FDA Drug Repurposing Data Preparation

Table of Contents

[Intro 2](#_Toc71816275)

[Detail of Data Sources 2](#_Toc71816276)

[Uniprot Protein Synonyms 2](#_Toc71816277)

[Description 2](#_Toc71816278)

[Source 2](#_Toc71816279)

[Process 2](#_Toc71816280)

[Code 5](#_Toc71816281)

[Virus Host Proteins 6](#_Toc71816282)

[Description 6](#_Toc71816283)

[Source 6](#_Toc71816284)

[Process 6](#_Toc71816285)

[Drug Bank 7](#_Toc71816286)

[Description 7](#_Toc71816287)

[Source 8](#_Toc71816288)

[Process: Small Molecule Drug 8](#_Toc71816289)

[Code 8](#_Toc71816290)

[Process: Drug-Protein 11](#_Toc71816291)

[Code 11](#_Toc71816292)

[Process: Drugbank-Synonyms 12](#_Toc71816293)

[Code 13](#_Toc71816294)

[FDA COVID Clinical Trials 14](#_Toc71816295)

[Description 14](#_Toc71816296)

[Source 14](#_Toc71816297)

[Process 14](#_Toc71816298)

[Code 16](#_Toc71816299)

[STRING Protein Protein Interactions 20](#_Toc71816300)

[Description 20](#_Toc71816301)

[Source 20](#_Toc71816302)

[Process 20](#_Toc71816303)

[Code 21](#_Toc71816304)

[Joining Tables 23](#_Toc71816305)

# Intro

The database for drug repurposing AI modeling consists of 7 tables from 5 sources:

* One to cross index protein identifiers
* One to describe the virus proteins
* Three to characterize drugs under investigation including proteins and synonyms
* One to link proteins by association
* One for testing the resulting groups of drugs prioritized by the AI methods.

Each of these tables, their sources, processing and challenges are described in the next section. It is worth noting that additional information can be extracted from these sources (especially DrugBank).

We also discuss possible methods of joining these tables to support supervised and unsupervised model development.

# Detail of Data Sources

## Uniprot Protein Synonyms

### Description

The Uniprot protein ID Mapping database is maintained by Uniprot.org. It contains dozens of IDs available for each protein. We use this database to standardize the proteins from each source on the UniProt-ID. The IDs we are concerned with are Ensemble-Pro (for the String database) and Gene-ID (for the Virus Host Protein Table). The Drug-Protein table from DrugBank is already standardized on UniProt-ID.

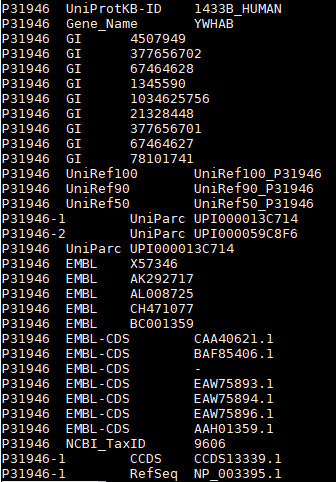
### Source

<https://ftp.uniprot.org/pub/databases/uniprot/knowledgebase/docs/sec%5Fac.txt>

HUMAN\_9606\_idmapping.dat

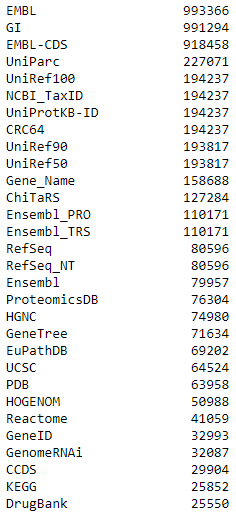
### Process

The idmapping file (HUMAN\_9606\_idmapping.dat) contains 3 columns. Column 1 is the UniProt-ID, Column 2 is the alternate-ID-name and Column 3 is the alternate-ID. Below is an excerpt of the first few rows from the idmapping file:



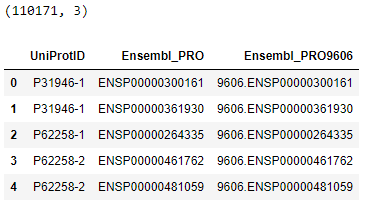
This file currently contains over 6 million rows which posed memory limit issues.

Since a large number of the rows are not needed, we were able to reduce the memory needed by reading in only column 2 for all rows, then selecting the rows that had the alternate-id-names we were interested in. In the table below, we can see that Ensemble-PRO has only 110,171 records and GeneID has only 32,993.

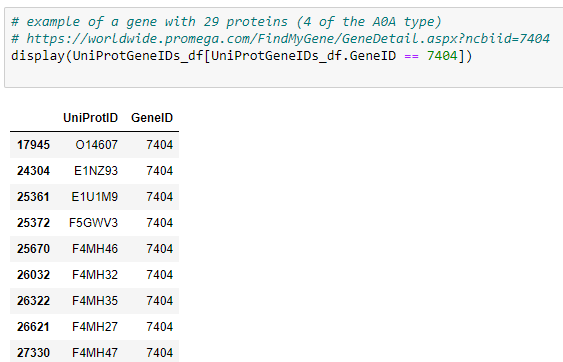


There are many more alternate-ID-names available, the table above displays only the most common.

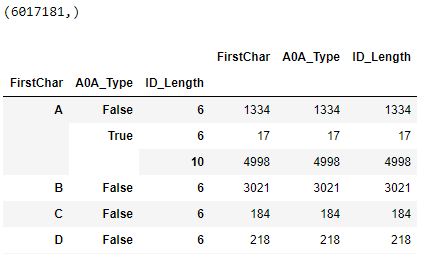
For the Ensembl\_PRO ID to match exactly with the protein IDs in STRING we pre-pended the organism code for humans, 9606.



Many Gene-IDs map to multiple proteins as show below for GeneID 7404.



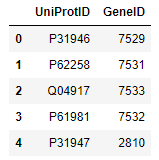
To resolve the duplicates, a selection preference order was established where proteins starting with “P” were preferred first, then “Q”, “O” then other single letter prefixes and last “A0A”.







This matching order resolved nearly all duplicates. A sample of the GeneID to UniProtID mappings is shown below:



### Code

## Virus Host Proteins

### Description

The Virus Host Protein table is the list of Host (human) proteins that either the new virus or similar known viruses effect. The table below shows an example of the UniProtIDs associated with various Coronavirus Host Proteins (along with the Gene\_ID and PubMed\_ID reference for establishing the connection).

Alternate sources of Virus Host Proteins can be swapped out in the process to investigate new viruses or as the proteins a new virus targets are discovered they can be added to the list of proxy viruses or replace it.

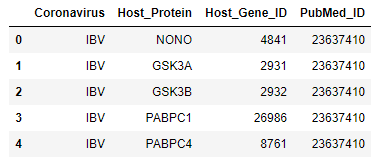
### Source

<https://www.ncbi.nlm.nih.gov/sars-cov-2/>

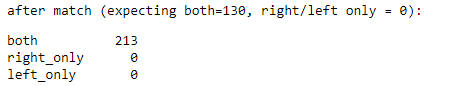
### Process

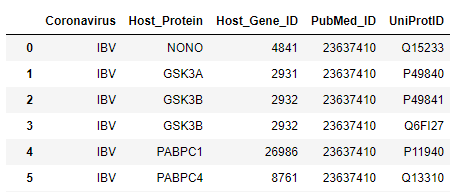
Our initial virus-host-protein table contained Host\_Protein, and Host\_Gene\_ID for a number of Coronaviruses. The PubMed\_ID for the reference publication describing the virus-protein relationship is also provided in the table, but not used.





To support matching into DrugBank, the Host\_Gene\_ID is linked to the UniProtID using the UniProt table described in the first section. Note that some Host\_Gene\_IDs mapped to multiple proteins even after the order of preference processing described for UniProt. Every Host\_Gene\_ID mapped to 1 or more UniProtIDs and we expanded our list from 130 genes to 213 proteins.





The modeling process starts with these as the proteins of interest.

## Drug Bank

### Description

DrugBank is an XML database rather than a table. For this study, we are extracting a subset of the information available in the database:

* A table of Small Molecule Drugs with a subset of available attributes
* Known Drug-Protein associations
* A list of synonyms for each drug

Additional information is available such as:

* PK and PD information on each drug
* Drug-Drug interactions
* Drug-Food interactions
* SMILES chemical structure (which can be converted into a data matrix)
* Indication

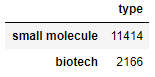
Some of the additional information is in text format and would require text analytical approaches to extract useful data for models.

### Source

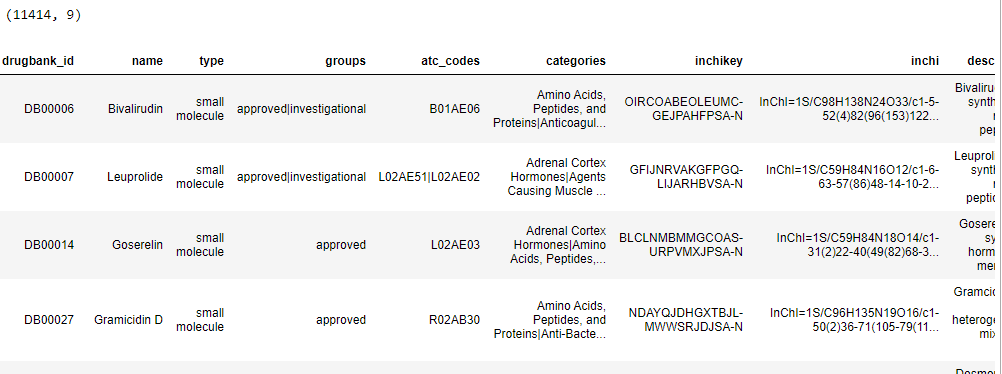
<https://go.drugbank.com/releases/latest>

### Process: Small Molecule Drug

As of the version used in this study, DrugBank contained 13,580 drug entries. Of these, the 11,414 small molecule drugs were selected.



For each of the 11,414 small molecule drugs, 9 of the available columns were extracted:



Three of the 9 columns are series columns that possibly contain multiple values. These columns are “groups”, “atc\_codes”, and “categories”. To be used in modeling, additional post processing such as creating separate tables for each or transposing the data from the series columns into “one hot” encoded variables is needed.

In addition, there is one text column “description”. Currently this is being used only as a descriptive column but may be processed with text analytical techniques to extract information if needed.

### Code

download = '/opt/sas/viya/config/data/cas/default/public/FDA/DrugBank/drugbank\_all\_full\_database'

savepath = '/home/sasdemo/FDAData/1\_DrugBank/DrugBank\_CSVs'

# moved to CAS and got locked out :-/

# /opt/sas/viya/config/data/cas/default/public/FDA/DrugBank/drugbank\_all\_full\_database/drugbank.xml

#xml\_path = os.path.join(download, 'full database.xml')

xml\_path = os.path.join(download, 'drugbank.xml') # just renamed above to match the xsd file

print(xml\_path)

with open(xml\_path,encoding="utf8") as xml\_file:

tree = ET.parse(xml\_file) # read the XML File (tree)

root = tree.getroot() # get root of the XML Tree

ns = '{http://www.drugbank.ca}'

inchikey\_template = "{ns}calculated-properties/{ns}property[{ns}kind='InChIKey']/{ns}value"

inchi\_template = "{ns}calculated-properties/{ns}property[{ns}kind='InChI']/{ns}value"

aliases = {}

newrows = list()

for i, drugnew in enumerate(root):

row = collections.OrderedDict()

assert drugnew.tag == ns + 'drug'

row['type'] = drugnew.get('type')

row['drugbank\_id'] = drugnew.findtext(ns + "drugbank-id[@primary='true']")

row['name'] = drugnew.findtext(ns + "name")

row['description'] = drugnew.findtext(ns + "description")

row['indication'] = drugnew.findtext(ns+'indication')

row['groups'] = [group.text for group in

drugnew.findall("{ns}groups/{ns}group".format(ns = ns))]

row['atc\_codes'] = [code.get('code') for code in

drugnew.findall("{ns}atc-codes/{ns}atc-code".format(ns = ns))]

row['categories'] = [x.findtext(ns + 'category') for x in

drugnew.findall("{ns}categories/{ns}category".format(ns = ns))]

row['inchi'] = drugnew.findtext(inchi\_template.format(ns = ns))

row['inchikey'] = drugnew.findtext(inchikey\_template.format(ns = ns))

row['SMILES'] = drugnew.findall("{ns}calculated-properties/{ns}property[@kind='SMILES']".format(ns = ns))

# Add drug aliases

aliases = {

elem.text for elem in

# added /{ns}name to correct missing names here

drugnew.findall("{ns}international-brands/{ns}international-brand/{ns}name".format(ns = ns)) +

# why use only english ? remove this limit

#drugnew.findall("{ns}synonyms/{ns}synonym[@language='English']".format(ns = ns)) +

drugnew.findall("{ns}synonyms/{ns}synonym".format(ns = ns)) +

# why have the international-brands in 2x ? remove this

#drugnew.findall("{ns}international-brands/{ns}international-brand".format(ns = ns)) +

drugnew.findall("{ns}products/{ns}product/{ns}name".format(ns = ns))

}

aliases.add(row['name'])

row['aliases'] = sorted(aliases)

newrows.append(row)

drugbanknew\_df = pandas.DataFrame.from\_dict(newrows)

pandas.options.display.max\_colwidth = 500

print("drugbanknew\_df class is: ", drugbanknew\_df.\_\_class\_\_)

print("drugbanknew\_df.groups class is: ", drugbanknew\_df.groups.\_\_class\_\_)

print("drugbanknew\_df.atc\_codes class is: ", drugbanknew\_df.atc\_codes.\_\_class\_\_)

print("drugbanknew\_df.categories class is: ", drugbanknew\_df.categories.\_\_class\_\_)

print("drugbanknew\_df.aliases class is: ", drugbanknew\_df.aliases.\_\_class\_\_)

print(drugbanknew\_df.shape)

#display(drugbanknew\_df.head(8))

display(drugbanknew\_df.iloc[:2])

display(drugbanknew\_df.iloc[5])

display(pandas.DataFrame(drugbanknew\_df['type'].value\_counts()))

drugbanknew\_df = drugbanknew\_df[drugbanknew\_df.type == 'small molecule']

display(pandas.DataFrame(drugbanknew\_df['type'].value\_counts()))

columns = ['drugbank\_id', 'name', 'type', 'groups', 'atc\_codes', 'categories', 'inchikey', 'inchi', 'description']

drugbank\_df = pandas.DataFrame.from\_dict(rows)[columns]

display(pandas.DataFrame(drugbank\_df['type'].value\_counts()))

display(drugbank\_df.head(10))

drugbank\_slim\_df = drugbank\_df[

#drugbank\_df.groups.map(lambda x: 'approved' in x) &

#drugbank\_df.inchi.map(lambda x: x is not None) &

drugbank\_df.type.map(lambda x: x == 'small molecule')

]

display(drugbank\_slim\_df.shape)

display(drugbank\_slim\_df.head())

# write slim drugbank tsv

path = os.path.join(savepath, 'drugbank-slim.tsv')

drugbank\_slim\_df.to\_csv(path, sep='\t', index=False)

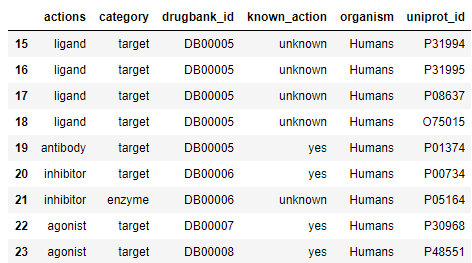
print(path)

print(drugbank\_slim\_df.shape)



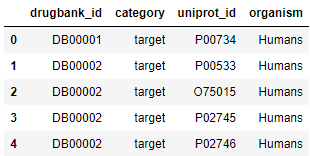
### Process: Drug-Protein

The drug-protein table is derived from multiple sections of the DrugBank XML database and includes target and enzyme proteins regardless of action.



Only the DrugBank ID, category, UniProt\_D and organism (always Humans in this case) are kept.





### Code

protein\_rows = list()

for i, drug in enumerate(root):

drugbank\_id = drug.findtext(ns + "drugbank-id[@primary='true']")

for category in ['target', 'enzyme', 'carrier', 'transporter']:

proteins = drug.findall('{ns}{cat}s/{ns}{cat}'.format(ns=ns, cat=category))

for protein in proteins:

row = {'drugbank\_id': drugbank\_id, 'category': category}

row['organism'] = protein.findtext('{}organism'.format(ns))

row['known\_action'] = protein.findtext('{}known-action'.format(ns))

actions = protein.findall('{ns}actions/{ns}action'.format(ns=ns))

row['actions'] = '|'.join(action.text for action in actions)

uniprot\_ids = [polypep.text for polypep in protein.findall(

"{ns}polypeptide/{ns}external-identifiers/{ns}external-identifier[{ns}resource='UniProtKB']/{ns}identifier".format(ns=ns))]

if len(uniprot\_ids) != 1:

continue

row['uniprot\_id'] = uniprot\_ids[0]

#ref\_text = protein.findtext("{ns}references[@format='textile']".format(ns=ns))

#pmids = re.findall(r'pubmed/([0-9]+)', ref\_text)

#row['pubmed\_ids'] = '|'.join(pmids)

protein\_rows.append(row)

protein\_df = pandas.DataFrame.from\_dict(protein\_rows)

columns = ['drugbank\_id', 'category', 'uniprot\_id', 'organism']

entrez\_df = protein\_df[columns]

path = os.path.join(savepath, 'proteins.tsv')

#entrez\_df.to\_csv(path, sep=',', index=False)

entrez\_df.to\_csv(path, sep='\t', index=False)

print(path)

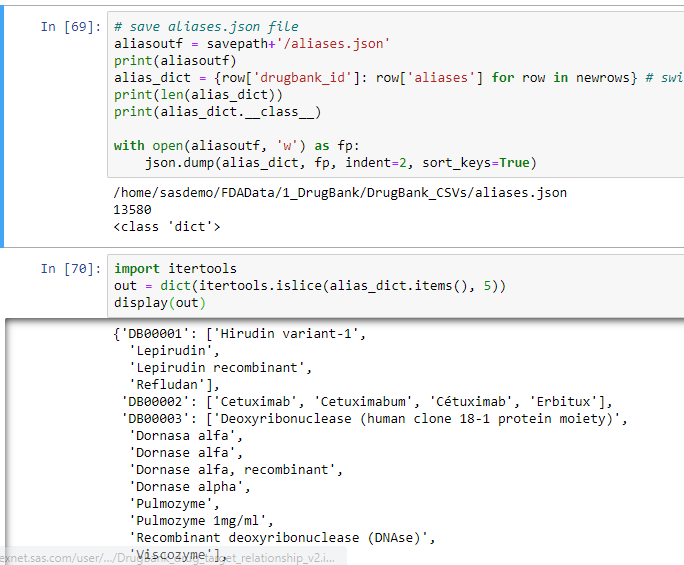
print(entrez\_df.shape)



### Process: Drugbank-Synonyms

Synonyms are extracted for each drug from the “synonyms”, “products” and “international-brands” sections of DrugBank.

Below is an example of the synonyms for the first 3 drugs in DrugBank:



Additional synonyms may possibly be extracted by also searching the “mixtures” and “prices” sections of DrugBank shown below:

<mixtures>

<mixture>

<name>Pulmozyme</name>

<ingredients>Dornase alfa</ingredients>

</mixture>

…

</mixtures>

<prices>

<price>

<description>Lufyllin 200 mg tablet</description>

<cost currency="USD">3.21</cost>

<unit>tablet</unit>

</price>

…

</prices>

### Code

# save aliases.json file

aliasoutf = savepath+'/aliases.json'

print(aliasoutf)

alias\_dict = {row['drugbank\_id']: row['aliases'] for row in newrows} # switch from rows to newrows

print(len(alias\_dict))

print(alias\_dict.\_\_class\_\_)

with open(aliasoutf, 'w') as fp:

json.dump(alias\_dict, fp, indent=2, sort\_keys=True)



## FDA COVID Clinical Trials

### Description

The FDA COVID Clinical Trials table is used to extract the small molecule drugs being investigated for use with COVID treatment. The drugs under investigation then represent a version of “the truth” or “expected” set of drugs that can be used to validate results of the modeling process.

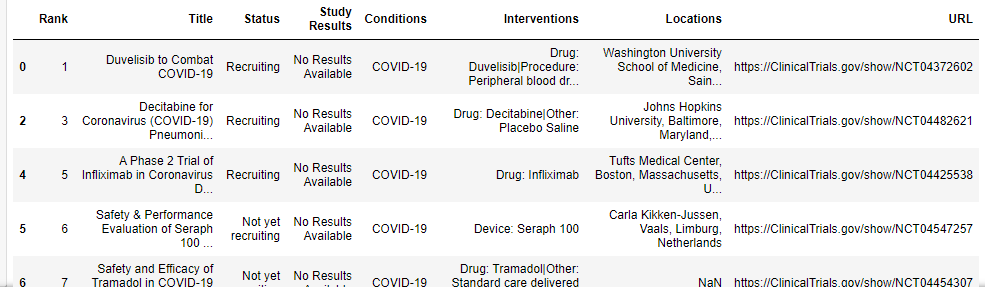
### Source

<https://clinicaltrials.gov/ct2/results?cond=COVID-19>

### Process

The Clinical Trials data is an semi-structured EXCEL spreadsheet, requiring the drug names to be extracted from the Interventions column and matched to the DrugBank synonym file. Other columns such as Title, status and “Study Results” could be further parsed to extract data if needed

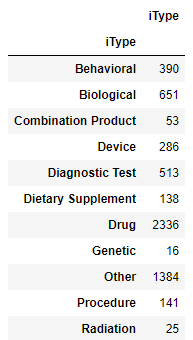
Below is a sample set of rows from the initial FDA Clinical Trials Input Table



To extract and standardize the list of drugs, we

* parse out each intervention separated by a pipe delimiter “|”
* within each intervention, parse out INTERVENTION\_TYPE as the portion before the “:” and the INTERVENTION\_NAME as the portion after the “:”

The count of each INTERVENTION\_TYPE observed is shown below:

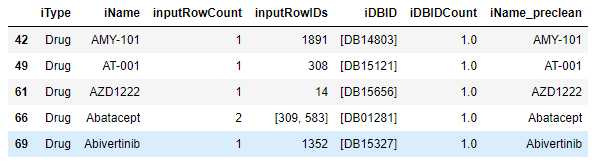


Our main interest is in the INTERVENTION\_TYPE of “Drug”, however some of the INTERVENTION\_NAME values from other INTERVENTION\_TYPES seem to be Drug names and some of type “Drug” do not match into DrugBank.

Matching on names causes particular issues, however we have implemented some name cleansing techniques such as using all upper-case, eliminating white-space, special characters and known other modifiers such as “oz”, or “liquid”.

To further refine the “truth” drug list, we have also implemented a feedback loop for manual review and cleansing of the names after an initial match to the DrugBank synonyms list (see the reference in the drugbank section).

The final list of names then goes through a final match to DrugBank synonyms list to extract the drugbank ID for each identified small-molecule drug.



This final list of intervention drugbank IDs (iDBID) is then exported as a final version of “the truth”.

### Code

display("reading: " + CTPath + 'clinicaltrial\_covid.csv')

ctrials\_df = pd.read\_csv(CTPath + 'clinicaltrial\_covid.csv')

ctrials\_df.dropna(subset=['Interventions'], inplace=True) # if interventions column is blank. drop row

display(ctrials\_df.head())

ctrials\_df.set\_index(['Rank'], inplace=True)

ctrials\_df['s1'] = ctrials\_df['Interventions'].str.split('|')

display(ctrials\_df.shape)

display(ctrials\_df['Interventions'].\_\_class\_\_)

display(ctrials\_df['s1'].\_\_class\_\_)

display(ctrials\_df.\_\_class\_\_)

display(ctrials\_df[['Interventions','s1']].head())

# now create one row for each array element in column s1

ctrials\_df = explode(ctrials\_df, 's1', preserve\_index=True)

ctrials\_df.reset\_index(inplace=True)

display(ctrials\_df.shape)

display(ctrials\_df.head())

# keep only rows with intervention type of "Drug"

ctrials\_drugs\_df = ctrials\_df[ ctrials\_df['iType'] == 'Drug' ]

# check results

display('ctrials\_drugs\_df')

display(ctrials\_drugs\_df.shape)

# get input row IDs for each iType + iName

print()

print('df2')

df2 = pd.DataFrame(ctrials\_drugs\_df[['iType', 'iName', 'index']].groupby(['iType', 'iName']).aggregate(lambda tdf: np.array(tdf.unique())))

df2.columns = ['inputRowIDs']

print(df2.shape)

print(df2['inputRowIDs'].\_\_class\_\_)

# counting number of row IDs for each iType + iName

Drug\_iNames\_df = pd.DataFrame(ctrials\_drugs\_df.groupby(['iType', 'iName'])['iType'].count())

Drug\_iNames\_df.columns = ['inputRowCount']

Drug\_iNames\_df['inputRowIDs'] = df2['inputRowIDs']

Drug\_iNames\_df.reset\_index(inplace=True)

display(Drug\_iNames\_df.shape)

display(Drug\_iNames\_df[15:20])

# cleanup memory

del df2

# keep only rows with intervention type of "Drug"

ctrials\_drugs\_df = ctrials\_df[ ctrials\_df['iType'] == 'Drug' ]

# check results

display('ctrials\_drugs\_df')

display(ctrials\_drugs\_df.shape)

# get input row IDs for each iType + iName

print()

print('df2')

df2 = pd.DataFrame(ctrials\_drugs\_df[['iType', 'iName', 'index']].groupby(['iType', 'iName']).aggregate(lambda tdf: np.array(tdf.unique())))

df2.columns = ['inputRowIDs']

print(df2.shape)

print(df2['inputRowIDs'].\_\_class\_\_)

# counting number of row IDs for each iType + iName

Drug\_iNames\_df = pd.DataFrame(ctrials\_drugs\_df.groupby(['iType', 'iName'])['iType'].count())

Drug\_iNames\_df.columns = ['inputRowCount']

Drug\_iNames\_df['inputRowIDs'] = df2['inputRowIDs']

Drug\_iNames\_df.reset\_index(inplace=True)

display(Drug\_iNames\_df.shape)

display(Drug\_iNames\_df[15:20])

# cleanup memory

del df2

import json

fObj = open(DBPath+'DrugBank\_CSVs/aliases.json',)

DBAlias = json.load(fObj)

fObj.close()

print(DBAlias.\_\_class\_\_)

display(DBAlias['DB12466'])

display(DBAlias["DB15327"])

# search dict in DBAlias format to return ID if element of an array matches

def searchDBAlias(byVal):

keysList = []

itemsList = DBAlias.items()

for item in itemsList:

if byVal in item[1]:

keysList.append(item[0])

return keysList

# this could be sped up if it becomes an issue

start = time.time()

Drug\_iNames\_df['iDBID'] = Drug\_iNames\_df['iName'].apply(searchDBAlias)

lapse = time.time() - start

print("lapse time to match: ", lapse)

display(Drug\_iNames\_df[Drug\_iNames\_df['iDBIDCount']==1].head())

# get list of possible stop-words by breaking out every word (based on space, %, etc) into a separate word

Drug\_iNames\_nomatch\_df = Drug\_iNames\_df[ Drug\_iNames\_df['iDBIDCount'] == 0 ].copy()

# a regular expression replacement would be more efficient - but this runs fast and I'm pretty quick at copy /paste

Drug\_iNames\_nomatch\_df['iName'] = Drug\_iNames\_nomatch\_df['iName'].str.replace('"', " ")

Drug\_iNames\_nomatch\_df['iName'] = Drug\_iNames\_nomatch\_df['iName'].str.replace("%", " ")

Drug\_iNames\_nomatch\_df['iName'] = Drug\_iNames\_nomatch\_df['iName'].str.replace(":", " ")

…

Drug\_iNames\_nomatch\_df['iName'] = Drug\_iNames\_nomatch\_df['iName'].str.replace("8", " ")

Drug\_iNames\_nomatch\_df['iName'] = Drug\_iNames\_nomatch\_df['iName'].str.replace("9", " ")

Drug\_iNames\_nomatch\_df['iName'] = Drug\_iNames\_nomatch\_df['iName'].str.strip()

# examine most common remaining terms as candidates for exclusion

stop\_words\_df = Drug\_iNames\_nomatch\_df['iName'].str.upper().str.split(" ", expand=True)

stop\_words\_df['iName'] = Drug\_iNames\_nomatch\_df['iName']

stop\_words\_df = pd.melt(stop\_words\_df, id\_vars=['iName'])

stop\_words\_df["value"]= stop\_words\_df.value.str.replace('.', ' ')

stop\_words\_df["value"]= stop\_words\_df.value.str.replace(' +', ' ')

stop\_words\_df["value"]= stop\_words\_df.value.str.strip()

stop\_words\_df.dropna(subset=['value'], inplace=True) # if word value column is blank. drop row

print("--" + stop\_words\_df.value[0] + "--")

print("--" + stop\_words\_df.value[1] + "--")

stop\_words\_df = stop\_words\_df[stop\_words\_df['value'] != ""]

stop\_words\_df.sort\_values(by=['iName', 'variable'], ascending=True, inplace=True)

display(stop\_words\_df.shape)

display(stop\_words\_df.loc[:20])

display(stop\_words\_df[stop\_words\_df['value']=='HYDROXYCHLOROQUINE'])

# get count of cleaned terms

stop\_wordsC\_df = pd.DataFrame(stop\_words\_df.groupby(['value'])['value'].count())

stop\_wordsC\_df.columns=['count']

stop\_wordsC\_df.reset\_index(inplace=True)

stop\_wordsC\_df.sort\_values(by='count', ascending=False, inplace=True)

display(stop\_wordsC\_df.shape)

display(stop\_wordsC\_df.loc[:5])

# now set up lists of terms to exclude from intervention names

stop\_wordsC\_df['conjunction'] = stop\_wordsC\_df.value.str.upper().isin(['IN', 'OF', 'AND', 'PLUS', 'OR', 'FOR', 'A', 'WITH'])

stop\_wordsC\_df['space'] = stop\_wordsC\_df.value.isin([' ', '.', ':', ''])

stop\_wordsC\_df['dose'] = stop\_wordsC\_df.value.str.upper().isin(['INHALATION', 'ML', 'MG', 'DOSE', 'ORAL', 'TABLETS', 'TABLET', 'ML', 'DAYS', 'INJECTION', 'KG'])

stop\_wordsC\_df['placebo'] = stop\_wordsC\_df.value.str.upper().isin(['NORMAL', 'AGENT', 'PLACEBO', 'SALINE', 'STANDARD', 'TREATMENT', 'CARE', 'THERAPY', 'NASAL', 'FLOW'])

stop\_wordsC\_df['concept'] = stop\_wordsC\_df.value.str.upper().isin(['CoV', 'COVID', 'DRUG', 'PRODUCT', 'SOLUTION', 'PLASMA'])

termstoshow = 10

display(stop\_wordsC\_df[(stop\_wordsC\_df['concept'] == False) & (stop\_wordsC\_df['placebo'] == False) & (stop\_wordsC\_df['conjunction'] == False) & (stop\_wordsC\_df['space'] == False) & (stop\_wordsC\_df['dose'] == False)].head(termstoshow))

stopWords = ['0.12%', '0.075%', '100 mg']

Drug\_iNamesClean\_df = Drug\_iNames\_df

Drug\_iNamesClean\_df['iName\_preclean'] = Drug\_iNamesClean\_df['iName']

display(Drug\_iNamesClean\_df.head())

# read manual exception override file

iName\_Overrides\_df = pd.read\_csv(CTPath + 'ctrials\_iname\_overrides.csv', delimiter='\t')

display(iName\_Overrides\_df.head())



# apply exception override file to rename Drugs with more complex / unique patterns if needed

Drug\_iNamesNew\_df = pd.merge(Drug\_iNames\_df, iName\_Overrides\_df, how='left', on='iName', indicator=True)

display(Drug\_iNamesNew\_df['\_merge'].value\_counts())

Drug\_iNamesNew\_df.rename(columns = {'iName':'iNameOld'}, inplace = True)

display(Drug\_iNamesNew\_df.head())

NOTE: need to update this section with newer code before publishing

# keep rows with a match into DrugBank synonyms

ProposedTruth\_df = Drug\_iNames\_df[ Drug\_iNames\_df['iDBIDCount'] > 0 ].copy()

display(ProposedTruth\_df.head())

display(ProposedTruth\_df.iDBID.\_\_class\_\_)

display(ProposedTruth\_df.iDBID.shape)

display(ProposedTruth\_df.iDBID.iloc[1])

display(ProposedTruth\_df.iDBID.iloc[1].\_\_class\_\_)

s = ProposedTruth\_df.apply(lambda x: pd.Series(x['iDBID']),axis=1).stack().reset\_index(level=1, drop=True)

s.name = 'iDBID'

ProposedTruth\_df = ProposedTruth\_df.drop('iDBID', axis=1).join(s)

display(ProposedTruth\_df[10:30])

ProposedTruth\_df.to\_csv("ProposedTruth.csv")

## STRING Protein Protein Interactions

### Description

The STRING Protein Protein interaction data lists pairs of proteins using their Ensemble-PRO ID with an association strength score. One use for this data is dimension reduction by creating groups or clusters of proteins with high degrees of interaction

It is worth noting that there are a small number of proteins linked to nearly every other protein. Care should be taken in using those entries as they may cause links between large numbers of proteins.

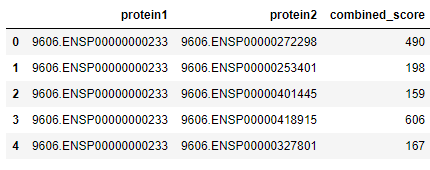
### Source

<https://string-db.org/cgi/download?sessionId=blXUTlQ4pHMK>

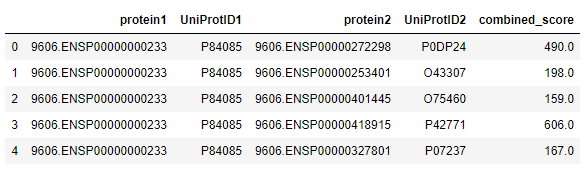
### Process

The STRING protein-protein interaction table was fairly clean, consisting only of the pair of proteins and their interaction score. The only processing required was matching of the Ensemble-PRO protein IDs in columns 1 and 2 with the UniProt-IDs to support referencing into DrugBank drug-protein table.





Here is a sample of the STRING data post matching with UniProtIDs appended:



### Code

stringPP\_df = pd.read\_csv(STRINGPath + '9606.protein.links.v11.0.txt', sep=' ')

display(stringPP\_df.shape)

display(stringPP\_df.head())

# merge Protein 1 to STRING on UniProt .dat file

stringPP\_df = pd.merge(stringPP\_df, UniProtSTRINGIDs\_df, how="left", left\_on='protein1', right\_on='STRINGID', indicator=True)

#display(stringPP\_df['\_merge'].value\_counts())

stringPP\_df.rename(columns = {'UniProtID':'UniProtID1', 'STRINGID':'STRINGID1'}, inplace = True)

stringPP\_df.drop(columns=['\_merge'], inplace=True)

# merge Protein 1 to Ensembl\_PRO on UniProt .dat file

stringPP\_df = pd.merge(stringPP\_df, UniProtEnsemblePro\_df[['UniProtID', 'Ensembl\_PRO9606']], how="left", left\_on='protein1', right\_on='Ensembl\_PRO9606', indicator=True)

#display(stringPP\_df['\_merge'].value\_counts())

# coalesce Protein 1 STRING and Ensemble\_PRO columns (use Ensembl\_PRO if STRING is missing)

stringPP\_df['UniProtID1'] = stringPP\_df['UniProtID1'].mask(pd.isnull, stringPP\_df['UniProtID'])

stringPP\_df.rename(columns = {'Ensembl\_PRO9606':'Ensembl\_PRO96061'}, inplace = True)

stringPP\_df.drop(columns=['\_merge', 'UniProtID'], inplace=True)

# merge protein 2 to STRING on UniProt .dat file

stringPP\_df = pd.merge(stringPP\_df, UniProtSTRINGIDs\_df, how="left", left\_on='protein2', right\_on='STRINGID', indicator=True)

#display(stringPP\_df['\_merge'].value\_counts())

stringPP\_df.rename(columns = {'UniProtID':'UniProtID2', 'STRINGID':'STRINGID2'}, inplace = True)

stringPP\_df.drop(columns=['\_merge'], inplace=True)

# merge Protein 2 to Ensembl\_PRO on UniProt .dat file

stringPP\_df = pd.merge(stringPP\_df, UniProtEnsemblePro\_df[['UniProtID', 'Ensembl\_PRO9606']], how="left", left\_on='protein2', right\_on='Ensembl\_PRO9606', indicator=True)

# coalesce Protein 2 STRING and Ensemble\_PRO columns (use Ensembl\_PRO if STRING is missing)

stringPP\_df['UniProtID2'] = stringPP\_df['UniProtID2'].mask(pd.isnull, stringPP\_df['UniProtID'])

stringPP\_df.rename(columns = {'Ensembl\_PRO9606':'Ensembl\_PRO96062'}, inplace = True)

stringPP\_df.drop(columns=['\_merge', 'UniProtID'], inplace=True)

print()

print('After matching to STRING and Ensembl\_PRO in UniProt')

display(stringPP\_df.shape)

print("Rows with no match for STRING: ", stringPP\_df.STRINGID1.isnull().count())

print("Rows with no match for Ensebml\_PRO: ", stringPP\_df.Ensembl\_PRO96061.isnull().count())

print("Rows with no match for either: ", stringPP\_df.UniProtID1.isnull().count())

#print("Rows with no match for either: ", stringPP\_df.UniProtID1.isnull().value\_counts())

print("NOTE: next most likely reason for missing is depricated IDs. see note in markup cell below")

print()

display(stringPP\_df.head())

print()

print('Sample of Null matches:')

display(stringPP\_df[stringPP\_df.UniProtID1.isnull()].head())

#display(stringPP\_df[stringPP\_df.UniProtID1.isnull()].groupby('protein1')['protein1'].count())

print("unique IDs with no match: ", stringPP\_df[stringPP\_df.UniProtID1.isnull()].groupby('protein1')['protein1'].count().count())

# reset dataframe to contain only relevant columns

stringPP\_df = stringPP\_df[['protein1', 'UniProtID1', 'protein2', 'UniProtID2', 'combined\_score']]

# Joining Tables

UniProt can be used to harmonize Protein IDs across the various data sources (DrugBank, Virus-Host Proteins and STRING)

The Virus-Host proteins can be directly matched to DrugBank to identify drugs that are associated with the proteins the Virus(es) effect in the host.

STRING can be directly joined to the DrugBank via the drug-proteins OR the Virus-Host Proteins, but this will expand the dimensionality of the 2nd order (or higher) associated proteins. An alternative use is to use the STRING database for dimension reduction by creating groups or clusters of associated Proteins.

The FDA Clinical Trials data can be joined to DrugBank on Drug-Bank ID (after cleansing of the clinical trials drug-names and matching through the DrugBank synonym table)

