Integrated Knowledgebase for Rare Diseases using KNIME

Sickle cell Anemia

This Project Explain a KNIME Work-Flow which can extract Disease Target Gene, Gene Sequence, Amino Acid Sequences, Potential Drugs, Smiles structures of those drugs, and Combine different IDs (Drugbank, UCML….) in a single Table for a list of Disease.

Name: Devbrat Anuragi

Roll No.17078

Integrated Knowledgebase for Rare Diseases using KNIME

Sickle cell Anemia

What's the motivation?
There are several open-source databases out there. But since there are not structured, they are not machine consumable. Also since there are many databases, they all have their own unique ID, and this work flow covers how we can combine those different IDs.

Some of the dataset does not have the data in downloadable format like XML or JSON, so this workflow uses Web Scraping to extract and parse required data.

Finally, we want to structure the fetched and parsed data so that we can use this data in NLP and Deep Learning algorithms. This part is left as a future work.

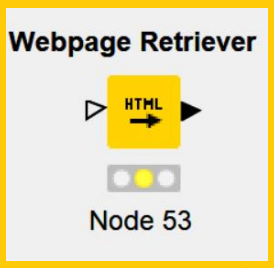
Why only Sickle Cell Anemia?
There is no specific reason why I only chose Sickle cell Anemia. One plausible reason could be The sickle gene is widespread among many tribal population groups in India as discussed in this paper

Is this Work Flow only for Sickle Cell Anemia?
No. With a tweak of single parameter one can easily extract the details of disease or for a list of disease.

Which platform and programming language you have used?
I used Knime and Python3. BeautifulSoup was used for web scraping.

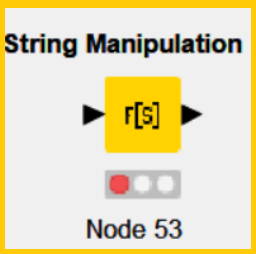
How I started?
I used the master's thesis given in the slack. That workflow is heavily dependent on OpenPhacts, But Openphacts officially has shutdown in 2019. Therefore, apart from the first few nodes, rest of the workflow is complete original.

Most used Knime nodes



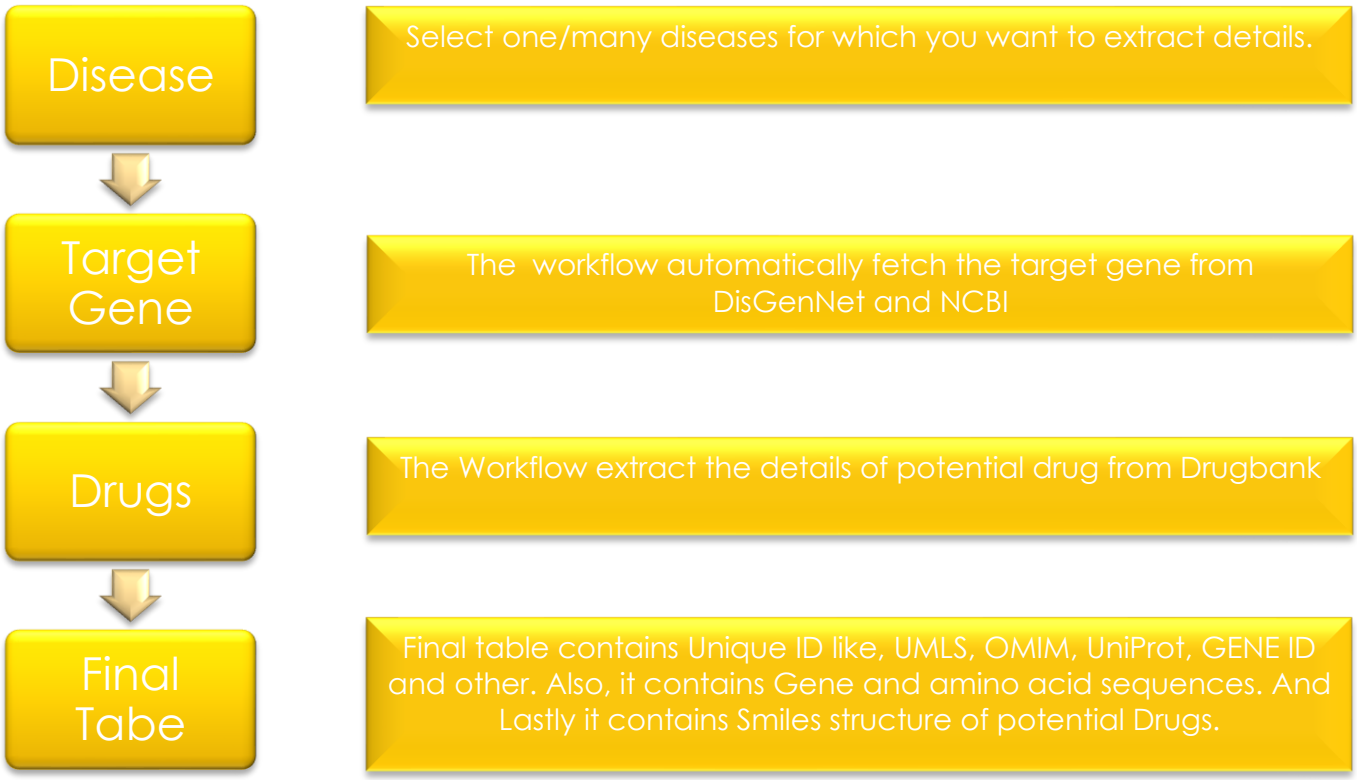
This node is Used to fetch the whole website.

This node is used to write custom python script, mostly for parsing scrapped data.

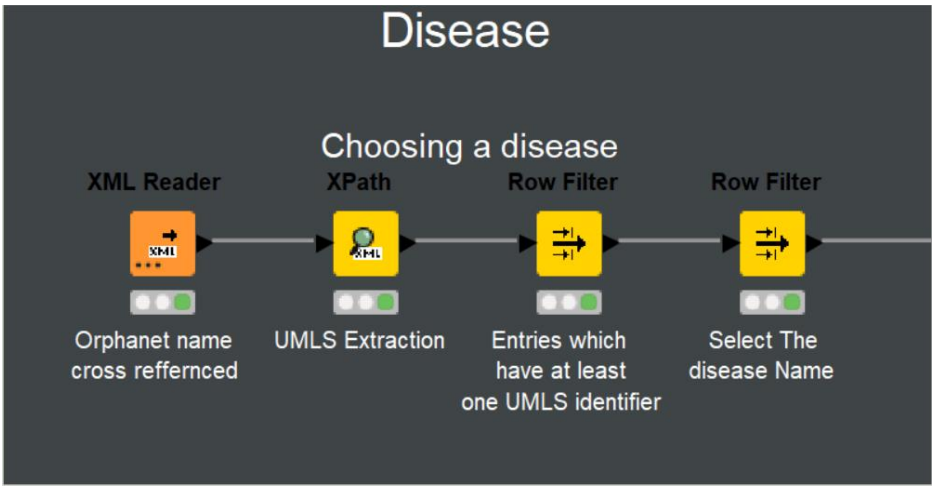


This node is se create URL for each entities

Knime Work Flow Concept.



Extraction of diseases



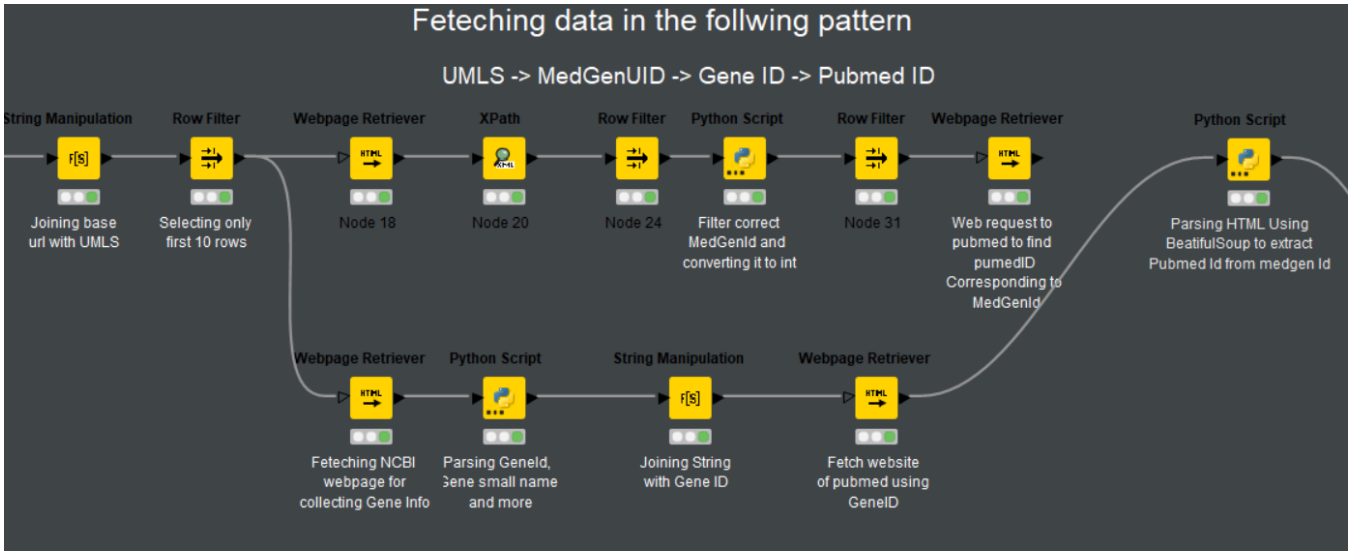
The workflow starts out by reading in the 'Rare diseases and cross-referencing'41 dataset, an XML file which includes cross-references between Orphanet identifiers and identifiers from other databases. The XML Reader provides the option of an XPath filter and for this XML file the XPath filter /JDBOR/DisorderList/Disorder was applied to divide the file into separate disease entries, shown in the KNIME® table as one disease per row As shown Below

Row ID	XML
Row0	<?xml version="1.0" encoding="UTF-8"?> <Disorder id="17601"> <OrphaCode>166024</OrphaCode> <ExpertLink lang="en">http://www.orpha.net/consor/cgi-bin/OC_Exp.php?lng=en&Expert=166024</ExpertLink> <Name lang="en">Multiple epiphyseal dysplasia, Al-Gazali type</Name>
Row1	<?xml version="1.0" encoding="UTF-8"?> <Disorder id="2"> <OrphaCode>58</OrphaCode> <ExpertLink lang="en">http://www.orpha.net/consor/cgi-bin/OC_Exp.php?lng=en&Expert=58</ExpertLink> <Name lang="en">Alexander disease</Name>
Row2	<?xml version="1.0" encoding="UTF-8"?> <Disorder id="17603"> <OrphaCode>166032</OrphaCode> <ExpertLink lang="en">http://www.orpha.net/consor/cgi-bin/OC_Exp.php?lng=en&Expert=166032</ExpertLink> <Name lang="en">Multiple epiphyseal dysplasia, with miniepiphyses</Name>

I have Extracted MLS identifiers sing the following XPath expression /Disorder/ExternalReferenceList/ExternalReference/Source[.="UMLS"]/following-sibling::Reference. Similarliy I have extracted several other identifiers. Following table shows it.

Row ID	Orpha Name	UMLS	OrphaC...	MeSH	OMIM
Row0_1	Multiple epiphyseal dysplasia, Al-Gazali type	C1846722	166024	?	607131
Row1_1	Alexander disease	C0270726	58	D038261	203450
Row2_1	Multiple epiphyseal dysplasia, with miniepi...	C1836307	166032	?	609325
Row3_1	Alpha-mannosidosis	C0024748	61	D008363	248500

Extraction of related PubMed Article ID



What does it do?

So from the previous step we have filter out the row which contains Sickle cell anemia. Then in the stage we use UMLS identifier(Extracted in the previous step) to fetch MedGenId, then using MedGenId we get GeneID and once We get the Gene ID we retrieve all related PubMed Articles.

In the Above image you can see the two banches, both the branches do the same thing the only difference between then is the upper branch used XML to parse the data and the bottom one use BeatifulSoup in Python. I started with the upper branch but I felt that workflow is not much customizable. So I created the second branch. In case You want to use the first one you only need to connect the webpage last retriever and python script node. In the following part I will be only explaining the second branch.

How does it do ?

So first of all UMLS is append to a url like this https://www.ncbi.nlm.nih.gov/medgen/<UMLS> using string Manipulation Node. Then a request id made to that URL and the whole webpage is retrieved in HTML format using Webpage Retriever Node. The I used following python script to scrape the GeneID, MedGenID and Gene short Name.

```
from bs4 import BeautifulSoup
import requests

# Copy input to output
output_table_1 = input_table_1.copy()
print()
output_table_1["MedGenUId"] = ""
output_table_1["Gene Name"] = ""
output_table_1["Gene ID"] = ""
for index, row in output_table_1.iterrows():
      soup = BeautifulSoup(row['Document'], 'lxml',)
      medgen_id = soup.find('dl', class_ = "rprtid").dd.text
      table = soup.find('table', class_ = "medgenTable")

      if table.find('a', class_= "medgenPMinfo") is not None:
            gene_table = table.find('a', class_= "medgenPMinfo")
            print(gene_table)


            output_table_1.at[index,"Gene Name"] = gene_table.text
            print(gene_table.text)
            output_table_1.at[index,"Gene ID"]
=gene_table["href"].split('/')[-1]
            output_table_1.at[index, 'MedGenUId'] = medgen_id
print(output_table_1.head())
```

This gives us the following output.

Row ID	Orpha Name	UMLS	OrphaC...	MeSH	OMIM	API Call URL	MedGenUId	Gene N...	Gene ID
Row162_1	Sickle cell anemia	C0002895	232	D000755	603903	https://www.ncbi.nlm.nih.gov/medgen/C0002895	287	HBB	3043

Now using the GeneID we fetched top 100 pubmed article ID related to that particular gene

```
(https://pubmed.ncbi.nlm.nih.gov/?size=100&linkname=gene_pubmed&from_uid=",$
Gene ID$)
```

Finally this iss the output which contains a list of PubMed Article ids.

Row ID	Orpha Name	UMLS	OrphaC...	MeSH	OMIM	API Call URL	MedGe...	Gene N...	Gene ID	GenIdtoPubmedUrl	PubMed_Id
Row162_1	Sickle cell anemia	C0002895	232	D000755	603903	https://www.ncbi.nlm.nih.gov/medgen/C0002895	287	HBB	3043	https://pubmed.ncbi.nlm.ni...	[34697800,34389729,34271589,...]

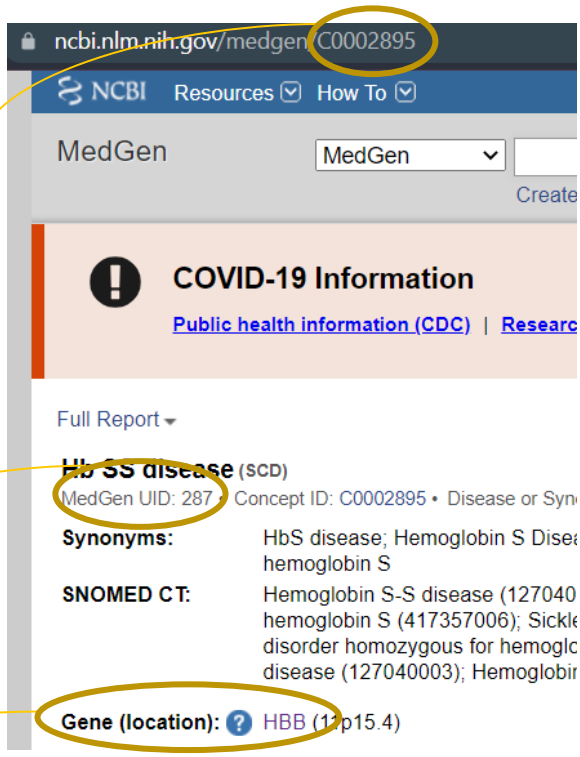
For this project I have only fetched the Pubmed ids, in te future work I will use this for test mining purose.

Want to see what I actually scraped form wepage?

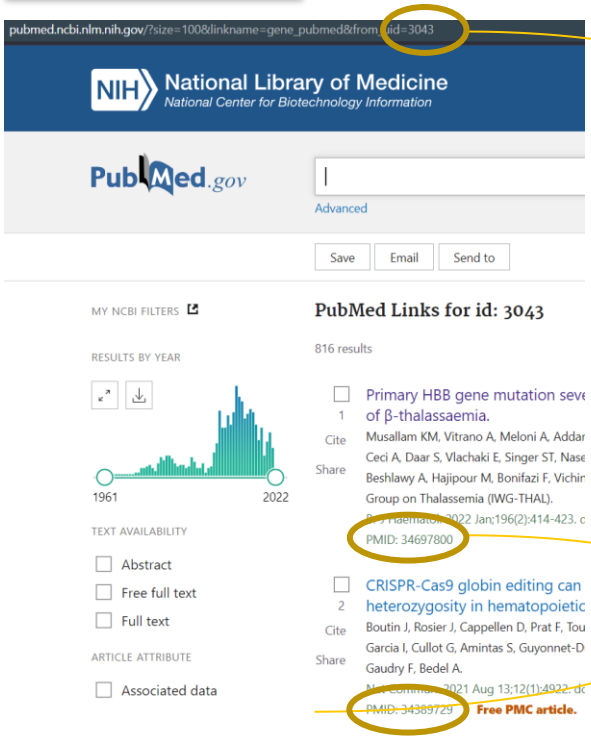
This is the UMLS Identifier we got from the Orphanet File

Scrapped this MedGenID

The GeneID is actually in the source code. Its not visible in this webpage

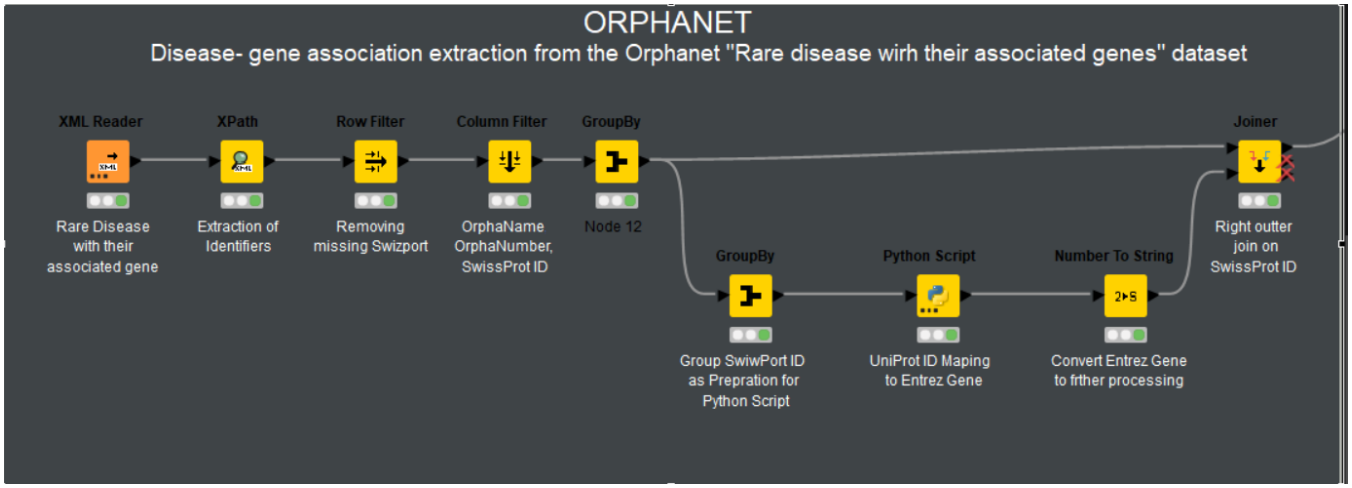


This is where I used the Extracted GeneId



This Pubmed ID were extracted

Fetching some more unique identifiers



What does it do?

First of all this workflow is reproduced from the previously mentioned thesis. This workflow convert SwissProt ID to NCBI and UniprotID for all the disease in the "Rare disease with their associate gene dataset."

How does it do?

This part of the retrieval of disease-gene associations, which is based on Orphanet data, starts out by reading in the 'Rare diseases with their associated genes' dataset with an XML Reader. The XPath filter /JDBOR/DisorderList/Disorder is used to divide the XML into single disease entries per row. In preparation for the mapping the dataset is filtered to include only entries that have SwissProt identifiers and the table is reduced to relevant columns only with t he Column Filter. To ensure that each association between a disease and a gene is unique a GroupBy node is additionally employed. The mapping service is executed with a Python script49 provided by UniProt

which I adapted for the use in KNIME® (shown in Listing 5) and which requires the input to be specifically formatted. The required format is generated with a GroupBy node which aggregates unique UniProt accession numbers by concatenating them with a comma in between, resulting in a single cell with all identifiers that need to be mapped (e.g. 'Q13315,

P05067, P01034, P18564, Q13751, Q9BZG2, Q9NP70, ...'). This GroupBy node feeds into a Python Script (1→1) node, which is configured with the script shown below.

```
import urllib.parse
import urllib.request
import io
import pandas as pd
url = 'https://www.uniprot.org/uploadlists/'

params = {
'from': 'ID',
'to': 'P_ENTREZGENEID',
'format': 'tab',
'query': input_table_1['Concatenate(SwissProt)'].iloc[0]
}

data = urllib.parse.urlencode(params)
data = data.encode('utf-8')
req = urllib.request.Request(url, data)
with urllib.request.urlopen(req) as f:
   response = f.read()

output_table_1= pd.read_csv(io.StringIO(response.decode('utf-8')), sep='\t')
output_table_1.rename(columns = {'From': 'UniProt ID',
                                 'To': "NCBI"}, inplace = True)

#print(response.decode('utf-8'))
print(output_table_1.head())
```

Now that both the top and the bottom section of the disease-gene association

extraction are equipped with NCBI identifiers both sources can be joined into one table which will be used for the retrieval of drugs and publication abstracts.

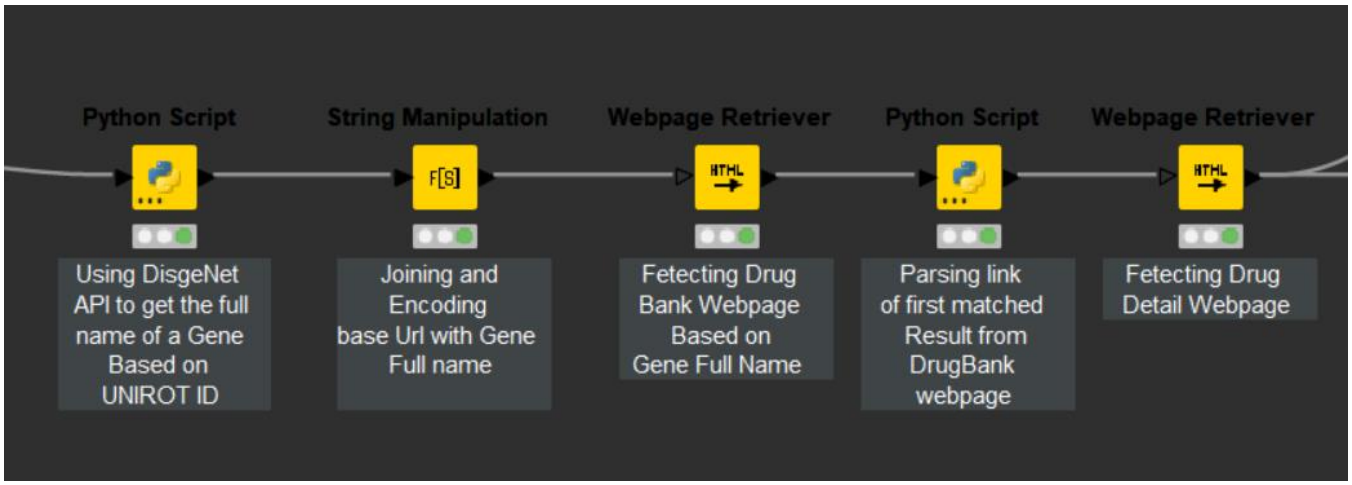
Following is the final results of this workflow

Row ID	OMIM	OrphaCode	Disorder Name	UniProt ID	NCBI
Row9_Row8	?	?	?	P33261	1557
Row99_Row67	104311	100069	Semantic dementia	P49768	5663
Row999_Row...	152790	96265	Leydig cell hypoplasia due to complete LH resistance	P22888	3973
"Row99_Row67" (2/6513)		3000	Familial male-limited precocious puberty	P22888	3973
Row997_Row...	152790	?	?	P22888	3973
Row996_Row...	152780	325448	Leydig cell hypoplasia due to LHB deficiency	P01229	3972

Then we join the table from the both workflow based on OrphaName. And now we have completed our first step of extracting several identifier related to Sickle cell Anemia

Row ID	Orpha Na...	UMLS	OrphaC...	MeSH	OMIM	API C...	MedGe...	Gene N...	Gene ID	GenIdt...	PubMe...	OMIM (...	OrphaC...	UniProt ID	NCBI
Row162_1_R...	Sickle cell anemia	C0002895	232	D000755	603903	https://ww...	287	HBB	3043	https://pub...	[34697800,...	141900	232	P68871	3043

Extraction of Gene



What does it do?

This uses as DisGenNet API to get the Full name of gene using the UNIPROT ID. Then that full name is appended to Drug bank query. Once that query is successful we extract the link to the first result. Then we retrieve that webpage using webpage Retriever node.

Now you see there are two branch emerging after the webpage retriever node. The upper branch do the gene extraction and we will discuss it here, the lower branch deals with drug extraction, discussed in the next section.

How Does it do?

Starting with the python script node, it Uses DisGenNet API. we pass a Uniprot ID in the api to get the gene full name. First of all you need to create an account on DisGenNet then you get a Authorization API. without that authorization token oone ccan't make the API call.

Following is the python script:

```
# Copy input to output
output_table_1 = input_table_1.copy()
'''
Script example to use the DisGeNET REST API with the new authentication
system
'''

#For this example we are going to use the python default http library
import requests
import json

#Build a dict with the following format, change the value of the two keys
your DisGeNET account credentials, if you don't have an account you can
create one here https://www.disgenet.org/signup/
auth_params = {"email":"add@your.email","password":"Tellmeyourpassword"}

api_host = "https://www.disgenet.org/api"

api_key = None
s = requests.Session()
try:
    r = s.post(api_host+'/auth/', data=auth_params)
    if(r.status_code == 200):
        #Lets store the api key in a new variable and use it again in new
requests
        json_response = r.json()
        api_key = json_response.get("token")
        print(api_key + "This is your user API key.") #Comment this line if
you don't want your API key to show up in the terminal
    else:
        print(r.status_code)
        print(r.text)
except requests.exceptions.RequestException as req_ex:
    print(req_ex)
    print("Something went wrong with the request.")

if api_key:
    #Add the api key to the requests headers of the requests Session object
in order to use the restricted endpoints.
    s.headers.update({"Authorization": "Bearer %s" % api_key})
    output_table_1['gene_fullname'] = ""
    for index, row in output_table_1.iterrows():
      gda_response =
s.get(api_host+'/gene/'+output_table_1.at[index,'NCBI'],
params={'source':'CTD_human'})
      json_response = json.loads(gda_response.text)
      output_table_1.at[index,'gene_fullname'] =
json_response[0]['gene_fullname']

if s:
    s.close()
```

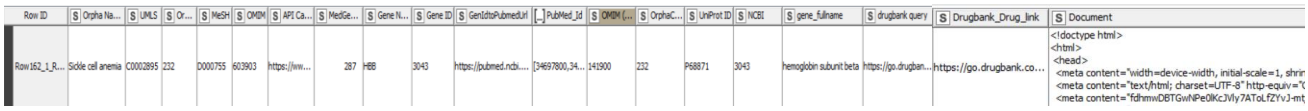
Once we have the Gene Full name we append it to Drugbank Query to get the potential drug details like this

```
https://go.drugbank.com/unearth/q?searcher=bio_entities&query=",replace($gen
e_fullname$," ","%20"
```

Then we make a request to thi url and fetched the whole HTML code using webpage retiriver node. Then to make thing simple I only parse the URL of first result that matches our query. Then again we go to newly fetched URL. The python script to parse the link is below.

```
from bs4 import BeautifulSoup
import requests

# Copy input to output
output_table_1 = input_table_1.copy()
output_table_1['Drugbank_Drug_link'] = ""
for index ,row in output_table_1.iterrows():
      soup = BeautifulSoup(row['Document'],'lxml')
      first_item = soup.find('h2', class_= 'hit-link')
      print(first_item)
      if first_item.text.lower() == row['gene_fullname'].lower():
            output_table_1.at[index,'Drugbank_Drug_link'] =
first_item.a['href']

output_table_1.drop("Document", axis= 1, inplace=True)
```

Finally table look like this after these steps.



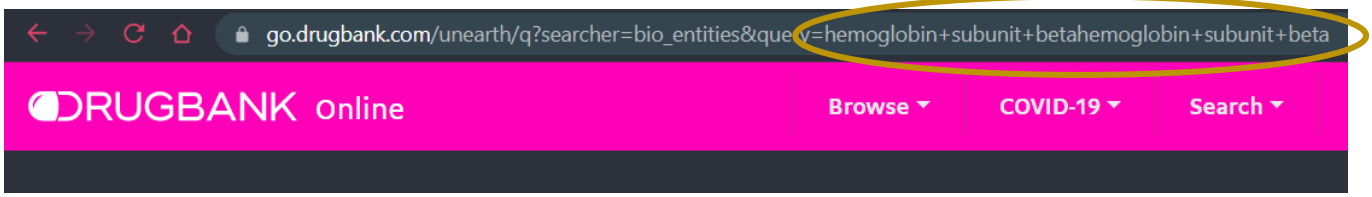
Want to see what I actually scraped form webpage?

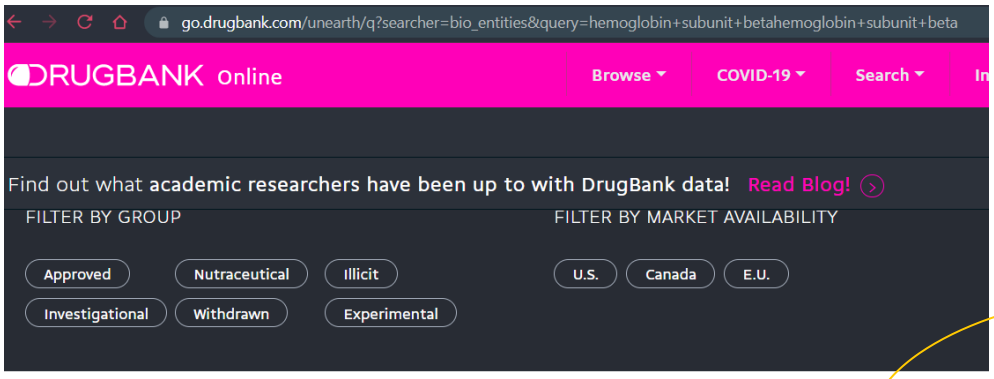
Response body

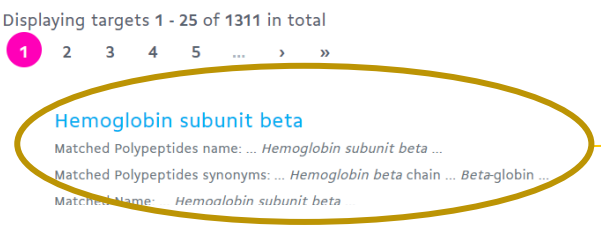

```
[
  {
    "geneid": "3043",
    "uniprotid": "P68871",
    "symbol": "HBB",
    "gene_fullname": "hemoglobin subunit beta",
    "source": "ALL",
    "protein_class": null,
    "protein_class_name": null,
    "num_diseases": 272,
    "num_variants": 188,
    "dsi": 0.494,
    "dpi": 0.808,
    "pli": 1.2274e-9
  }
]
```


This is the response of DisGenNet API. I take full name from here.

Use that full name to create a DrugBank query like this.

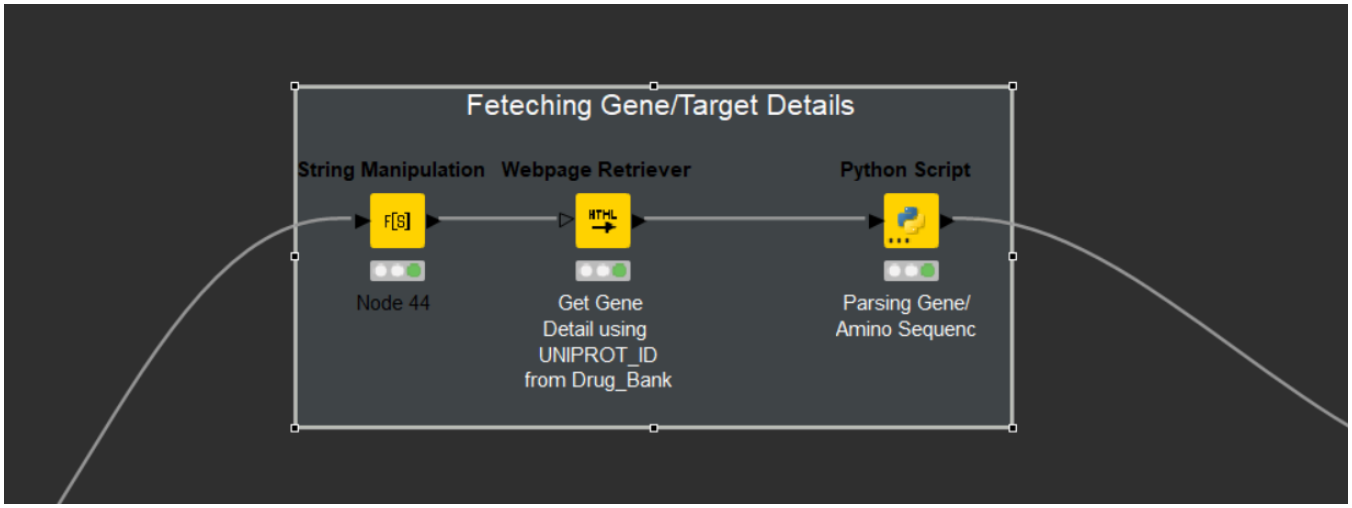






Finally when search is complete I extract the link to the first result.

The upper branch/ Gene sequence fetching Branch



What does it do?

In the previous section I mentioned about two branches appearing from the webpage retriever node, so in this section I disscusss the upper branch. It go the gene detail page of Drug Bank using UniProt id and then extract amino acid and gene sequences.

How does it do?

First we append Uniprot it like this

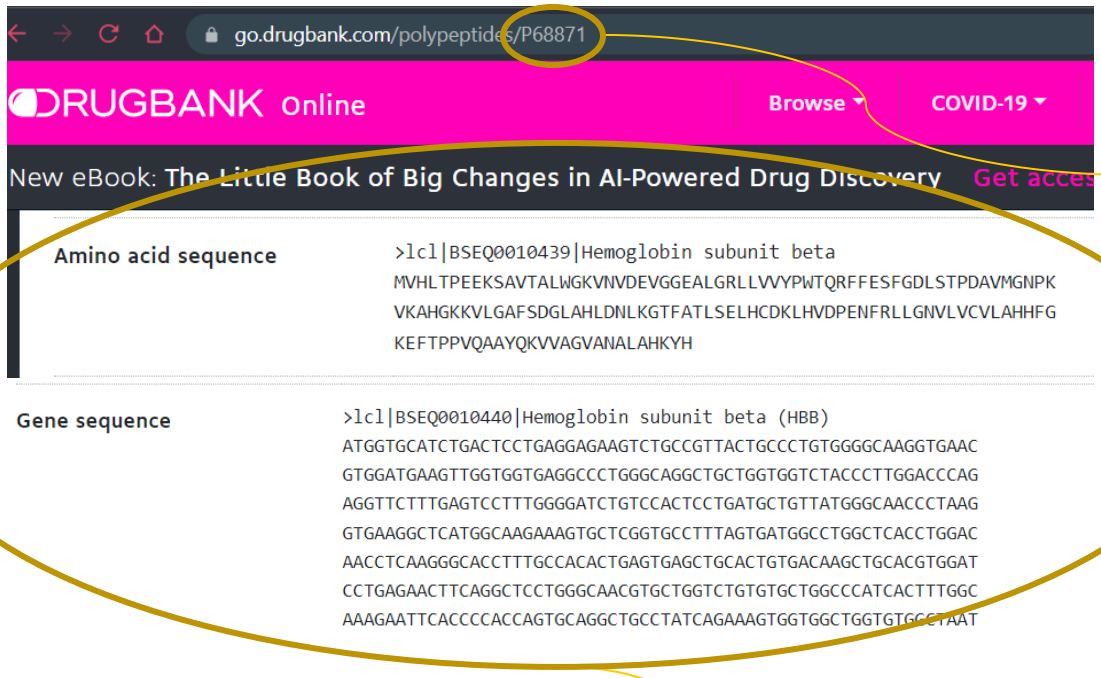
```
"https://go.drugbank.com/polypeptides/",$UniProt ID$
```

Then retrive the whole page and parse the required results. Following is the python script:

```
from bs4 import BeautifulSoup
import pandas as pd

# Copy input to output
output_table_1 = input_table_1.copy()
```

```
output_table_1["amino_acid_sequence"] = ""
output_table_1["gene_sequence"] = ""
for index, row in output_table_1.iterrows():
      soup = BeautifulSoup(row['Document (#1)'],'lxml')
      output_table_1.at[index, "amino_acid_sequence"] = soup.find('div',
class_ = 'card-content' ).dl.find("dt", id= "amino-acid-
sequence").findNext("dd").pre.text
      output_table_1.at[index, "gene_sequence"] = soup.find('div', class_ =
'card-content' ).dl.find("dt", id= "gene-sequence").findNext("dd").pre.text
      #output_table_1.at[index, "chromosome_location"] = soup.find('div',
class_ = 'card-content' ).dl.find("dt", id= "chromosome-
location").findNext("dd").pre.text
      #print(attri.split(" "))
output_table_1.drop('Document (#1)',axis = 1, inplace = True)
output_table_1.drop('Document',axis = 1, inplace = True)
```

Want to see what I actually scraped form webpage?

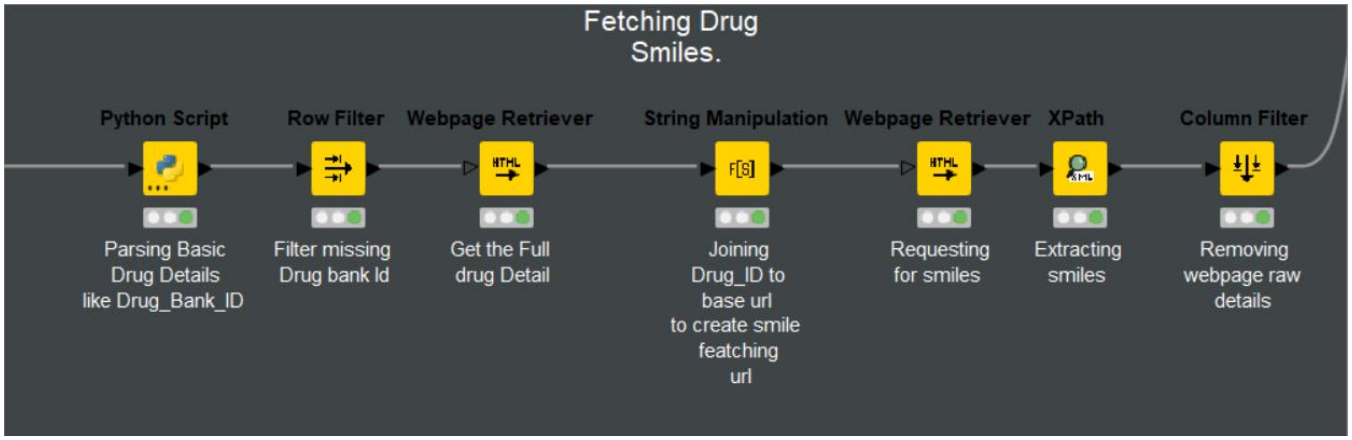


This is where we use UNIPROT ID

This is the gene and amino acid sequence. This complete our gene extraction.

Extraction of Drug

The lower branch/ Smiles fetching Branch



What does it do?

SO if you are following closely you may be asking earlier we have fetched the link to the first result of drug bank query, what its use. Here we will be using it. We parse the Durg ID, Name, Drug Group, Prarmocolgical Action, and detail's link from that table. Then I go to the Details page and then extract Smiles.

How does it do?

Script to extract drug related details is given below:

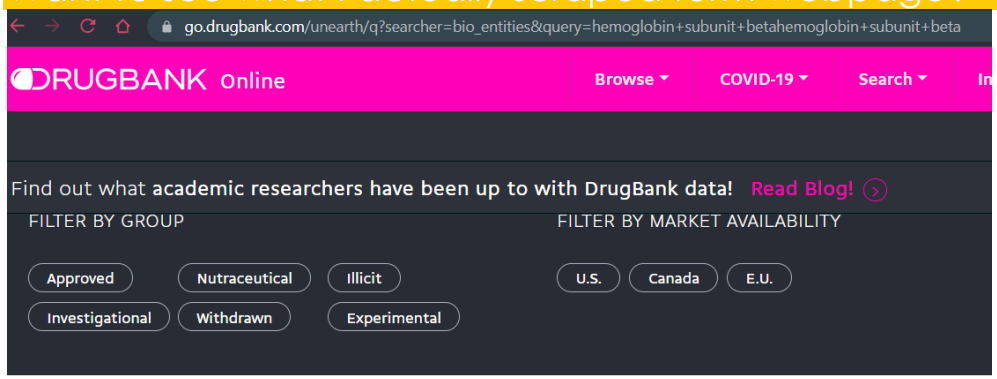
```
from bs4 import BeautifulSoup
import pandas as pd
output_table_1 = pd.DataFrame()
output_table_1['Drug_Id'] = ""
output_table_1['Drug_Name'] = ""
output_table_1['Drug_Group'] = ""
output_table_1['Pharmacological action'] = ""
output_table_1['Actions'] = ""
```

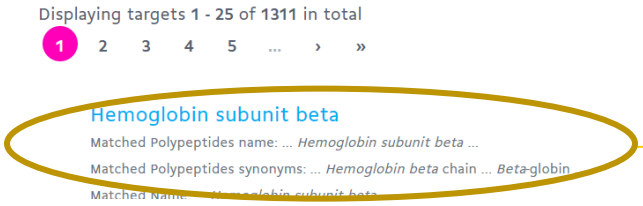
```
output_table_1['Drug_Detail'] = ""


for index, row in input_table_1.iterrows():
      soup = BeautifulSoup(row['Document'],'lxml')
      table = soup.find("table", id = "target-relations"
).find("tbody").find_all('tr')
      #print(len(table))
      for i,item in enumerate(table):
            output_table_1 = output_table_1.append(row, ignore_index = True)
            cells = item.find_all("td")
            group = cells[2].get_text()
            """
            if "approved" not in group:
                  continue
            """
            output_table_1.at[i,'Drug_Id'] = cells[0].get_text()
            output_table_1.at[i,'Drug_Name'] = cells[1].get_text()


            output_table_1.at[i,'Drug_Group'] = group
            output_table_1.at[i,'Pharmacological action'] =
cells[3].get_text()
            output_table_1.at[i,'Actions'] = cells[4].get_text()
            output_table_1.at[i,'Drug_Detail'] = cells[5].a['href']
            print("="*10)
output_table_1.drop('Document', axis = 1, inplace= True)
```

Then we join drug id to a url to get the Smiles like this:

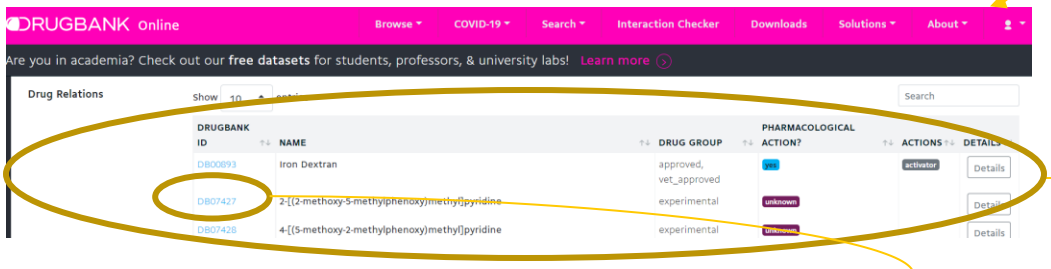
```
https://go.drugbank.com/structures/small_molecule_drugs/",$Drug_Id$,".smiles
```

Want to see what I actually scraped form webpage?

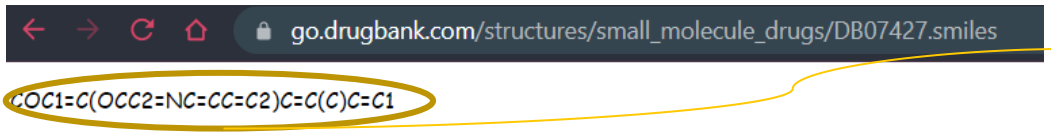




Go to this link



Extract These details



Smiles

Final table ready for Deep Learning

Row ID	amino_acid_sequence	gene_sequence	Drug_Id	Drug_Group	Gene ID	Smiles
Row162_1_R...	>lcl|BSEQ0010439|Hemoglobin subunit beta MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAV... VKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAH... KEFTPPVQAAYQKVVAGVANALAHKYH	>lcl|BSEQ0010440|Hemoglobin subunit beta (HBB) ATGGTGCATCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGGG... GTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGGCTGCTGGTGGTCTACC... AGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATGCTGTTATGGG... GTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGG... AACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGC...	DB00893	approved, vet_approved	3043	?
Row162_1_R...	>lcl|BSEQ0010439|Hemoglobin subunit beta MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAV... VKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAH... KEFTPPVQAAYQKVVAGVANALAHKYH	>lcl|BSEQ0010440|Hemoglobin subunit beta (HBB) ATGGTGCATCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGGG... GTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGGCTGCTGGTGGTCTACC... AGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATGCTGTTATGGG... GTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGG... AACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGC...	DB07427	experimental	3043	COC1=C(OCC2=NC=CC...
Row162_1_R...	>lcl|BSEQ0010439|Hemoglobin subunit beta MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAV... VKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAH... KEFTPPVQAAYQKVVAGVANALAHKYH	>lcl|BSEQ0010440|Hemoglobin subunit beta (HBB) ATGGTGCATCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGGG... GTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGGCTGCTGGTGGTCTACC... AGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATGCTGTTATGGG... GTGAAGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGG... AACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCACTGTGACAAGC...	DB07428	experimental	3043	COC1=CC(OCC2=CC=N...

How this is ready for Deep Learning?

Yeah, it not completely ready yet because its very small since it is only for one disease. But in this project I am just giving a flavor of my approach. Of course if we do this for large number of disease we can get a big data set.

But whats the independent and dependent variable here?

So I propose we make a classification model that can predict the score how likely is a Smile of a drug is suitable for a particular gene and amino acid sequence. So our feature column will be : amino_acid_sequence, gene_sequence and Smiles. Our target column will be Drug_Group.