Web_Scraping_DLiP_Advanced_Search

November 30, 2023

1 Data scraping project - Advanced Search

In this project, data scraping techniques will be employed to extract information from the DLiP dataset within the advanced search section. This necessity arises from the absence of an explicit means to download the dataset directly, thus compelling the utilization of data scraping methodologies to acquire the requisite data.

In the context of our research endeavors, the procurement of this dataset is fundamental for its subsequent integration into our deep learning models. The overarching goal is to leverage this dataset for predictive analyses, encompassing phenomena such as the anticipation of protein-protein interactions and similar biological event.

In the context of this undertaking, we shall employ the *Selenium* and *BeautifulSoup* libraries

Source: DLiP data base website -> https://skb-insilico.com/dlip.

1.1 Import libraries

[1]:
```
import os
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import StaleElementReferenceException
import urllib.parse
import time
import re
from bs4 import BeautifulSoup
import pandas as pd
```

1.2 Getting started

The subsequent procedure entails ensuring the availability of the appropriate web driver for our website, be it Chrome or an alternative browser

Subsequently, it is imperative to incorporate said driver into our system's environment PATH.

[37]:
```
os.environ['PATH'] += r'C:\Users\gavvi\ChromeDrivers\chrome-win64\chrome-win64'
```

1.3 Performing Data Scraping Techniques to Extract the Dataset

The next phase encompasses several steps:

- First, open the DLip dataset website, execute data scraping procedures, including pressing buttons to filter samples and retrieve them. Also interact with the search button to obtain the filtered data (i.e. by pressing the "*All* button).
- Next, initialize an empty DataFrame to be used later for filling it with the read data. The objective is to create an empty dataset containing all the columns and their names from the dataset on the website.
- Create a function to update the DataFrame with the current webpage data. This function includes tasks such as swapping the data under the *Mol_Image* column with its corresponding Canonical SMILES (RDKit) value.
- Finally, create a function that takes the driver, base URL, current page number, and page threshold number. The function iterates until it reaches the threshold number, utilizing the previous function to extract information from every page. By pressing the "Next" button, it moves to the next.e.eled.

1.3.1 Web Scraping Setup

The first step is to open the chrome website and navigate to the advanced search data set in the DLiP website.

[20]:
```
# Open chrome website
driver = webdriver.Chrome()

# Use the correct url in order to navigate to the correct data set in the DLiP␣
 ↪data base website
url = "https://skb-insilico.com/dlip/compound-search/ppi-library/rule-based"
driver.get(url)

driver.implicitly_wait(5)
```

[39]:
```
# Seatch for the "All" button and click on it.
all_btn = driver.find_element(By.XPATH, "//button[text()=' All']")
all_btn.click()

driver.implicitly_wait(2)
```

[40]:
```
# Search for the green colored "Search" button, and click on it.
search_btn = WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.CLASS_NAME, 'btn-green'))
)
search_btn.click()
```

1.3.2 Empty DataFrame Initialization

The next step involves creating an empty DataFrame with the column titles found on the DLiP database website's advanced search dataset. This blank DataFrame will be used later in a process where it will go through a set of steps and functions to eventually be filled with data from a table.

[41]:
```
# Get the table of the current page
table = driver.find_element(By.CLASS_NAME, "dataTables_scrollBody")

# Extract the HTML content of the table
table_html = table.get_attribute('outerHTML')

# Use BeautifulSoup library to parse the HTML
soup = BeautifulSoup(table_html, 'html.parser')


header = [th.text for th in soup.find_all('th')]
df = pd.DataFrame([], columns=header)
```

[42]:
```
df
```

[42]:
```
Empty DataFrame
Columns: [DLiP-ID, Mol Image, MW, XLogP, HBA, HBD, PSA, nRotatableBonds, nRings]
Index: []
```

1.3.3 DataFrame Update Function

[21]:
```
def update_dataframe_on_new_page(driver, base_url, existing_df, extract):
    """
    This function utilizes a set of inputs to update the 'existing_df' with␣
 ↪data extracted from the currently displayed website page.

    Parameters:

    driver: The web driver used (e.g., Chrome, Firefox).
    base_url: The base URL for navigating to the previous page, especially when␣
 ↪moving to another HTML file to extract the molecule's Canonical SMILES value.
    existing_df: The foundational DataFrame that undergoes updates at each step.
    extract: A boolean value indicating whether additional information, such as␣
 ↪the molecule's Canonical SMILES, should be extracted.
    """
    # Extract the HTML content of the table
    table_html = driver.find_element(By.CLASS_NAME, "dataTables_scrollBody").
 ↪get_attribute('outerHTML')

    # Use BeautifulSoup to parse the HTML
    soup = BeautifulSoup(table_html, 'html.parser')
```

```
    data = []

    # Extract table data manually
    for row in soup.find_all('tr')[1:]:
        row_data = [td.text for td in row.find_all('td')]

        if extract == True:
            # Extract the DLiP-ID and Canonical SMILES(RDKit) links
            dlip_id_link = row.find('a', {'href': re.compile(r'/dlip/compound/
↪')})
            smiles_link = row.find('a', {'href': re.compile(r'/dlip/compound/
↪[A-Z]\d+')}),

            # Navigate to the DLiP-ID link
            dlip_id_url = urllib.parse.urljoin(base_url, dlip_id_link['href'])
            driver.get(dlip_id_url)

            # Extract the Canonical SMILES(RDKit) value
            smiles_value = driver.find_element(By.XPATH, '//
↪td[text()="Canonical SMILES(RDKit)"]/following-sibling::td').text

            # Replace the Mol Image value with the Canonical SMILES(RDKit)
            mol_image_index = existing_df.columns.get_loc("Mol Image")
            row_data[mol_image_index] = smiles_value

            # Return to the initial page
            driver.back()

        # Append the modified row_data to the DataFrame
        data.append(row_data)

    # Ensure the columns are in the correct order
    new_df = pd.DataFrame(data, columns=existing_df.columns)

    # Concatenate DataFrames
    updated_df = pd.concat([existing_df, new_df], ignore_index=True)

    return updated_df
```

1.3.4 Iterative Web Page Extraction Step

[22]:
```
def get_batch_of_table(driver, base_url, df, page_number_, page_threshold,␣
↪extract):
    """
    This function takes a set of parameters and iterates from "page_number_"␣
↪until it reaches the specified "page_threshold." For each page,
```

```
   it utilizes the update_dataframe_on_new_page(driver, base_url, existing_df,␣
↪extract) function to update the DataFrame.

   pameters:
   driver: The web driver used (e.g., Chrome, Firefox).
   base_url: The base URL for navigating to the previous page, especially when␣
↪moving to another HTML file to extract the molecule's Canonical SMILES value.
   df: The foundational DataFrame that undergoes updates at each step.
   page_number_: The current page number the website is opened on.
   page_threshold: The page at which the while loop stops iterating.
   extract: A boolean value indicating whether additional information, such as␣
↪the molecule's Canonical SMILES, should be extracted.

   """

   # Loop through pages until the last page
   page_number = page_number_
   while page_number <= page_threshold:
       try:
           # Wait for the loading overlay to disappear
           WebDriverWait(driver, 120).until(
               EC.invisibility_of_element_located((By.CLASS_NAME,␣
↪"loadingoverlay"))
           )

           # Update the old dataframe with the content of the next website␣
↪page using our helper function
           df = update_dataframe_on_new_page(driver, base_url, df, extract)

           if page_number < page_threshold:
               # Find the "Next" button
               next_button = driver.find_element(By.XPATH, '//
↪*[@id="compound-list-table_next"]/a')

               # Click on the "Next" button
               next_button.click()

               # Wait for the table to be present on the next page
               table = WebDriverWait(driver, 120).until(
                   EC.presence_of_element_located((By.CLASS_NAME,␣
↪"dataTables_scrollBody"))
               )

       except StaleElementReferenceException:
           continue
```

```
        page_number += 1

    return df
```

1.4 Get the data

The subsequent phase involves leveraging the previously established steps to extract the data in batches. This approach is adopted due to the abundance of pages (exceeding 600), coupled with potential website crashes or machine errors that could compromise the program's continuity. To mitigate this risk, the data extraction process will be conducted in batches, typically consisting of 200-300 pages. Each batch will be saved as a CSV file, ensuring data preservation in the event of program interruptions. Upon completion of the entire extraction process, the accumulated dataframes will be merged to construct the final dataframe encompassing all the rows.

[27]:
```
page_threshold = 609
page_number_ = 600
base_url = "https://skb-insilico.com"

df = get_batch_of_table(driver, base_url, df, page_number_, page_threshold,␣
 ↪True)
```

[28]:
```
df.to_csv("2811_ppi_first_609.csv", index=False)
```

[15]:
```
df
```

[15]:

	DLiP-ID	Mol Image	MW	XLogP	HBA	HBD	PSA	nRotatableBonds	nRings
0	D00000	COc1cccc2c1OCC21CCN(C(=O)CC2(c3cccc(Br)c3)CCNC…	499.449	3.49	4	1	50.8	4	5
1	D00001	COc1ccccc1Cn1nc(C)c(C(=O)N2CCC(CN3Cc4ccc(F)cc4…	490.579	4.05	5	0	67.67	6	5
2	D00002	COc1cccc(C2(CC(=O)NC3(Cc4ccc(Cl)cc4)CCS(=O)(=O…	505.08	3.868	5	2	84.5	7	4
3	D00003	NC1CCN(Cc2ccccc2C(=O)NC2CCN(Cc3ccccc3C(=O)O)CC…	450.583	2.666	5	3	98.9	7	4
4	D00004	COc1cccc(C2(CC(=O)Nc3ccccc3N3CCC(C(=O)O)CC3)CC…	451.567	2.902	5	3	90.9	7	4
…	…	…	…	…	…	…	…	…	…
12520	D009I2	O=C(NC1CCN(C(=O)c2cccc(Br)c2)CC1)NC12CC3CC(CC(…	460.416	4.928	2	2	61.44	3	6
12521	D009I3	COc1cccc(C2(C(=O)NC3(Cc4cccc(F)c4)CCC(F)(F)CC3…	462.512	5.276	4	1	60.45	6	4
12522	D009I4	COc1cccc(C2(C(=O)NC3CCN(Cc4ccc(C(=O)O)cc4)CC3)…	451.567	3.073	5	3	90.9	7	4
12523	D009I5	CN1CCN(c2ccc(F)cc2NC(=O)C2(Cc3ccc(F)cc3)CCC(F)…	463.519						
12524	D009I6	O=C(NC1(Cc2ccncc2)CCC(F)(F)CC1)c1nnc2cc(Br)ccn12	450.287						

```
12523  5.019   3   1  35.58              5      4
12524  5.738   5   1  72.18              4      4

[12525 rows x 9 columns]
```

1.5 Merge the data frames and check for duplicated values

Following the extraction of data in batches, the subsequent step involves merging the individual dataframes into a unified dataframe. It is imperative to ensure the absence of duplicated rows within this consolidated dataframe. The necessity for this verification arises from the counting mechanism employed in the `get_batch_of_table()` function, wherein the first page of each batch is counted twice. To rectify this, the removal of duplicated rows becomes crucial to maintain the integrity of the final dataframe.

[46]:
```
file1_path = '2811_ppi_first_500_.csv'
file2_path = '2811_ppi_first_609_.csv'

df1 = pd.read_csv(file1_path)
df2 = pd.read_csv(file2_path)

# Concatenate the DataFrames vertically (one after the other)
merged_df = pd.concat([df1, df2], ignore_index=True)

# Remove duplicate rows across all columns and resetting index after dropping␣
 ↪duplicates
merged_df.drop_duplicates(inplace=True)
merged_df.reset_index(drop=True, inplace=True)
```

[30]:
```
merged_df.to_csv("ppi_609_Dataset.csv", index=False)
```

[34]:
```
merged_df_no_duplicated = merged_df

merged_df_no_duplicated
```

[34]:
```
      DLiP-ID                                              Mol Image       MW  \
0      D00000  COc1cccc2c1OCC21CCN(C(=O)CC2(c3cccc(Br)c3)CCNC…  499.449
1      D00001  COc1ccccc1Cn1nc(C)c(C(=O)N2CCC(CN3Cc4ccc(F)cc4…  490.579
2      D00002  COc1cccc(C2(CC(=O)NC3(Cc4ccc(Cl)cc4)CCS(=O)(=O…  505.080
3      D00003  NC1CCN(Cc2ccccc2C(=O)NC2CCN(Cc3ccccc3C(=O)O)CC…  450.583
4      D00004  COc1cccc(C2(CC(=O)Nc3ccccc3N3CCC(C(=O)O)CC3)CC…  451.567
…        …                                                  …       …
15136  D00BQH  CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)NC…  598.748
15137  D00BQI  COCCOc1cccnc1C(=O)N[C@H]1CC[C@H](NCc2cccc(C(=O…  643.760
15138  D00BQJ  O=C(Nc1ccccc1Cl)NC1CCN(C(=O)c2cccc(CN[C@H]3CC[…  620.238
15139  D00BQK  CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)N4…  625.770
15140  D00BQL  COc1ccc(C(C)(C)CC(=O)N[C@@H]2CCN(c3ccc(F)cc3NC…  645.751

       XLogP  HBA  HBD     PSA  nRotatableBonds  nRings
```

```
0       3.490    4    1   50.80                4       5
1       4.050    5    0   67.67                6       5
2       3.868    5    2   84.50                7       4
3       2.666    5    3   98.90                7       4
4       2.902    5    3   90.90                7       4
…       …   …   …     …                …      …
15136   4.200    7    3  125.55               11       5
15137   3.907    7    2  113.10               11       6
15138   6.435    4    4  102.57                8       6
15139   5.002    7    2  113.10               10       6
15140   5.476    7    2   97.72               10       6

[15141 rows x 9 columns]
```

[35]:
```
merged_df = pd.concat([df1, df2], ignore_index=True)

merged_df
```

[35]:
```
       DLiP-ID                                            Mol Image       MW  \
0       D00000  COc1cccc2c1OCC21CCN(C(=O)CC2(c3cccc(Br)c3)CCNC…  499.449
1       D00001  COc1ccccc1Cn1nc(C)c(C(=O)N2CCC(CN3Cc4ccc(F)cc4…  490.579
2       D00002  COc1cccc(C2(CC(=O)NC3(Cc4ccc(Cl)cc4)CCS(=O)(=O…  505.080
3       D00003  NC1CCN(Cc2ccccc2C(=O)NC2CCN(Cc3ccccc3C(=O)O)CC…  450.583
4       D00004  COc1cccc(C2(CC(=O)Nc3ccccc3N3CCC(C(=O)O)CC3)CC…  451.567
…        …                                                 …       …
15309   D00BQH  CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)NC…  598.748
15310   D00BQI  COCCOc1cccnc1C(=O)N[C@H]1CC[C@H](NCc2cccc(C(=O…  643.760
15311   D00BQJ  O=C(Nc1ccccc1Cl)NC1CCN(C(=O)c2cccc(CN[C@H]3CC[…  620.238
15312   D00BQK  CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)N4…  625.770
15313   D00BQL  COc1ccc(C(C)(C)CC(=O)N[C@@H]2CCN(c3ccc(F)cc3NC…  645.751

        XLogP  HBA  HBD     PSA  nRotatableBonds  nRings
0       3.490    4    1   50.80                4       5
1       4.050    5    0   67.67                6       5
2       3.868    5    2   84.50                7       4
3       2.666    5    3   98.90                7       4
4       2.902    5    3   90.90                7       4
…       …   …   …     …                …      …
15309   4.200    7    3  125.55               11       5
15310   3.907    7    2  113.10               11       6
15311   6.435    4    4  102.57                8       6
15312   5.002    7    2  113.10               10       6
15313   5.476    7    2   97.72               10       6

[15314 rows x 9 columns]
```

[36]:
```
unique_count_merged_no_dup = merged_df_no_duplicated['DLiP-ID'].nunique()
unique_count_merged = merged_df['DLiP-ID'].nunique()

unique_count_merged_no_dup, unique_count_merged
```

[36]:
```
(15141, 15141)
```

[50]:
```
print("--------------------------------------------------------------------------------")
print(f"The number of unique id's without removing duplicated rows is␣
 ↪{unique_count_merged}")
print(f"The number of unique id's after removing duplicated rows is␣
 ↪{unique_count_merged_no_dup}")
print(f"Is the two values euqal ? ->␣
 ↪{unique_count_merged==unique_count_merged_no_dup}")
print("--------------------------------------------------------------------------------")
```

```
--------------------------------------------------------------------------------
The number of unique id's without removing duplicated rows is 15141
The number of unique id's after removing duplicated rows is 15141
Is the two values euqal ? -> True
--------------------------------------------------------------------------------
```

1.6 Get the data without switching values

To ensure comprehensive data retrieval, the table extraction process will be repeated However, in this iteration, the"Mol_Imag" data will not be switched, eliminating the need for navigating to additional HTML files to extract the Canonical SMILES (RDKit) value for each protein. This modification aims to significantly reduce the running time, enabling a more efficient and expedited extraction of all protein information.

[45]:
```
page_threshold = 609
page_number_ = 1
base_url = "https://skb-insilico.com"

df = get_batch_of_table(driver, base_url, df, page_number_, page_threshold,␣
 ↪False)
```

[ ]:
```
df_first_400 = df
df_first_400.to_csv("as_first400.csv", index=False)
```

[48]:
```
df.to_csv("ppi_609_nm_.csv" , index=False)
```

[47]:
```
df_all = df
```

[46]:
```
df
```

[46]:

	DLiP-ID	Mol Image	MW	XLogP	HBA	HBD	PSA	nRotatableBonds	nRings
0	D00000		499.449	3.49	4	1	50.8	4	5

```
1      D00001            490.579   4.05   5   0   67.67               6       5
2      D00002             505.08  3.868   5   2    84.5               7       4
3      D00003            450.583  2.666   5   3    98.9               7       4
4      D00004            451.567  2.902   5   3    90.9               7       4
…        …       …       …     …   ..   ..     …              …     …
15209  D00BQH            598.748    4.2   7   3  125.55              11       5
15210  D00BQI             643.76  3.907   7   2   113.1              11       6
15211  D00BQJ            620.238  6.435   4   4  102.57               8       6
15212  D00BQK             625.77  5.002   7   2   113.1              10       6
15213  D00BQL            645.751  5.476   7   2   97.72              10       6

[15214 rows x 9 columns]
```

[115]:
```
df_no_duplicated = df
# Remove duplicate rows across all columns and resetting index after dropping␣
 ↪duplicates
df_no_duplicated.drop_duplicates(inplace=True)
df_no_duplicated.reset_index(drop=True, inplace=True)
```

[116]:
```
df_no_duplicated
```

[116]:
```
      DLiP-ID Mol Image       MW  XLogP HBA HBD     PSA nRotatableBonds nRings
0      D00000            499.449   3.49   4   1   50.8                4       5
1      D00001            490.579   4.05   5   0   67.67               6       5
2      D00002             505.08  3.868   5   2    84.5               7       4
3      D00003            450.583  2.666   5   3    98.9               7       4
4      D00004            451.567  2.902   5   3    90.9               7       4
…        …       …       …     …   ..   ..     …              …     …
15122  D00BQ3            598.672  4.427   7   2  117.58               7       5
15123  D00BQ4              501.4  4.425   5   1   59.39               7       4
15124  D00BQ5            522.365   5.88   4   1   56.15               5       4
15125  D00BQ6             639.61  5.864   5   1   76.46               9       5
15126  D00BQ7            504.444  3.895   4   3   62.39               7       4

[15127 rows x 9 columns]
```

1.6.1 Find the correct Canonical SMILES(RDKit) value of each molecule

[7]:
```
def fill_canonical_smiles_values(source_df, target_df) -> None:
    """
    The function gets two data frames: source and target, and try to
    find for each DLiP-ID value of the target_df matching one in the
    source data frame. If the function find match, it's extracts the
    value of the canonical smile located in the 'Mol Image' column
    and puts that in the coressponding cell in the target data frame
    """
```

```
    # Iterate through the rows of df1 with missing "Mol Image" values
    for index, row in target_df[target_df['Mol Image'].isnull()].iterrows():
        # Get the corresponding "DLiP-ID" value
        dlip_id = row['DLiP-ID']

        # Search for the matching row in df2
        matching_row = source_df[source_df['DLiP-ID'] == dlip_id]

        # If a match is found, copy the "Mol Image" value to df1
        if not matching_row.empty:
            target_df.at[index, 'Mol Image'] = matching_row.iloc[0]['Mol Image']
```

[14]:
```
# Read the CSV files into DataFrames
df1 = pd.read_csv('ppi_609_nm_.csv')
df2 = pd.read_csv('ppi_609_Dataset.csv')

fill_canonical_smiles_values(df2, df1)
```

The subsequent stage involves ascertaining the presence of `NaN` values within the `Mol Image` column.

[15]:
```
# Count the number of NaN values in the 'Mol Image' column
nan_count = df1['Mol Image'].isna().sum()

print(f'The number of NaN values in the "Mol Image" column is: {nan_count}')
```

```
The number of NaN values in the "Mol Image" column is: 0
```

[16]:
```
df1
```

[16]:

	DLiP-ID	Mol Image	MW	XLogP	HBA	HBD	PSA	nRotatableBonds	nRings
0	D00000	COc1cccc2c1OCC21CCN(C(=O)CC2(c3cccc(Br)c3)CCNC…	499.449	3.490	4	1	50.80	4	5
1	D00001	COc1ccccc1Cn1nc(C)c(C(=O)N2CCC(CN3Cc4ccc(F)cc4…	490.579	4.050	5	0	67.67	6	5
2	D00002	COc1cccc(C2(CC(=O)NC3(Cc4ccc(Cl)cc4)CCS(=O)(=O…	505.080	3.868	5	2	84.50	7	4
3	D00003	NC1CCN(Cc2ccccc2C(=O)NC2CCN(Cc3ccccc3C(=O)O)CC…	450.583	2.666	5	3	98.90	7	4
4	D00004	COc1cccc(C2(CC(=O)Nc3ccccc3N3CCC(C(=O)O)CC3)CC…	451.567	2.902	5	3	90.90	7	4
…	…	…	…						
15209	D00BQH	CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)NC…	598.748						
15210	D00BQI	COCCOc1cccnc1C(=O)N[C@H]1CC[C@H](NCc2cccc(C(=O…	643.760						
15211	D00BQJ	O=C(Nc1ccccc1Cl)NC1CCN(C(=O)c2cccc(CN[C@H]3CC[…	620.238						
15212	D00BQK	CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)N4…	625.770						
15213	D00BQL	COc1ccc(C(C)(C)CC(=O)N[C@@H]2CCN(c3ccc(F)cc3NC…	645.751						

```
…       …   …   …      …                 …       …
15209  4.200    7    3  125.55               11       5
15210  3.907    7    2  113.10               11       6
15211  6.435    4    4  102.57                8       6
15212  5.002    7    2  113.10               10       6
15213  5.476    7    2   97.72               10       6

[15214 rows x 9 columns]
```

It appears that there are no longer any `NaN` values. This signifies that we have successfully addressed all molecules and, for each unique molecule, identified its corresponding canonical smile value.

1.7 Final steps

The concluding measure entails renaming the column titled *Mol Image* to *Canonical SMILES(RDKit)* and subsequently preserving the resultant data frame as a CSV file for its utilization in subsequent stages of our deep learning models.

[17]:
```
df1.rename(columns={'Mol Image': 'Canonical SMILES(RDKit)'}, inplace=True)
```

[18]:
```
df1
```

[18]:
```
      DLiP-ID                              Canonical SMILES(RDKit)       MW  \
0      D00000  COc1cccc2c1OCC21CCN(C(=O)CC2(c3cccc(Br)c3)CCNC…  499.449
1      D00001  COc1ccccc1Cn1nc(C)c(C(=O)N2CCC(CN3Cc4ccc(F)cc4…  490.579
2      D00002  COc1cccc(C2(CC(=O)NC3(Cc4ccc(Cl)cc4)CCS(=O)(=O…  505.080
3      D00003  NC1CCN(Cc2ccccc2C(=O)NC2CCN(Cc3ccccc3C(=O)O)CC…  450.583
4      D00004  COc1cccc(C2(CC(=O)Nc3ccccc3N3CCC(C(=O)O)CC3)CC…  451.567
…       …                                                  …       …
15209  D00BQH  CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)NC…  598.748
15210  D00BQI  COCCOc1cccnc1C(=O)N[C@H]1CC[C@H](NCc2cccc(C(=O…  643.760
15211  D00BQJ  O=C(Nc1ccccc1Cl)NC1CCN(C(=O)c2cccc(CN[C@H]3CC[…  620.238
15212  D00BQK  CCOc1cccc(C(=O)N[C@H]2CC[C@H](NCc3cccc(C(=O)N4…  625.770
15213  D00BQL  COc1ccc(C(C)(C)CC(=O)N[C@@H]2CCN(c3ccc(F)cc3NC…  645.751

       XLogP  HBA  HBD     PSA  nRotatableBonds  nRings
0      3.490    4    1   50.80                4       5
1      4.050    5    0   67.67                6       5
2      3.868    5    2   84.50                7       4
3      2.666    5    3   98.90                7       4
4      2.902    5    3   90.90                7       4
…       …   …   …      …                 …       …
15209  4.200    7    3  125.55               11       5
15210  3.907    7    2  113.10               11       6
15211  6.435    4    4  102.57                8       6
15212  5.002    7    2  113.10               10       6
15213  5.476    7    2   97.72               10       6
```

[15214 rows x 9 columns]

[19]:
```
df1.to_csv("ppi_advanced_search_609.csv", index=False)
```

[12]:
```
df3 = pd.read_csv("ppi_1033_fp.csv")

fill_canonical_smiles_values(df3, df1)
```

[13]:
```
df1.head(50)
```

[13]:

	DLiP-ID	Mol Image	MW	\
0	T00000	CCC(C)(C)C(=O)C(=O)N1CCCCC1C(=O)OCCCc1cc(OC)cc…	433.545	
1	T00001	COc1ccccc1C1C2=C(N=c3s/c(=C\c4ccc(/C=C/C(=O)O)…	520.610	
2	T00002	CSc1ccc(-c2c(C#N)c3cccc(Cl)n3c2NCCc2ccccc2)cc1	417.965	
3	T00003	COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…	519.554	
4	T00004	COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…	519.554	
5	T00005	COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…	519.554	
6	T00006	CC(=O)O[C@H](CC1CC(=O)NC(=O)C1)[C@@H]1C[C@@H](…	323.389	
7	T00007	COc1cc(C=O)ccc1Oc1ccccc1	228.247	
8	T00008	CC(=O)N[C@H](C(=O)N[C@@H](Cc1c[nH]cn1)C(=O)N[C…	717.829	
9	T00009	COc1cc2c(c(OC)c1)C1C3CCCC(C(=O)N1CC2)N3C(=O)N(…	497.595	
10	T0000A	COc1ccccc1Sc1ccc(-c2ccnc(N3CCN(C(C)=O)CC3)c2)c…	487.547	
11	T0000B	O=C(O)CCc1cc(=O)n(CC(=O)NCC2CCC(Nc3nc4ccccc4[n…	501.587	
12	T0000C	CC[C@H](N)C(=O)N[C@@H]1C(=O)N2[C@H](CC[C@H]2C(…	595.788	
13	T0000D	CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc1ccc(OCc2c(Cl…	531.845	
14	T0000E	CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc1ccc(OCc2c(Cl…	531.845	
15	T0000F	Cc1cccc(NC(=O)Nc2nccccc2OCc2ccccc2)c1C	347.418	
16	T0000G	Cc1ccc(NC(=O)CNC(=O)CC23CC4CC(CC(C4)C2)C3)c(O)c1	356.466	
17	T0000H	O=C(O)C[C@H](NC(=O)[C@@H]1CCCN(C(=O)CCC2CCNCC2…	416.522	
18	T0000I	O=C(O)C[C@H](NC(=O)[C@@H]1CCCN(C(=O)CCC2CCNCC2…	416.522	
19	T0000J	O=C(O)C[C@@H](NC(=O)[C@@H]1CCCN(C(=O)CCC2CCNCC…	416.522	
20	T0000K	O=C(O)C[C@@H](NC(=O)[C@@H]1CCCN(C(=O)CCC2CCNCC…	416.522	
21	T0000L	O=C(O)CC(NC(=O)C1CCCN(C(=O)CCC2CCNCC2)C1)c1cccnc1	416.522	
22	T0000M	O=C(O)CC(NC(=O)C1CCCN(C(=O)CCC2CCNCC2)C1)c1cccnc1	416.522	
23	T0000N	CN1Cc2cc(C(=O)N(CCc3ccccc3)Cc3nc4ccccc4[nH]3)c…	511.582	
24	T0000O	O=C(NS(=O)(=O)c1ccc(COc2ccccc2)cc1)c1ccc(-c2cc…	461.514	
25	T0000P	Fc1cccc(F)c1Nc1nc2c(-c3nnc[nH]3)cccc2c2cnccc12	374.354	
26	T0000Q	CC(C)C[C@@H]1C(=O)N[C@@H](CCCN=C(N)N)C(=O)NCC(…	602.693	
27	T0000R	CCCCC(=O)[C@@H]1CCCCN1C(=O)C(=O)C(C)(C)CC	295.423	
28	T0000S	O=C(N[C@@H](Cc1ccc(OCCCNc2ccccn2)cc1)C(=O)O)c1…	488.371	
29	T0000T	O=C(N[C@@H](Cc1ccc(OCCCNc2ccccn2)cc1)C(=O)O)c1…	488.371	
30	T0000U	CCN(CC)c1nc(C)c([N+](=O)[O-])c(NCCNC(=S)Nc2ccc…	451.984	
31	T0000V	CC(C)C[C@H]1NC(=O)[C@H](CC(=O)O)NC(=O)CNC(=O)[…	1138.346	
32	T0000W	CC(C)C[C@H]1NC(=O)[C@H](CC(=O)O)NC(=O)CNC(=O)[…	1138.346	
33	T0000X	CC(C)C[C@H]1NC(=O)[C@H](CC(=O)O)NC(=O)CNC(=O)[…	1138.346	
34	T0000Y	CC(C)C[C@H]1NC(=O)[C@H](CC(=O)O)NC(=O)CNC(=O)[…	1138.346	
35	T0000Z	CC(C)C[C@H]1NC(=O)[C@H](CC(=O)O)NC(=O)CNC(=O)[…	1138.346	

36	T00010	O=C(NC1CCCCCC1)C(Cc1ccccc1)NS(=O)(=O)c1cccc2ns…	458.609
37	T00011	CC(C)c1ccccc1Sc1ccc(-c2cc(N3CCC(C(=O)O)CC3)ncn…	501.574
38	T00012	C/C(=N\Nc1nc2c(F)cccc2s1)c1ccc(-c2ccc(Cl)c(C(=…	429.860
39	T00013	Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…	502.593
40	T00014	Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…	502.593
41	T00015	Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…	502.593
42	T00016	COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…	522.623
43	T00017	COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…	522.623
44	T00018	COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…	522.623
45	T00019	N=C(N)c1cccc(NC(=O)c2ccc3c(c2)CN(CCc2ccccc2)C(…	485.544
46	T0001A	N=C(N)c1cccc(NC(=O)c2ccc3c(c2)CN(CCc2ccccc2)C(…	485.544
47	T0001B	CCCCNS(=O)(=O)c1ccc(OCC(=O)N2CCOCC2)cc1	356.444
48	T0001C	Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N[C@@H](CC(C)C)C(=…	508.619
49	T0001D	Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N[C@@H](CC(C)C)C(=…	508.619

	XLogP	HBA	HBD	PSA	nRotatableBonds	nRings	Common Target Pref Name	\
0	3.548	6	0	82.14	10	2	FKBP1A/FK506	
1	5.492	6	1	80.89	5	6	BCL-like/BAX,BAK	
2	7.388	4	1	40.23	6	4	Neuropilin-1/VEGF-A	
3	5.147	7	2	131.24	10	4	Integrins	
4	5.147	7	2	131.24	10	4	Integrins	
5	5.147	7	2	131.24	10	4	Integrins	
6	1.069	5	1	89.54	4	2	FKBP1A/FK506	
7	2.846	3	0	35.53	4	2	Transthyretin tetramer	
8	-3.475	10	9	278.71	16	3	Cyclophilins	
9	4.363	4	0	62.32	4	6	FKBP1A/FK506	
10	6.234	5	0	45.67	5	4	Integrins	
11	4.112	6	4	129.11	9	5	Integrins	
12	4.859	5	4	116.56	12	5	XIAP/SMAC	
13	4.374	5	2	95.94	8	3	Integrins	
14	4.374	5	2	95.94	8	3	Integrins	
15	4.246	3	2	63.25	5	3	Cyclophilins	
16	5.055	3	3	78.43	5	5	RAC1/TIAM1	
17	0.199	5	3	111.63	8	3	Integrins	
18	0.199	5	3	111.63	8	3	Integrins	
19	0.199	5	3	111.63	8	3	Integrins	
20	0.199	5	3	111.63	8	3	Integrins	
21	0.199	5	3	111.63	8	3	Integrins	
22	0.199	5	3	111.63	8	3	Integrins	
23	3.601	5	3	118.63	8	5	Integrins	
24	6.200	4	1	72.47	7	4	BCL-like/BAX,BAK	
25	5.381	5	2	79.38	3	5	BCL-like/BAX,BAK	
26	1.487	7	7	238.41	10	2	Integrins	
27	2.827	3	0	54.45	7	1	FKBP1A/FK506	
28	3.065	5	3	100.55	11	3	Integrins	
29	3.065	5	3	100.55	11	3	Integrins	
30	5.280	7	3	108.25	9	2	Cyclophilins	

31	-1.785	16	16	493.98	17	3	Integrins
32	-1.785	16	16	493.98	17	3	Integrins
33	-1.785	16	16	493.98	17	3	Integrins
34	-1.785	16	16	493.98	17	3	Integrins
35	-1.785	16	16	493.98	17	3	Integrins
36	4.598	6	2	101.05	7	4	Integrins
37	7.045	5	1	66.32	6	4	Integrins
38	5.482	6	2	87.72	5	4	BCL-like/BAX,BAK
39	2.100	5	4	144.91	9	3	Integrins
40	2.100	5	4	144.91	9	3	Integrins
41	2.100	5	4	144.91	9	3	Integrins
42	4.713	6	3	121.80	9	4	Integrins
43	4.713	6	3	121.80	9	4	Integrins
44	4.713	6	3	121.80	9	4	Integrins
45	3.620	5	5	148.61	8	4	Integrins
46	3.620	5	5	148.61	8	4	Integrins
47	1.050	5	1	84.94	8	2	FKBP1A/FK506
48	3.549	4	4	127.84	9	3	Integrins
49	3.549	4	4	127.84	9	3	Integrins

	Active
0	Active
1	Inactive
2	Active
3	Active
4	Active
5	Active
6	Inactive
7	Active
8	Inactive
9	Active
10	Active
11	Active
12	Active
13	Active
14	Active
15	Active
16	Active
17	Active
18	Active
19	Active
20	Active
21	Active
22	Active
23	Active
24	Active
25	Active

```
26    Active
27    Active
28    Active
29    Active
30    Active
31    Active
32    Active
33    Active
34    Active
35  Inactive
36  Inactive
37    Active
38    Active
39    Active
40  Inactive
41    Active
42    Active
43    Active
44    Active
45    Active
46    Active
47  Inactive
48  Inactive
49  Inactive
```

[ ]: