Web_Scraping_DLiP_Curation_Search

November 30, 2023

1 Data scraping project - PPI Curation Search

In this project, data scraping techniques will be employed to extract information from the DLiP dataset within the *PPI Curation 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 events.

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

[3]:
```
import os
import selenium
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.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.

[57]:
```
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, create a function that opens the DLip dataset website, executes data scraping procedures, including pressing buttons to filter samples and retrieve them. It also interacts with the search button to obtain the filtered data.
- 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. Additionally, the function will assess each row in the column named *Active* to determine if there is an image representing the protein's activity in PPI. If an image is present, the function will label the corresponding row as `Active`; otherwise, it will be labeled as `Inactive`.
- 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 .t page..

1.3.1 Web Scraping Setup

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

[58]:
```
driver = webdriver.Chrome()
url = "https://skb-insilico.com/dlip/compound-search/curated-data/rule-based"

driver.get(url)

driver.implicitly_wait(5)
```

[59]:
```
# Click on the "All" button
all_btn = driver.find_element(By.XPATH, "//button[text()=' All']")
all_btn.click()

driver.implicitly_wait(2)
```

[60]:
```
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.

[61]:
```
# 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)
```

[62]:
```
df
```

[62]:
```
Empty DataFrame
Columns: [DLiP-ID, Mol Image, MW, XLogP, HBA, HBD, PSA, nRotatableBonds, nRings,
Common Target Pref Name, Active]
Index: []
```

1.3.3 DataFrame Update Function

[16]:
```
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')]

        # Check if need to extract the additional information (i.e. proteins␣
↪canonical smile and if he is active or not)
        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+')}),

            if dlip_id_link is not None:
                # 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()

        try:
            # Check if the "fa-check" class exists in the row
            is_active = row.find('i', {'class': 'fa-check'})

            # Update "Active" column based on the presence of the specified␣
↪class
            active_index = existing_df.columns.get_loc("Active")
            row_data[active_index] = "Active" if is_active else "Inactive"
        except Exception as e:
            # Handle the error if there is an issue checking the class
            print(f"Error while checking 'Active' column. Setting as 'Inactive'.
↪ Error: {e}")
            row_data[active_index] = "Inactive"
```

```
        # 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

[17]:
```
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
```

1.3.5 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.

[46]:
```
page_threshold = 1033
page_number_ = 800
base_url = "https://skb-insilico.com"

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

[47]:
```
df.to_csv("2811_active_first_1033.csv", index=False)
```

[15]:
```
df
```

[15]:

	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)…	520.61	
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
…       …                                                  …      …
15020  J0001W     CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2ccc(I)cc2)CC1  505.444
15021  J0001X  CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2cc(F)cc(C(F)(…  465.535
15022  J0001Y    CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2ccc(Br)cc2)CC1  458.444
15023  J0001Z  CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2ccc3c(c2)CCC3…  419.613
15024  J00020  CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2ccc(Cl)c([N+]…   458.99

       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
…      …  ..  ..     …             …    …                     …
15020  6.042   2   1   35.58               7      3              DCN1/UBE2M
15021   6.06   2   1   35.58               7      3              DCN1/UBE2M
15022  5.772   2   1   35.58               7      3              DCN1/UBE2M
15023  5.867   2   1   35.58               7      4              DCN1/UBE2M
15024  5.488   4   1   78.72               8      3              DCN1/UBE2M

         Active
0        Active
1      Inactive
2        Active
3        Active
4        Active
…         …
15020    Active
15021    Active
15022    Active
15023    Active
15024    Active

[15025 rows x 11 columns]
```

1.4 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.

[82]:
```
file1_path = '2811_active_first_600_.csv'
file2_path = '2811_active_first_1033_.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)

duplicates_mask = merged_df.duplicated(keep=False)
duplicates_df = merged_df[duplicates_mask]

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

[ ]:
```
merged_df.to_csv("ppi_1033_Dataset.csv", index=False)
```

[84]:
```
merged_df_no_duplicated = merged_df

merged_df_no_duplicated
```

[84]:

	DLiP-ID	Mol Image	MW	XLogP	HBA	HBD	PSA	nRotatableBonds	nRings
0	J00021	CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2cccc(OC(F)(F)…	463.544	6.141	3	1	44.81	8	3
1	J00022	C=CC(=O)Nc1ccccc1CN(C(=O)Nc1cccc(C(F)(F)F)c1)C…	552.641	5.663	3	2	64.68	8	5
2	J00023	CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2ccc(Cl)cc2)CC1	413.993	5.596	2	1	35.58	7	3
3	J00024	CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2cccc(Cc3ccccc…	469.673	6.940	2	1	35.58	9	4
4	J00025	CC(C1CCC1)N1CCC(N(Cc2ccccc2)C(=O)Nc2cc(F)cc(C(…	477.546	6.017	2	1	35.58	6	4
…	…	…	…	…	…	…	…	…	…
10837	T00010	O=C(NC1CCCCCC1)C(Cc1ccccc1)NS(=O)(=O)c1cccc2ns…	458.609	4.598	6	2	101.05	7	4
10838	T00011	CC(C)c1ccccc1Sc1ccc(-c2cc(N3CCC(C(=O)O)CC3)ncn…	501.574	7.045	5	1	66.32	6	4
10839	T00012	C/C(=N\Nc1nc2c(F)cccc2s1)c1ccc(-c2ccc(Cl)c(C(=…	429.860	5.482	6	2	87.72	5	4
10840	T00013	Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…	502.593	2.100	5	4	144.91	9	3
10841	T00014	Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…	502.593						

```
10841  2.100    5    4  144.91                9       3

      Common Target Pref Name    Active
0                   DCN1/UBE2M    Active
1                   DCN1/UBE2M    Active
2                   DCN1/UBE2M    Active
3                   DCN1/UBE2M    Active
4                   DCN1/UBE2M    Active
…                         …       …
10837                Integrins  Inactive
10838                Integrins    Active
10839         BCL-like/BAX,BAK    Active
10840                Integrins    Active
10841                Integrins  Inactive

[10842 rows x 11 columns]
```

[85]:
```
merged_df = pd.concat([df1, df2], ignore_index=True)

merged_df
```

[85]:
```
       DLiP-ID                                              Mol Image       MW  \
0       J00021  CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2cccc(OC(F)(F)…  463.544
1       J00022  C=CC(=O)Nc1ccccc1CN(C(=O)Nc1cccc(C(F)(F)F)c1)C…  552.641
2       J00023    CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2ccc(Cl)cc2)CC1  413.993
3       J00024  CCCC(C)N1CCC(N(Cc2ccccc2)C(=O)Nc2cccc(Cc3ccccc…  469.673
4       J00025  CC(C1CCC1)N1CCC(N(Cc2ccccc2)C(=O)Nc2cc(F)cc(C(…  477.546
…          …                                                 …       …
21779   T00010  O=C(NC1CCCCCC1)C(Cc1ccccc1)NS(=O)(=O)c1cccc2ns…  458.609
21780   T00011  CC(C)c1ccccc1Sc1ccc(-c2cc(N3CCC(C(=O)O)CC3)ncn…  501.574
21781   T00012  C/C(=N\Nc1nc2c(F)cccc2s1)c1ccc(-c2ccc(Cl)c(C(=…  429.860
21782   T00013  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…  502.593
21783   T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…  502.593

        XLogP  HBA  HBD     PSA  nRotatableBonds  nRings  \
0       6.141    3    1   44.81                8       3
1       5.663    3    2   64.68                8       5
2       5.596    2    1   35.58                7       3
3       6.940    2    1   35.58                9       4
4       6.017    2    1   35.58                6       4
…         …    …    …      …                  …       …
21779   4.598    6    2  101.05                7       4
21780   7.045    5    1   66.32                6       4
21781   5.482    6    2   87.72                5       4
21782   2.100    5    4  144.91                9       3
21783   2.100    5    4  144.91                9       3
```

```
      Common Target Pref Name    Active
0                    DCN1/UBE2M    Active
1                    DCN1/UBE2M    Active
2                    DCN1/UBE2M    Active
3                    DCN1/UBE2M    Active
4                    DCN1/UBE2M    Active
…                          …       …
21779                Integrins  Inactive
21780                Integrins    Active
21781         BCL-like/BAX,BAK    Active
21782                Integrins    Active
21783                Integrins  Inactive

[21784 rows x 11 columns]
```

[86]:
```
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
```

[86]:
```
(10842, 10842)
```

[87]:
```
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 10842
The number of unique id's after removing duplicated rows is 10842
Is the two values euqal ? -> True
--------------------------------------------------------------------------------
```

It appears that numerous molecules were overlooked, as the count of unique IDs significantly decreased after removing duplicate values, falling far below the actual number of rows in the website dataset (10842 « 25817).

Consequently, an alternative approach will be explored, aiming to retrieve the table without extracting each molecule's `Canonical SMILES` value. This adjustment is intended to address potential issues arising from navigating to other HTML pages for individual molecules during the extraction process..

1.5 Get the data without switching values

To ensure comprehensive data retrieval, the table extraction process will be repeated. However, in this iteration, the `Mol_Image` 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.

[65]:
```
page_threshold = 1033
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_600 = df
df_first_600.to_csv("cs_first600.csv", index=False)
```

[66]:
```
df.to_csv("ppi_1033_nm_.csv", index=False)
```

[67]:
```
temp_df = df

temp_df
```

[67]:

	DLiP-ID	Mol Image	MW	XLogP	HBA	HBD	PSA	nRotatableBonds	nRings	Common Target Pref Name	Active
0	T00000		433.545	3.548	6	0	82.14	10	2	FKBP1A/FK506	Active
1	T00001		520.61	5.492	6	1	80.89	5	6	BCL-like/BAX,BAK	Inactive
2	T00002		417.965	7.388	4	1	40.23	6	4	Neuropilin-1/VEGF-A	Active
3	T00003		519.554	5.147	7	2	131.24	10	4	Integrins	Active
4	T00004		519.554	5.147	7	2	131.24	10	4	Integrins	Active
…	…	…	…	..	..	…	…	…	…	…	…
25812	T00010		458.609	4.598	6	2	101.05	7	4	Integrins	Inactive
25813	T00011		501.574	7.045	5	1	66.32	6	4	Integrins	Active
25814	T00012		429.86	5.482	6	2	87.72	5	4	BCL-like/BAX,BAK	Active
25815	T00013		502.593	2.1	5	4	144.91	9	3	Integrins	Active
25816	T00014		502.593	2.1	5	4	144.91	9			

```
25816      3              Integrins  Inactive

[25817 rows x 11 columns]
```

[70]:
```
unique_count_merged_temp = temp_df['DLiP-ID'].nunique()

unique_count_merged_temp
```

[70]:
```
23741
```

[ ]:
```
df.to_csv("ppi_1033_nm_.csv" , index=False)
```

[69]:
```
ppi_609_merged_df = pd.concat([ppi_df_second_part, ppi_df_fourth_part],␣
 ↪ignore_index=True)

duplicates_mask = ppi_609_merged_df.duplicated(keep=False)
duplicates_df = mppi_609_merged_df[duplicates_mask]

unique_count_no_duplicated = ppi_609_merged_df.drop_duplicates(inplace=True)

unique_count_merged = ppi_609_merged_df['DLiP-ID'].nunique()
#unique_count_merged_no_dup = unique_count_no_duplicated['DLiP-ID'].nunique()

unique_count_merged
```

```
None
```

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

[1]:
```
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']
```

[4]:
```
# Read the CSV files into DataFrames
df1 = pd.read_csv('ppi_1033_nm_.csv')
df2 = pd.read_csv('ppi_1033_Dataset.csv')

fill_canonical_smiles_values(df2, df1)
```

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

[5]:
```
# 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: 14926
```

It seems like there is still missing values in some rows. Therefore we will use another partial data frame from previous searches to try fill those empy values.

[6]:
```
df3 = pd.read_csv("ppi_1033_fp.csv")

fill_canonical_smiles_values(df3, df1)
```

[7]:
```
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
```

[9]:
```
df1.head(50)
```

[9]:

	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)Nc2ncccc2OCc2ccccc2)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
```

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.

1.6 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.

[13]:
```
df1.rename(columns={'Mol Image': 'Canonical SMILES(RDKit)'}, inplace=True)
```

[14]:
```
df1
```

[14]:

	DLiP-ID	Canonical SMILES(RDKit)	MW	XLogP	HBA	HBD	PSA	nRotatableBonds	nRings	Common Target Pref Name	Active
0	T00000	CCC(C)(C)C(=O)C(=O)N1CCCCC1C(=O)OCCCc1cc(OC)cc…	433.545	3.548	6	0	82.14	10	2	FKBP1A/FK506	Active
1	T00001	COc1ccccc1C1C2=C(N=c3s/c(=C\c4ccc(/C=C/C(=O)O)…	520.610	5.492	6	1	80.89	5	6	BCL-like/BAX,BAK	Inactive
2	T00002	CSc1ccc(-c2c(C#N)c3cccc(Cl)n3c2NCCc2ccccc2)cc1	417.965	7.388	4	1	40.23	6	4	Neuropilin-1/VEGF-A	Active
3	T00003	COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…	519.554	5.147	7	2	131.24	10	4	Integrins	Active
4	T00004	COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…	519.554	5.147	7	2	131.24	10	4	Integrins	Active
…	…	…	…	…	…	…	…	…	…	…	…
25812	T00010	O=C(NC1CCCCCC1)C(Cc1ccccc1)NS(=O)(=O)c1cccc2ns…	458.609	4.598	6	2	101.05	7	4	Integrins	Inactive
25813	T00011	CC(C)c1ccccc1Sc1ccc(-c2cc(N3CCC(C(=O)O)CC3)ncn…	501.574	7.045	5	1	66.32	6	4	Integrins	Active
25814	T00012	C/C(=N\Nc1nc2c(F)cccc2s1)c1ccc(-c2ccc(Cl)c(C(=…	429.860	5.482	6	2	87.72	5	4	BCL-like/BAX,BAK	Active
25815	T00013	Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…	502.593	2.100	5	4	144.91	9	3	Integrins	Active
25816	T00014	Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…	502.593	2.100	5	4	144.91	9	3	Integrins	Inactive

```
[25817 rows x 11 columns]
```

[15]:
```
df1.to_csv("ppi_curation_search_1033.csv", index=False)
```

[ ]:
```

```