámetro size al final para que sólo retorne los 15 primeros archivos)

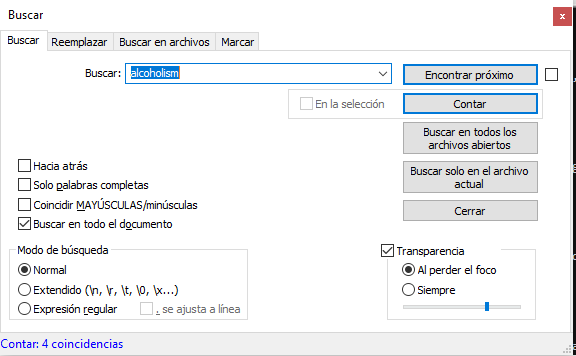
Como se puede ver en el archivo json que ha generado, es una consulta con el mismo formato que la que hay más arriba en cerebro. Ya solo es cuestión de ejecutarla:



* Una vez ya obtenidos los documentos que genera la consulta MLT hay que almacenarlos en un archivo JSON, lo cual se hace exactamente igual que en primer ejercicio, mediante la librería json se crea un archivo y se van añadiendo los documentos, no he creído conveniente explicar esta parte ya que el objetivo de este ejercicio es entender el funcionamiento de las consultas more like this.

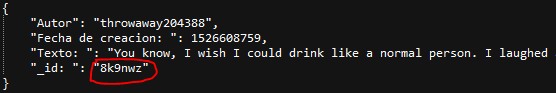
**Análisis del fichero de salida y su precisión:**

Mediante el parámetro “size”: 15 en el cuerpo de la consulta MLT (para que devuelva 15 documentos diferentes) se obtiene una precisión del 100% de documentos relevantes, con un total de 4 coincidencias de la palabra alcoholism:

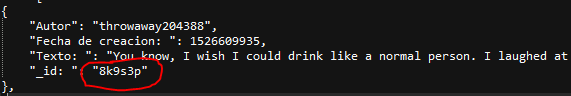


Cabe destacar que hay documentos que se repiten, pero es porque salen repetidos en el índice de reddit-mentalhealth, pero en realidad son posts distintos porque tienen un atributo \_id en el índice distinto, lo que me hace pensar que la persona ha copiado y pegado el contenido y publicado varios posts, probablemente en varios subforos, ya que también las fechas son distintas. Eso sí, en cada una de las salidas no se repiten, cada post de cada salida es diferente.

Por ejemplo:



En salida.json (la salida del ejercicio 1)

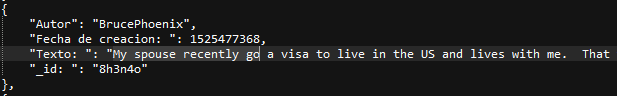


En salidaEjercicio2.json

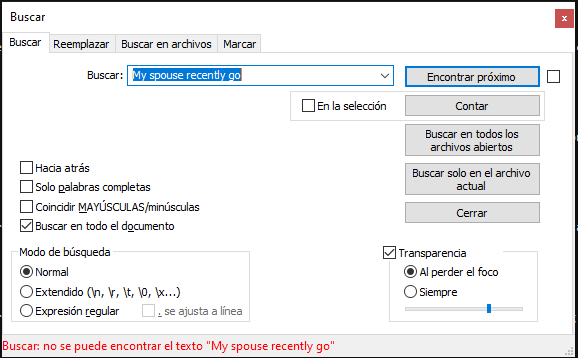
Esto se podría resolver controlando que los selftext sean distintos, pero realmente lo son, sólo cambia que hay algún carácter que cambia de los dos textos, así que sería bastante difícil diferenciar si un post es el mismo o no.

**Conclusión**

Por otra parte, la consulta more like this se podría considerar exitosa ya que realmente ha encontrado documentos que hablan sobre problemas con la bebida que no estaban en el primer documento, como por ejemplo el siguiente post:



No aparece en la salida del ejercicio 1 porque buscando las cuatro primeras palabras, no sale:



Y trata sobre el alcoholismo, habla de un amigo suyo que bebe mucho y tiene problemas.

**Ejercicio 3**

Se exponen los pasos seguidos para llegar a la solución.

1.- El ejercicio en cuestión es el siguiente:

Se desea obtener una lista exhaustiva de los medicamentos utilizados por los usuarios que

aparecen en la colección.

Describir de manera detallada los pasos seguidos para explorar la colección, cómo se ha

construido la consulta o consultas que han permitido llegar a una lista potencial y qué recursos

externos se han empleado para validar la información obtenida.

¡Atención! No es necesaria la validación automática de la lista de medicamentos obtenida pero

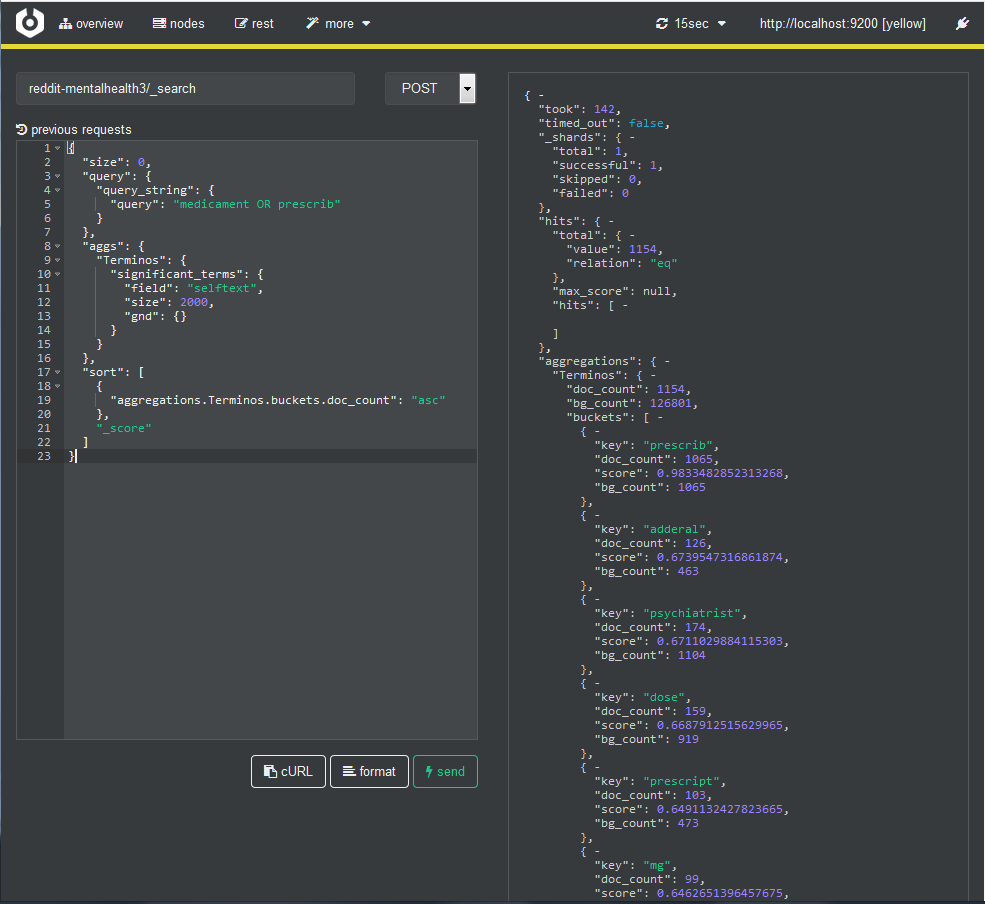
sí se valorará positivamente que se explore qué posibilidades de automatización existen al

respecto.

Si se usa algún conocimiento experto de partida debe indicarse claramente en la

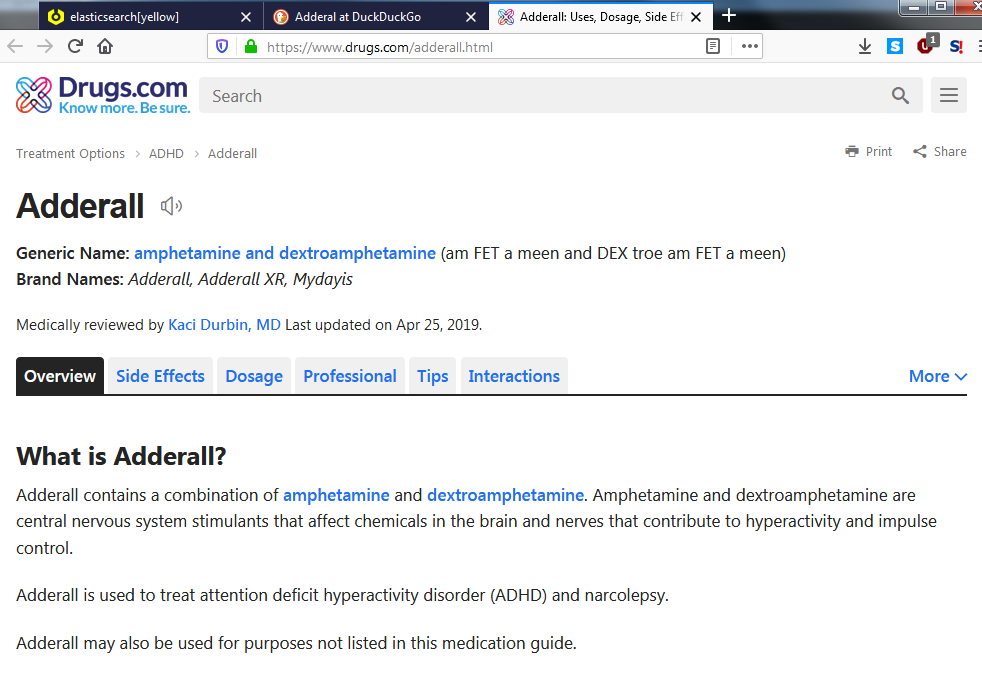
documentación.

2.- Se ha empezado por construir una pequeña consulta en elasticsearch en cerebro donde sacar las claves para luego pasarlo a python usando un script.



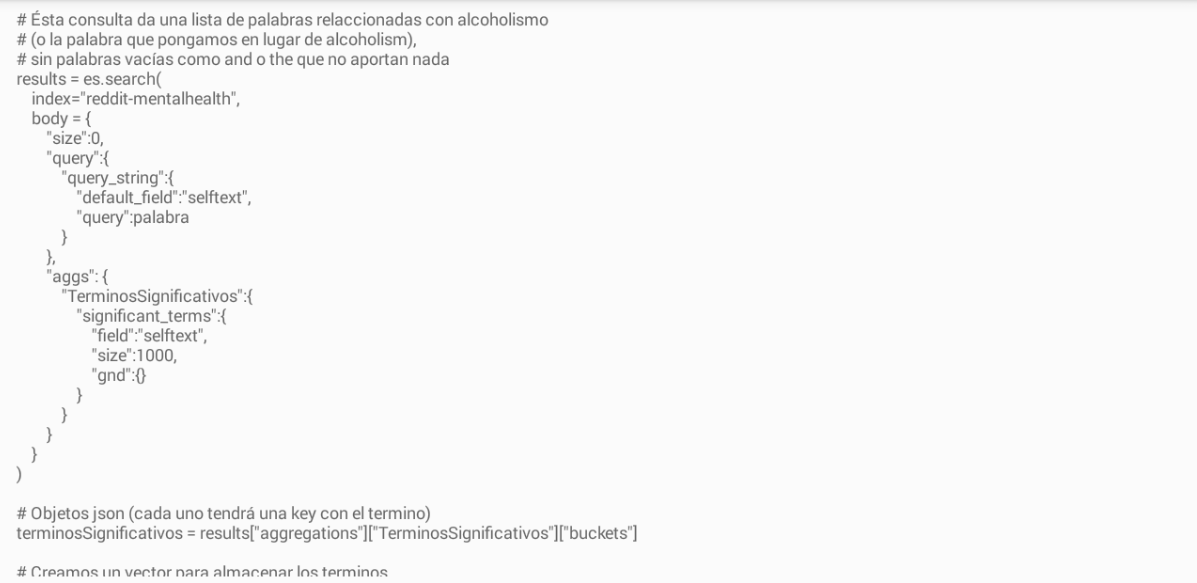
3.- Se ha buscado en google translate las palabras relacionadas con posible medicación, en español normalmente se recetan o se hace mención al medicamento, se ha construido la consulta en torno a eso

4.- Se ha buscado algún termino y comprobar que efectivamente, es una medicina, a lo largo de la documentación se ha usado Adderal, ya que por orden es de los primeros y sale como medicina en varias páginas.



5.- A partir de aquí, se ha construido una consulta en python, ayudada por un compañero de grupo, donde obtener los diferentes términos que aparecen en la consulta y guardarlos en un fichero. Se ha usado el entorno de PyScript por su facilidad para debuggear.

Código del compañero:

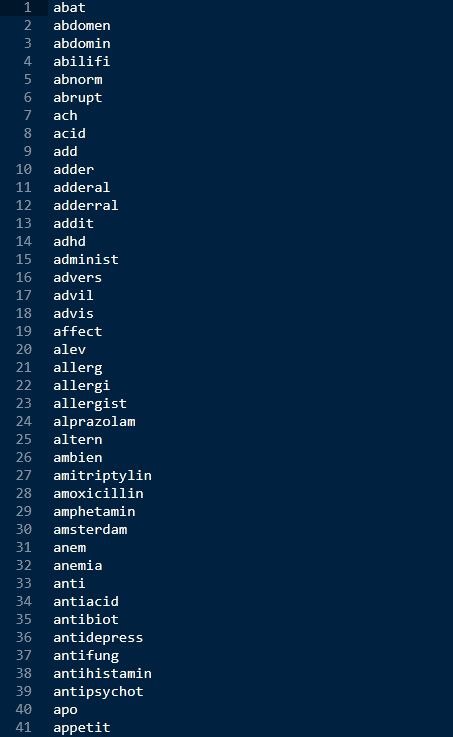


No se ha guardado una copia de la primera versión de este código, se pone un ejemplo aproximado:



6.- Tras abrir el resultado, se vio que aparecen las cantidades de la medicina, 10mg, 1g, 100mg… Como tiene fácil solución y solo esos términos tienen número, se añadió una sentencia if que comprueba si hay algún número en el término para que no se añadiese.



7.- Esta lista fue ordenada alfabéticamente y se fue comprobando diversos elementos para ver que algunos salían como medicinas (paracetamol, adderal, amoxicillin) y otros no (area, april, asian).

8.- En este punto se le pidió ayuda al profesor Daniel Gayo, quien amablemente me respondió con varias opciones

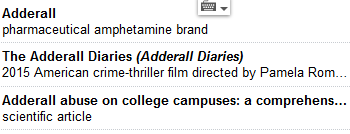
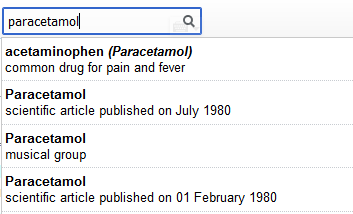
1 – Decir cómo se haría y enviar la lista tal cual

2 – Usar la API de wikidata

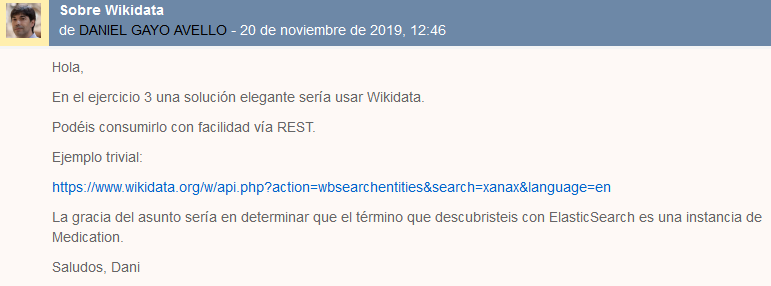
3 – Cruzar esta lista con otras que ya existen

Dado que había un pequeño ejemplo de uso y todavía se tenían algunos días hasta la entrega, se decidió tratar de usar la API de wikidata con Python

9.- Wikidata es una pagina que contiene elementos o palabras y se clasifican “según lo que son”. Por ejemplo, Paracetamol es un medicamento pero también un grupo musical. Adderal es un medicamento pero también tiene otras características.



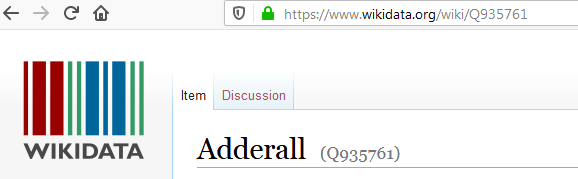
10.- A partir de aquí se uso el consejo de Daniel Gayo del foro, comprobar que el termino es una instancia de meditación



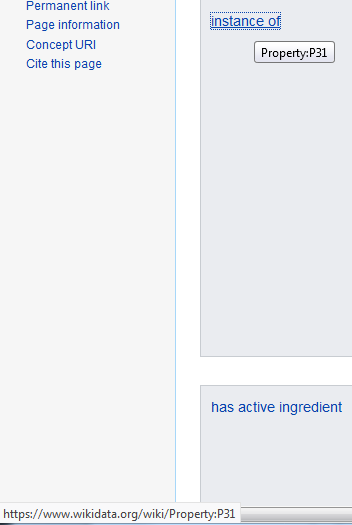
11.- Después de investigar, se conoció como trabaja Wikidata:

Wikidata tiene muchos datos y palabras, asigna a cada tipo de dato un identificador único, estos identificadores tienen propiedades, y las propiedades también están identificadas por su id. Ejemplo:

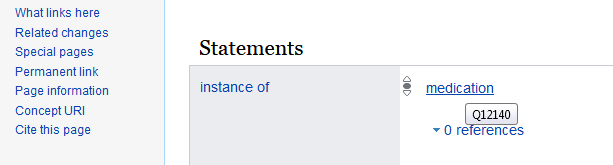
Adderal tiene de id Q935761 <https://www.wikidata.org/wiki/Q935761>



Tiene diferentes propiedades, una de ella instance of. Esta propiedad tiene el id P31, inicialmente se supo por la redirección del enlace (abajo del todo).



Dentro de esta propiedad, medicación tiene el id Q12140

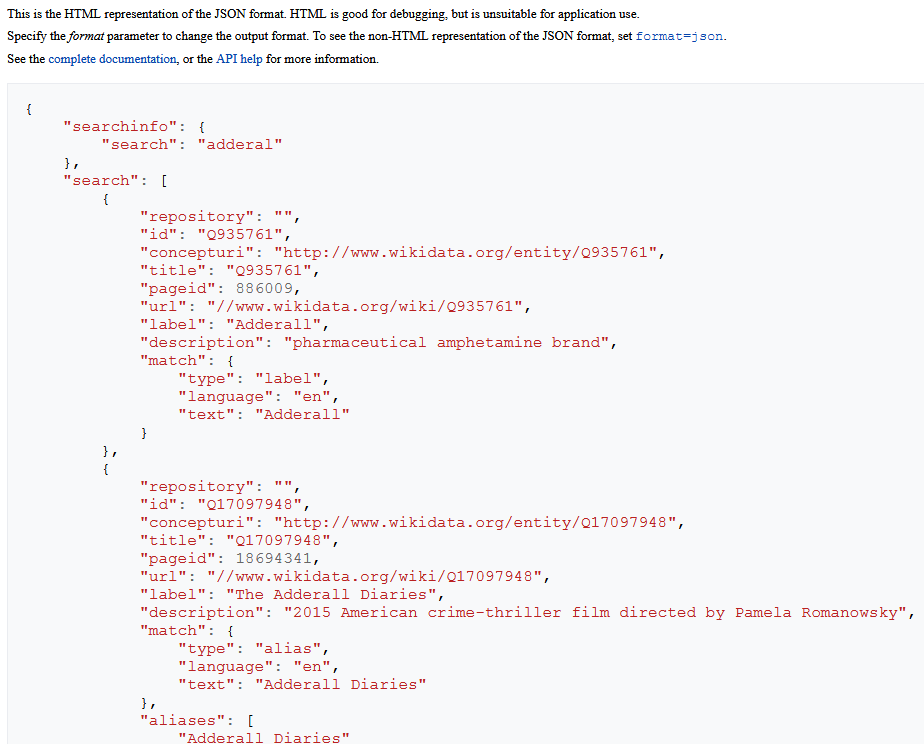


Resumiendo, se obtiene la id de un termino y luego se mira que tenga como propiedad P31 y dentro algún valor sea Q12140.

12.- Se estudió la API de wikidata para ver como forma los datos y de que forma se puede trabajar con ellos, primeramente de forma manual con el ejemplo de Adderal. El proceso manual que se siguió es como sigue:

Para averiguar la id se usa el ejemplo del foro, solo que en lugar de Xana se puso Adderal

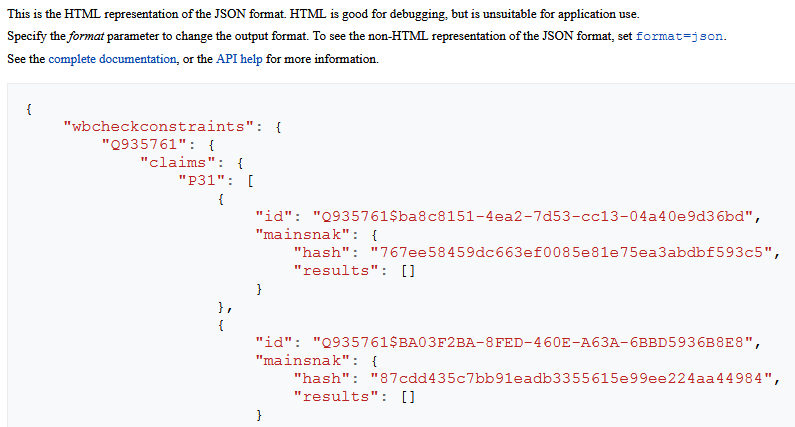
<https://www.wikidata.org/w/api.php?action=wbsearchentities&search=adderal&language=en>



Aparecen varias ids, la que nos interesa es la primera, Q935761.

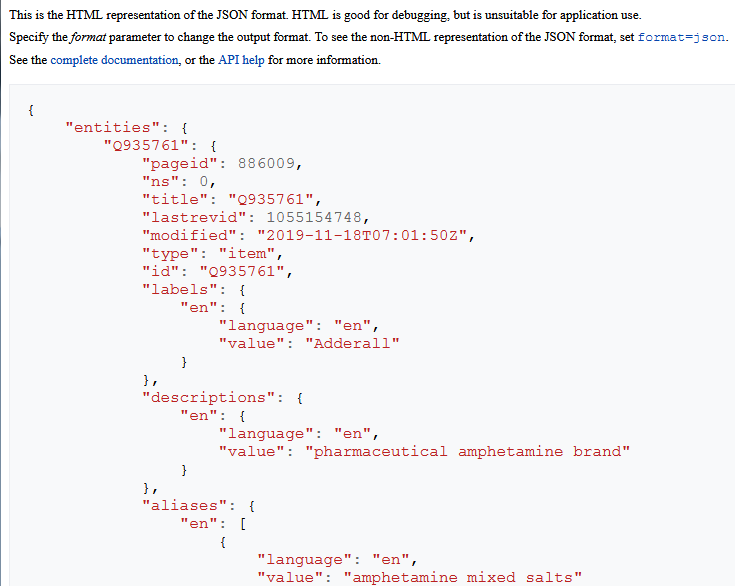
Con esta id, se averigua que constraints tiene, es decir, sus características.

<https://www.wikidata.org/w/api.php?action=wbcheckconstraints&id=Q935761>

Y aquí falla, ya que no se puede saber si es un medicamento o no a través de este tipo de consulta, el hash es único y no hay ningún dato que identifique a todos los medicamentos.

13.- Se siguió investigando y se encontró otra función de la API que si tiene los datos que se necesitan. Wbgetentities

<https://www.wikidata.org/w/api.php?action=wbgetentities&ids=Q935761&languages=en>



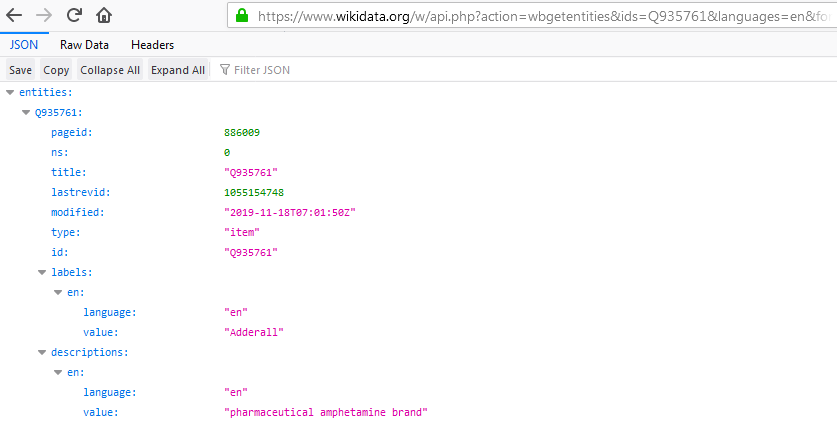
Este es el comienzo, la parte que nos interesa tiene que ver con P31 (propiedad “instance of”). Se busca…



Y sale que tiene la id que buscamos, Q12140, y es única porque wikidata no repite ids, es mencionado en su manual.

14.- Entonces el proceso manual cambia a – se tiene un término → se convierte a una id→ se mira si tiene P31 y un valor de id Q12140, el paso que falta es automatizarlo con un script.

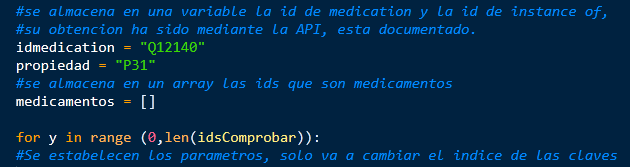
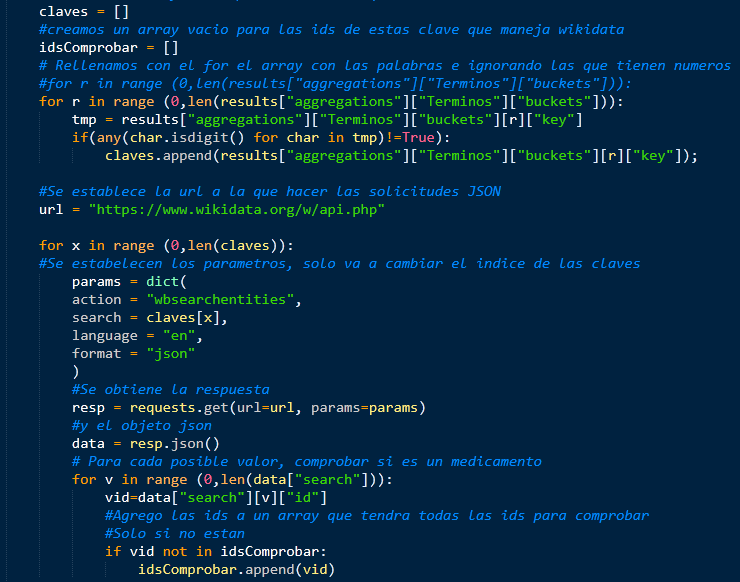
15.- Se investiga algo más profundamente en como trabajar con JSON en python, también se fija en que la parte superior de cada solicitud tiene un format=json, se obtiene un json con exactamente el contenido html



16.- La forma más sencilla de hacer solicitudes a la web es usando la librería requests de python, con un ejemplo sencillo de StackOverflow se pudo adaptar a este caso:



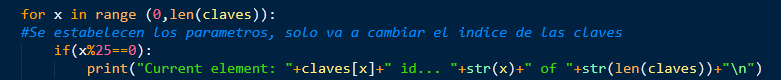
17.- Se altera el código añadiendo diferentes arrays para la id, las propiedades y la lista de medicinas final. Todo el código esta comentado para facilitar su comprensión



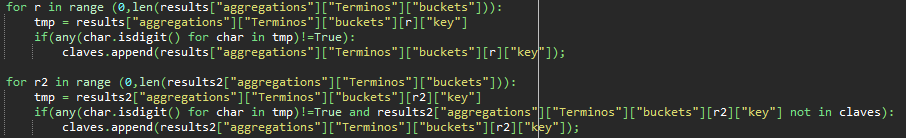
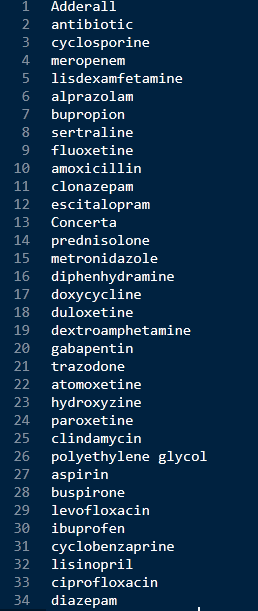
…

Recorre los arrays y hace consultas a la api de wikidata explicada anteriormente, al final guarda los datos en el fichero. Como no se sabe si el medicamento se corresponde a la primera id (ej, adderal es Q935761) u otras (adderal también tiene la id de Q17097948), hay que añadir y comprobarlas todas.

18.- Es un proceso bastante lento y puede parecer que este bloqueado, se decidió poner a modo de barra de proceso, una información de cuanto le queda y por donde va:



19.- Una vez finalizado el programa, crea un fichero medicamentos.txt con la lista de medicamentos. Se confirmó que el programa funciona.



20.- A partir de aquí se puede obtener más o menos exhaustividad que se desee con cualquier número de consultas, hay que tener en cuenta que a mayor exhaustividad, mayor duración ya que tiene que ver y comprobar más términos.

**Ejercicio 4**

En este ejercicio se nos pide encontrar factores comórbidos relativos a la ideación suicida y a las conductas autolesivas.

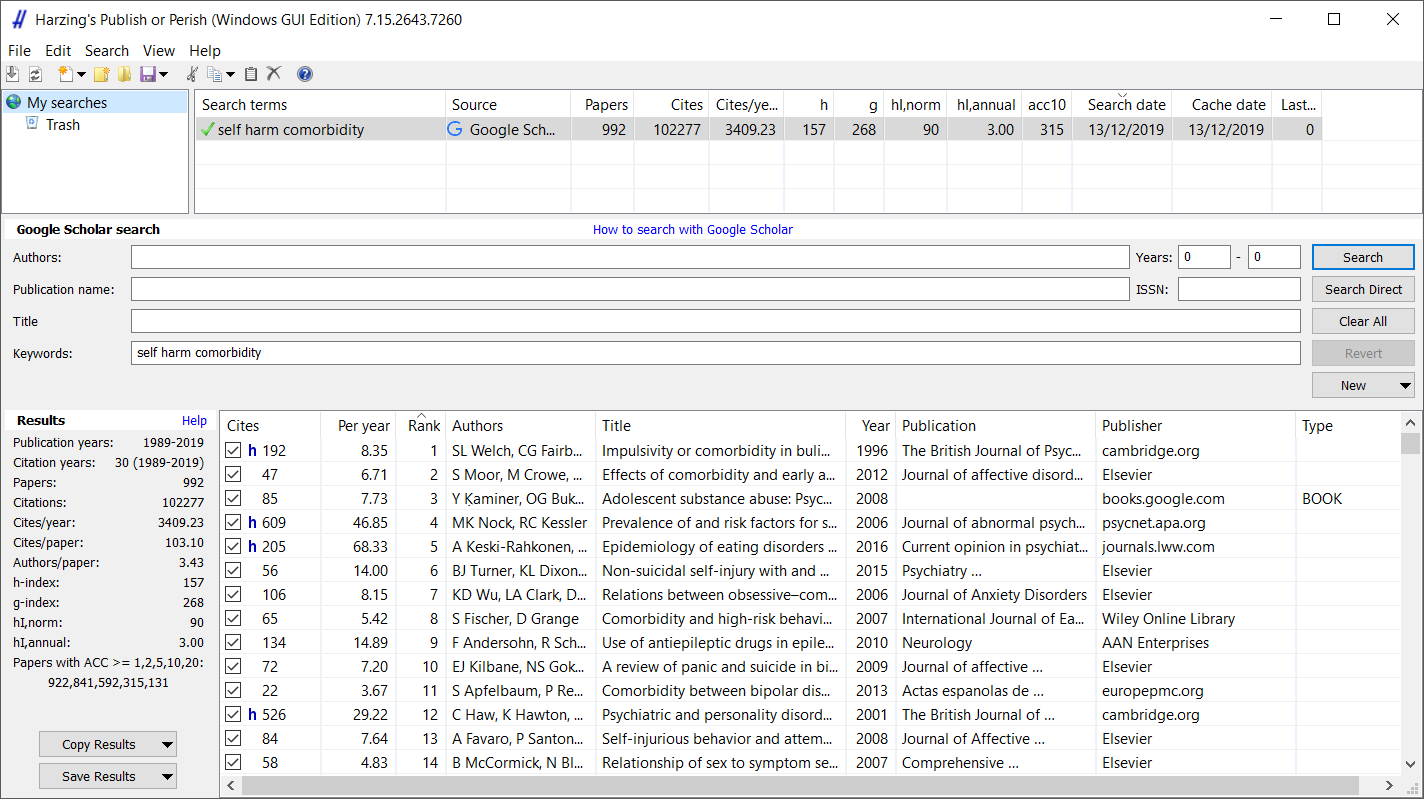
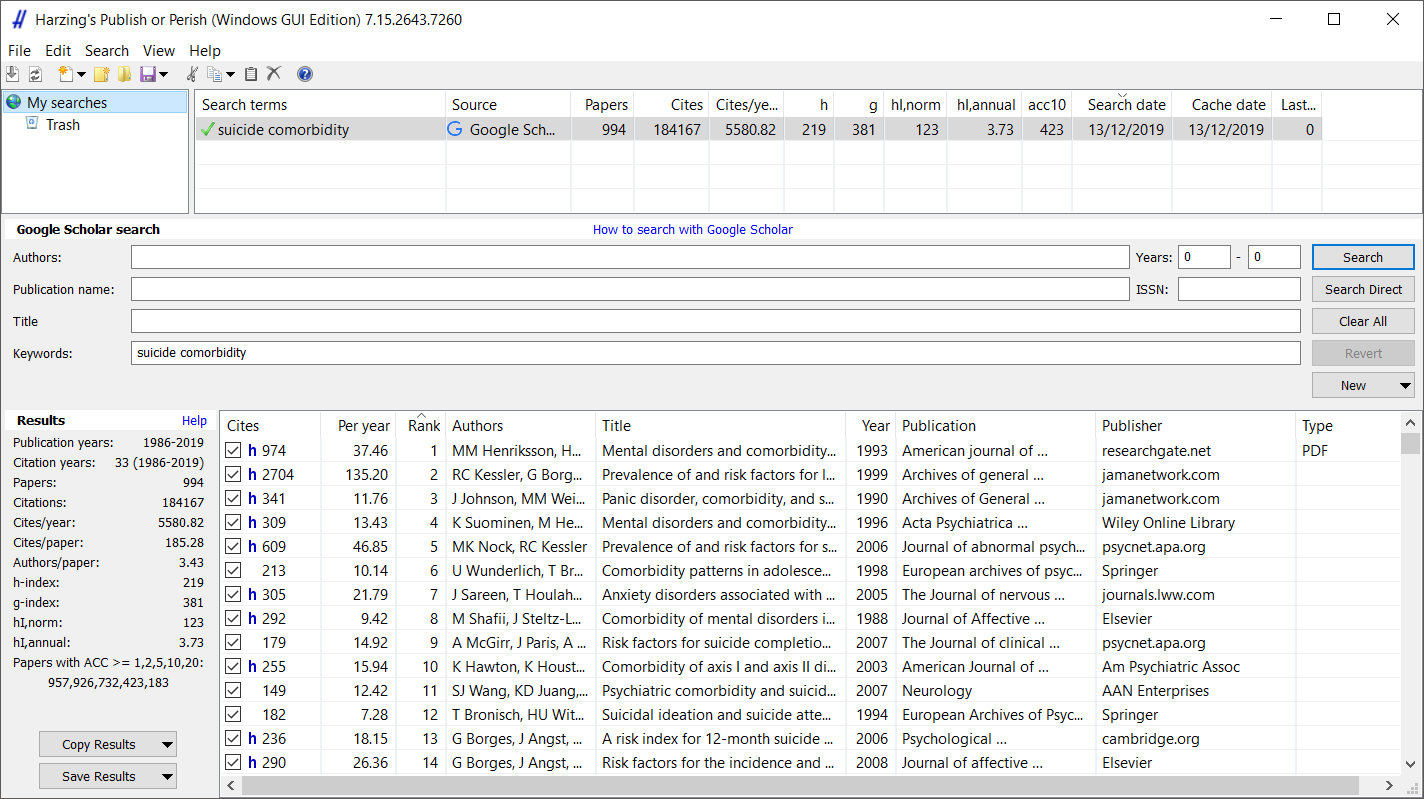
Para ello lo primero que vamos a hacer es encontrar los términos más importantes dentro del conjunto de post de Reddit proporcionado mediante una consulta:

|  |
| --- |
| es = Elasticsearch()  #numero de resultados maximo que dara la consulta  numero\_salidas = 500  results = es.search(  index="reddit-mentalhealth",  body = {  "size": 0,  "query": {  "query\_string": {  "default\_field": "selftext",  "query": query  }  },  "aggs": {  "Title": {  "significant\_terms": {  "field": "title",  "size": numero\_salidas,  "gnd": {}  }  },  "Text": {  "significant\_terms": {  "field": "selftext",  "size": numero\_salidas,  "gnd": {}  }  },  "Subreddit": {  "significant\_terms": {  "field": "subreddit",  "size": numero\_salidas,  "gnd": {}  }  }  }  })  words = []  for j in ["Subreddit", "Text", "Title"]:  for i in results["aggregations"][j]["buckets"]:  if i["key"] not in words and i["key"] not in querywords:  words.append(i["key"])  print("Obtenidas palabras relacionadas con "+query+": "+str(len(words)))  return words |

En esta consulta buscamos los términos dados en el parámetro query ("suicide suicidal \"kill myself\" \"killing myself\" \"end my life\"" en el caso de suicidio y "\"self harm\"" en el caso de conductas autolesivas) y devolvemos los resultados en una lista donde quitamos los términos usados, ya que por definición los factores comórbidos no pueden ser la propia enfermedad.

A continuación necesitamos validar los resultados con un conocimiento experto, para ello utilizaremos la herramienta Publish or Perish ofrecida en <https://harzing.com/resources/publish-or-perish>.

Una vez obtenida la herramienta hacemos las consultas de “suicide comorbidity” y “self harm comorbidity” en Google Scholar obteniendo así dos archivos Json que usaremos más adelante en el script



Obtenemos los títulos de los artículos obtenidos en los Json mediante el script:

|  |
| --- |
| words = []  #Es necesario especificar el encoding para que no de error  with open(name, encoding='utf-8-sig') as json\_file:  data = json.load(json\_file)  for word in data:  words.append(word['title'])  print("Obtenidas palabras del json "+name+": "+str(len(words)))  return words |

Y una vez hecho esto solo hay que comparar las dos listas de palabras para obtener así los términos que hacen referencia a las comorbilidades

|  |
| --- |
| wordsElasticSearch = readElasticsearch(query)  wordsJson = readJson(json)  finalWords = []  for we in wordsElasticSearch:  for wj in wordsJson:  #separamos las palabras de los titulos y comprobamos que no se  #hayan añadido ya  if we in wj.split() and we not in finalWords:  finalWords.append(we)  return finalWords |

Nota:

Para realizar la consulta con elasticsearch correctamente es necesario configurarlo anteriormente, para ello se ha optado en meter el siguiente código en el script, aunque también se podría hacer desde cerebro:

|  |
| --- |
| es = Elasticsearch()  mapping = {  "properties": {  "author": {  "type": "text",  "fielddata": "true"  },  "selftext": {  "type": "text",  "fielddata": "true"  },  "title": {  "type": "text",  "fielddata": "true"  },  "subreddit": {  "type": "text",  "fielddata": "true"  }  }  }  response = es.indices.put\_mapping(  index="reddit-mentalhealth",  body=mapping,  ignore=400 # ignore 400 already exists code  ) |