MOLECULAR PROPERTY DIAGNOSTIC SUITE - COMPOUND LIBRARY- (MPDS-CL)

(A disease independent web portal of MPDS)

http://mpds.neist.res.in:8086/

Index of MPDS Compound Library

1 Introduction

Compound library

The compound library represents a commendable endeavour aimed at achieving a methodical categorization of the vast chemical space. This is accomplished through the identification of fundamental molecular scaffolds, which play a pivotal role in conferring distinct properties and functions upon molecules. The realm of molecular classification encompasses various methodologies, each with its own merits. However, an approach solely reliant on the 'structure' of molecules faces certain challenges in terms of its presentation, primarily stemming from the intricate nature of molecular systems. The compound library module within the Molecular Property Diagnostic Suite (MPDS) web portal offers a distinctive approach to structural classification. This library stands out due to its innovative methodology, which is tailored to the specific needs of disease analysis.

A comprehensive collection of 149.16 million distinct molecules was meticulously curated from diverse chemical databases, exhibiting a remarkable diversity of chemical structures spanning across fifty-six distinct structural classes. The users are offered a streamlined single-window interface that empowers them to conduct a comprehensive exploration of the chemical space through a multi-layer search paradigm. This paradigm encompasses various search methodologies, including exact structure search, substructure search, and molecular property-based search. By seamlessly navigating through these search layers, users can effectively uncover and analyse chemical entities of interest. Each molecule in the dataset is assigned a distinct MPDS-ID, which serves as a representation of its class and various significant druglike characteristics. The utilisation of such a classification system holds immense value in diverse computer-aided drug design methodologies, as well as in efficiently traversing the vast chemical space. This is achieved through the application of sophisticated machine learning algorithms, which enable accurate predictions of structure-properties relationships.

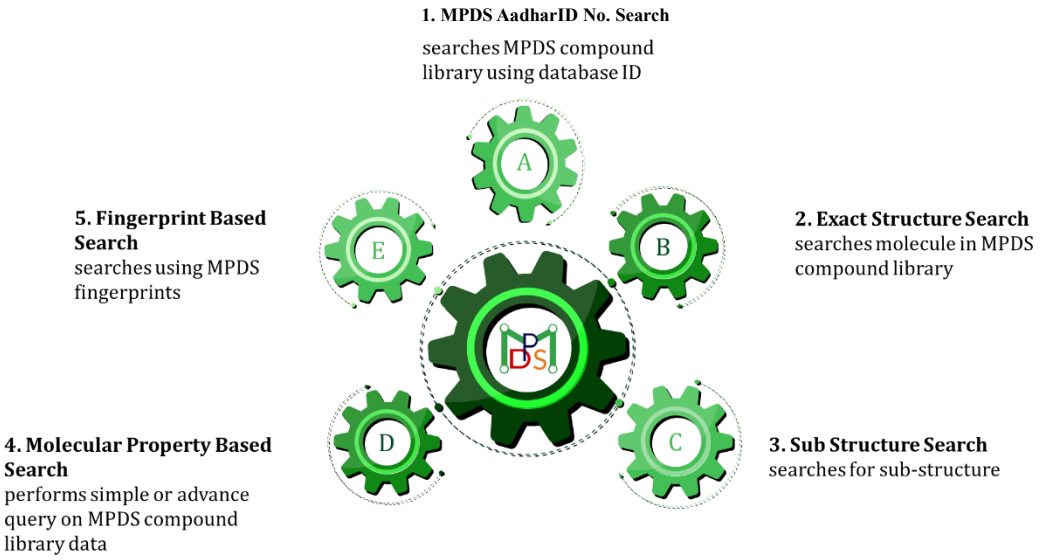


Figure 1. Various Search Options in Compound Library

Home Page of Compound Library

MPDS is based on Galaxy main server and it has three main panels:

1. Left Panel: Available Modules (Outlined with Green colour)
2. Middle Panel: View your data and results (Outlined with Blue colour)
3. Right Panel: Full record of your analysis history (Outlined by Pink colour)

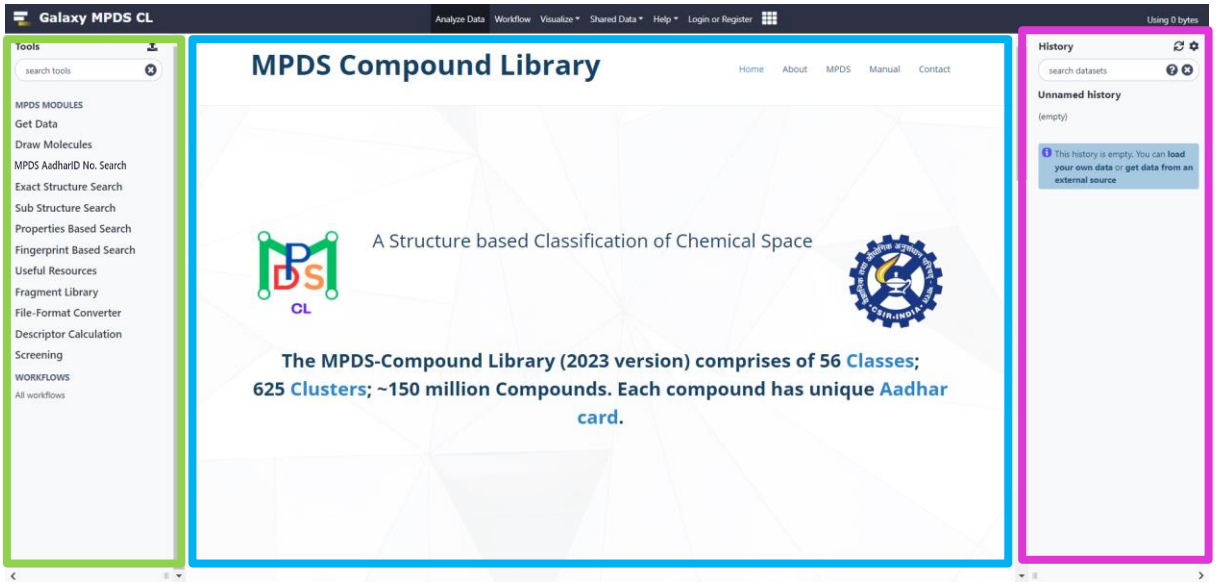


Figure 2. Home Page of MPDS-CL Web Portal.

2. Get data

Get Data module has two options viz:

(i) Get File of Drug molecule in particular format from DrugBank or from other source.

(ii) Upload File from your computer.

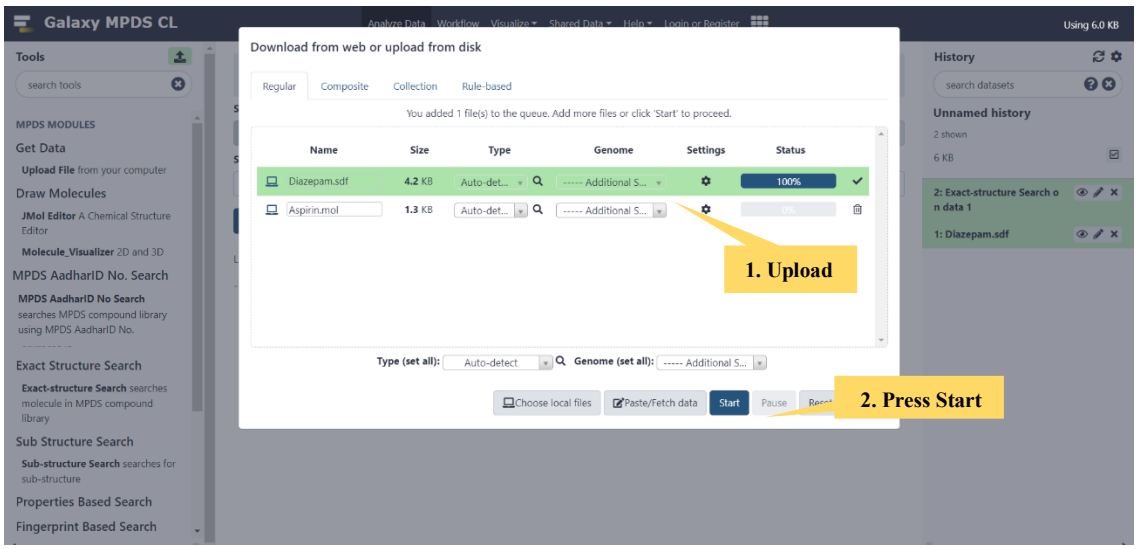


Figure 3. Showing process of uploading file.

3. Draw Molecules

Draw molecule module consists of two tools like:

(i) JMol Editor and

(ii) Molecular visualizer,

The user can draw their desired molecule and save in different file format (sdf, mol, mol2, and pdb). JMol Editor tool save the file in 2D format while Molecular Visualiser tools save the molecule in both 2D and 3D format.

3.1 JMol Editor A Chemical Structure Editor

In MPDS compound library, Jmol has been intergrated as it is an incredibly versatile and powerful molecular visualisation tool. It has emerged as an indispensable tool within the realms of computational chemistry and molecular biology. Its multifaceted capabilities have revolutionised the way researchers and scientists perceive and analyse molecular structures. By seamlessly integrating cutting-edge technology and sophisticated algorithms, Jmol has elevated the field, enabling a deeper understanding of complex molecular systems. Its impact on the Jmol, a cutting-edge open-source software, empowers researchers, students, and scientists alike to engage in a dynamic and immersive exploration of intricate three-dimensional models of molecules.

Step 1. After clicking the option **JMol Editor,** a small white window will appear**.** The small molecular structure can be drawn by clicking the symbol of rings, bonds and elements (C, H, N, O) etc.

Step 2. The generated structure can be exported to smile format by clicking the option “Export Smiles to Galaxy”. The structure may also be exported in mol format to galaxy with the given option.

Step 3. The exported smile can be downloaded by clicking the download option in the History panel represented in a small square box.

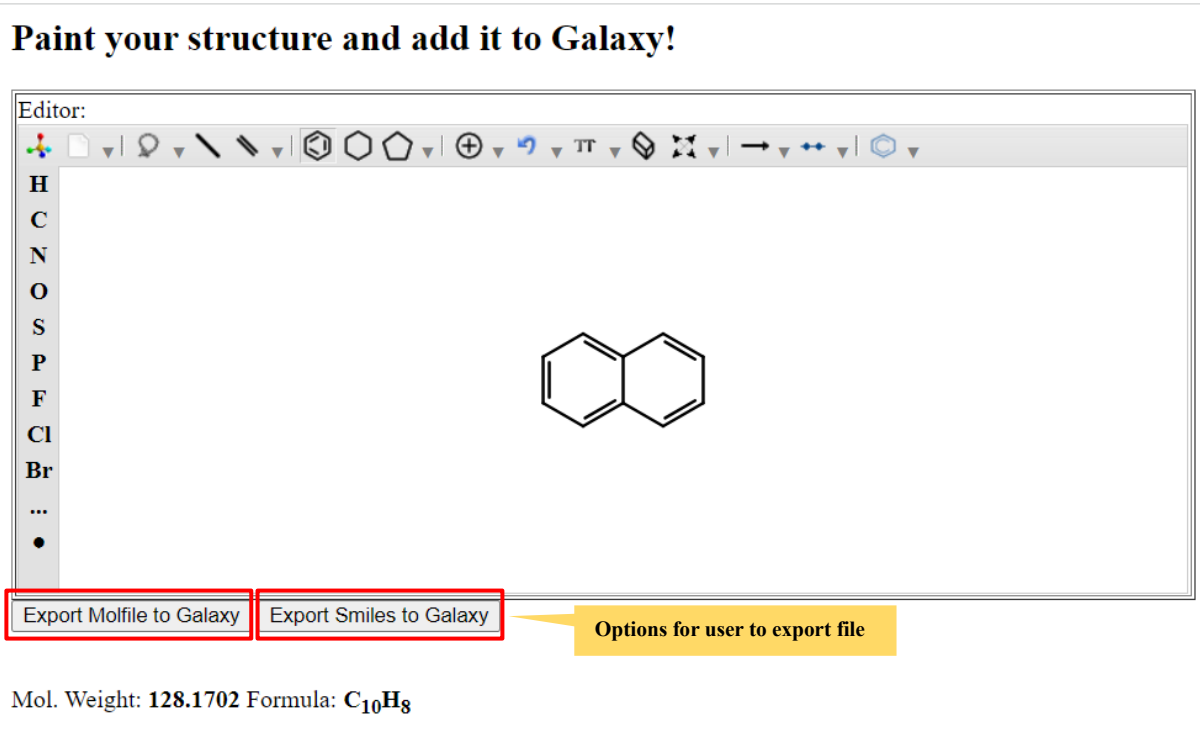


Figure 4. After clicking the "JMole Editor" option, user will get a window to paint the structures.

3.2 Molecules Visualizer 2D and 3D

Ketcher, an open-source web-based chemical structure editor, boasts a remarkable combination of high performance, exceptional portability, minimal resource consumption, and seamless integration capabilities within custom web applications. Ketcher, an innovative software solution, has been meticulously crafted to cater to the discerning needs of chemists, laboratory scientists, and technicians who are engaged in the intricate task of drawing structures and reactions.

In MPDS compound Library, to visualize the 2D and 3D structures of the molecule, Ketcher 6.2 can be used by the Users extensively. There are a variuos options that the user can use and work as per the requirement. With the aid of this integrated tool, the user can easily analyse the chemical structure and perform a number of editing and other associated actions required for the chemical structure of interest.

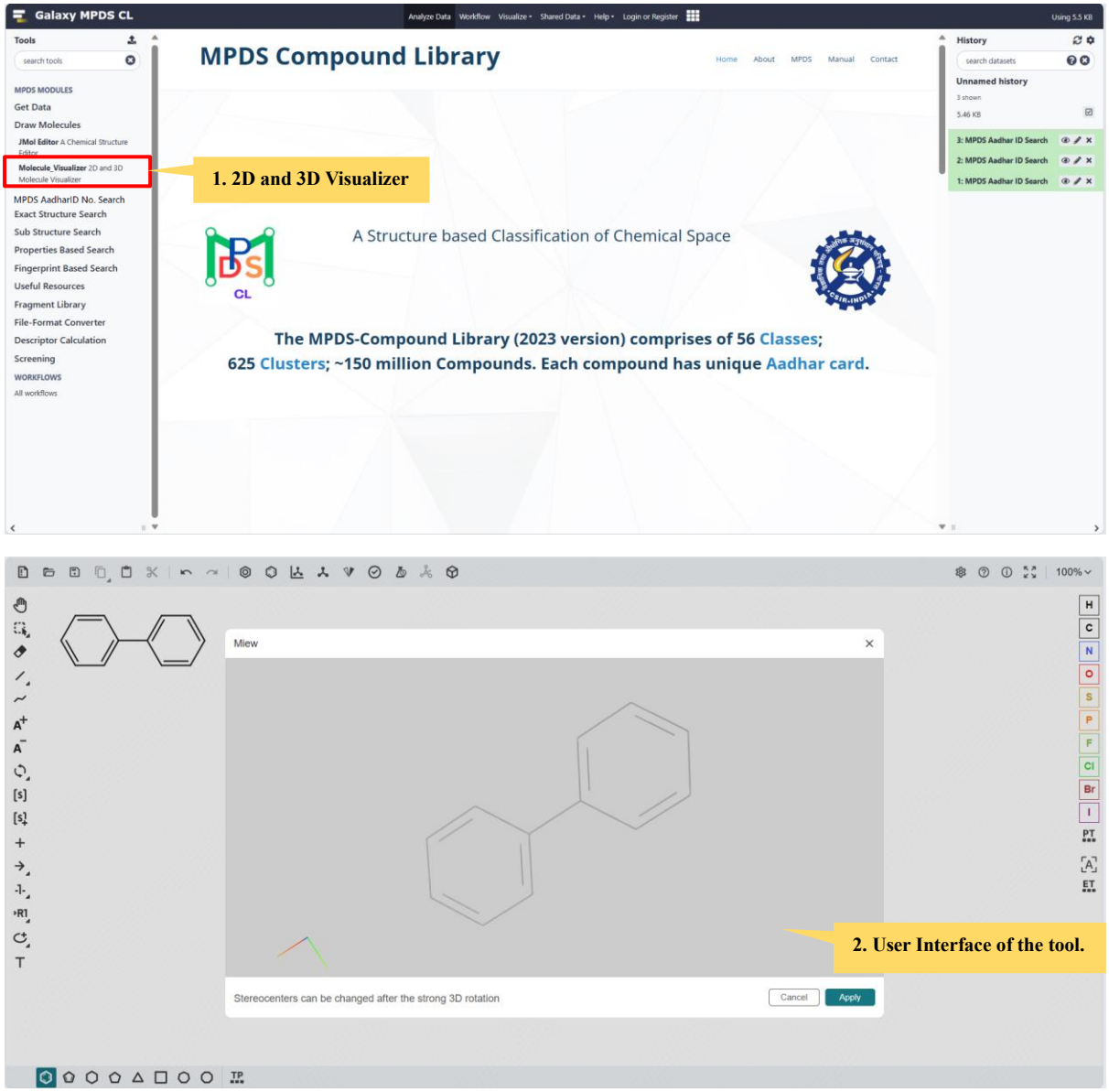


Figure 5. 2D and 3D Molecule Visualizer integrated in MPDS compound library.

4. MPDS ID Search

The Molecular Property Diagnostic Suite (MPDS) provides a cutting-edge advanced approach in the realm of molecular property analysis, drug discovery and forever growing chemical space. The innovative suite, MPDS, offers a comprehensive range of tools and methodologies for the accurate assessment and characterization of molecular properties. By leveraging state-of-the-art techniques, it enables researchers and scientists to gain valuable insights into the intricate nature of complex molecular structures. The MPDS AadharID search is a unique search option that is available for the users to have access to the vast chemical space. It helps the users to retrieve all the relevant information of a particular compound in one search.

In MPDS-CL, MPDS AadharID is a novel and unique idea that helps the user to get all the relevant information of the chemical compounds. The MPDS AadharID is a akin to the Adhaar card in India whereby all the relevant information is stored on the card.

4.1 MPDS AadharID Search searches MPDS compound library using database ID

MPDS Aadhar ID Search searches: The process of conducting an MPDS AadharID Search entails looking for specific information or records within a database by utilizing unique identifiers that are linked to those particular records. Numerous databases employ unique identifiers to facilitate efficient retrieval and referencing of data.

Steps 1. The user as per the choice may enter the MPDS ID.

Step 2. Click on execute to generate the MPDS AadharID No. of the molecule.

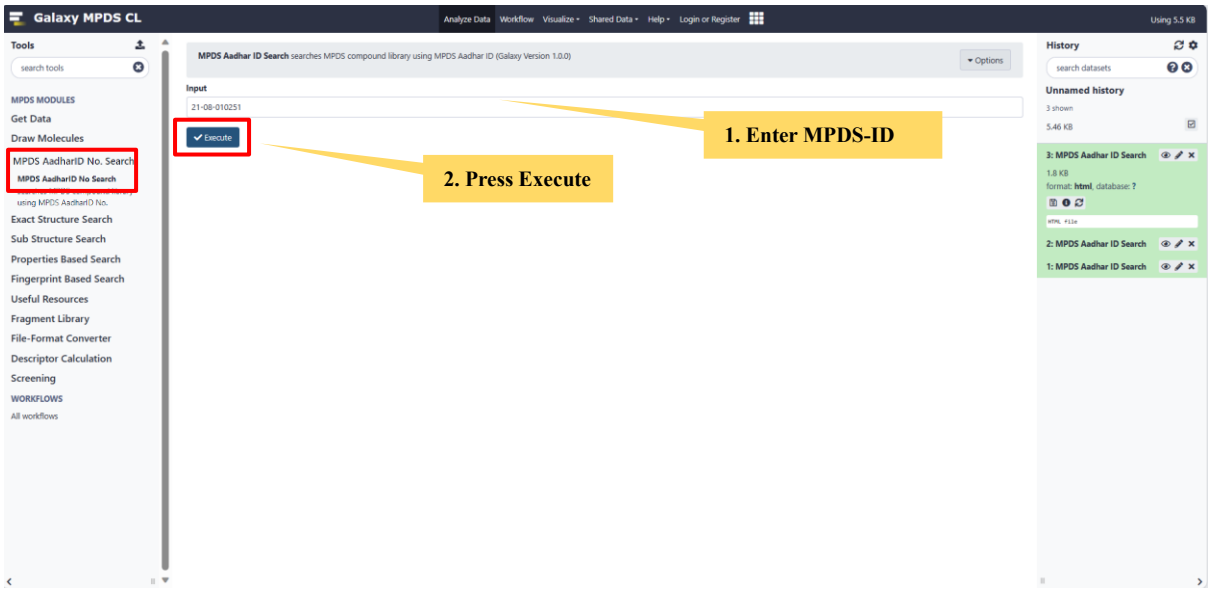


Figure 6. The searched molecule will appear along with the MPDS AadharID.

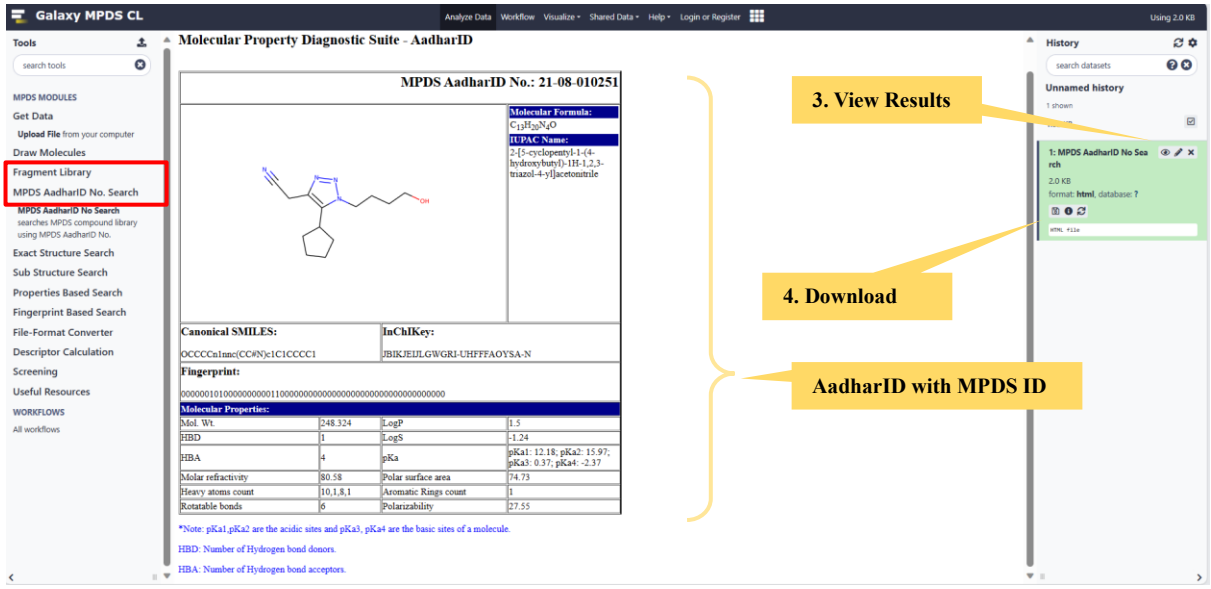


Figure 7. Results generated by MPDS AadharID search.

5. Exact-structure Search

In MPDS-Compound Library (MPD-CL), the process of exact structure search entails the discernment of molecules exhibiting highly specific attributes, including but not limited to a distinct molecular configuration, physicochemical properties, or biological activity. The implementation of targeted exploration is of utmost importance in order to maximise the optimization of the selection process for potential drug candidates or novel materials possessing desired traits. Through the strategic utilisation of precise search methodologies within the expansive realm of chemical space, diligent researchers possess the capability to expedite the intricate process of pinpointing highly auspicious compounds. This remarkable endeavour ultimately paves the way for unprecedented breakthroughs across a myriad scientific domain, propelling the boundaries of knowledge and innovation to unprecedented heights.

5.1 Exact-structure Search searches molecule in MPDS compound library

An exact-structure search is a search method that is frequently used in chemical databases to locate compounds that possess an identical or precisely comparable chemical structure to a provided query structure.

Step 1. The user may select the file for an exact structure search in compound library.

Step 2. The users may select the input file in SDF, MOL, or SMILES format for exact structure search in Compound Library. Here, Alprazolam which is used to treat panic disorders and generalized anxiety has been uploaded in sdf format.

Step 3. Press Execute.

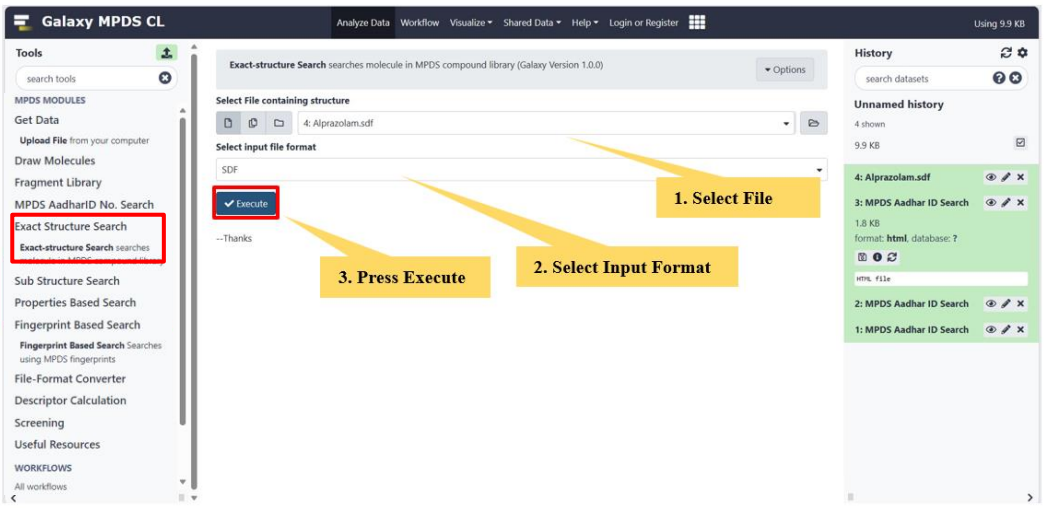


Figure 8. Steps involved in exact structure search.

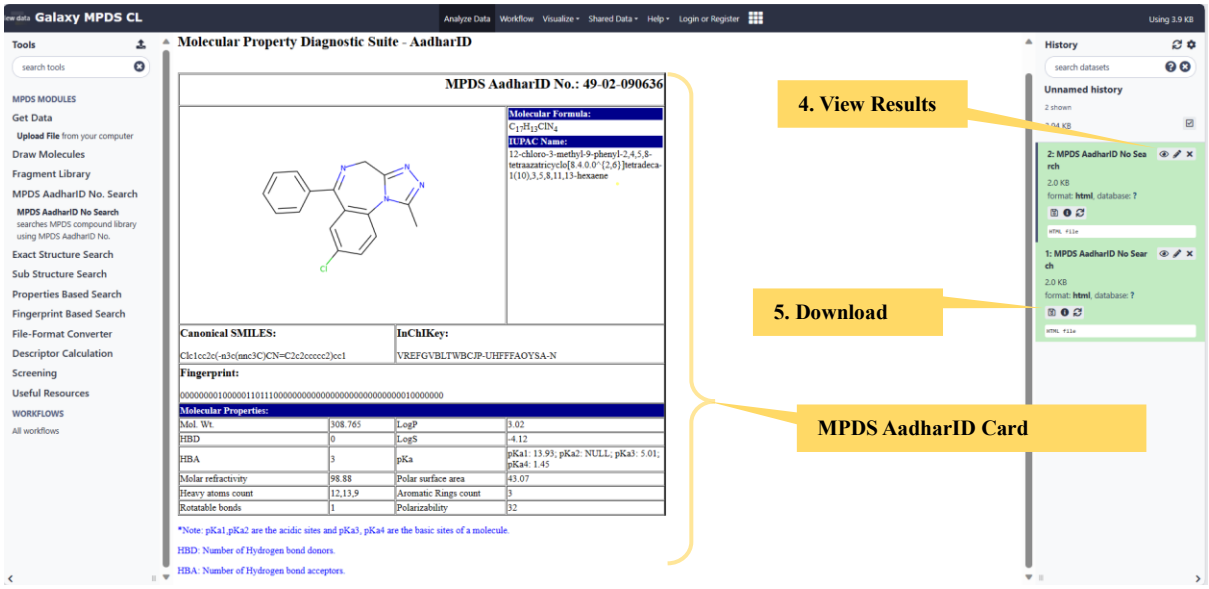


Figure 9. Results generated by the exact structure search.

6. Sub-structure Search

In MPDS-CL, the sub-structure search stands as an indispensable and formidable search option within the realm of chemical information retrieval and analysis on the basis of structural classification. The process of sub-structure search entails the meticulous exploration of a vast repository of chemical compounds in order to identify and locate specific patterns or structures. The utilisation of this approach holds immense significance in the realm of drug discovery and various other domains wherein the identification of molecules possessing specific functional groups or arrangements assumes paramount importance. Through the utilization of sophisticated algorithms and the implementation of graph-based representations, researchers are able to effectively navigate extensive chemical databases and retrieve compounds that possess a desired sub-structure.

The technique of sub-structure search empowers chemists to efficiently discern potential lead compounds by identifying specific key features shared with known active molecules. This expedited approach significantly accelerates the drug and material design process, enabling the discovery of novel compounds with enhanced therapeutic or functional properties. Furthermore, this cutting-edge search serves as a catalyst for the comprehensive examination of chemical data, empowering researchers to discern invaluable insights and establish intricate connections between various structures and properties.

6.1 Sub-structure Search searches for sub-structure

The approach of substructure search is commonly employed in the fields of chemoinformatics and computational chemistry. It enables the identification of molecules or compounds that possess a particular substructure or chemical motif of interest. This tool enables researchers to effectively identify compounds possessing a specific structural feature within a comprehensive database or compound set.

Step 1. The user may select the file for sub- structure search in compound library.

Step 2. The users may select the input file in SDF, MOL, or SMILES format for exact structure search in Compound Library.

Step 3. Press Execute.

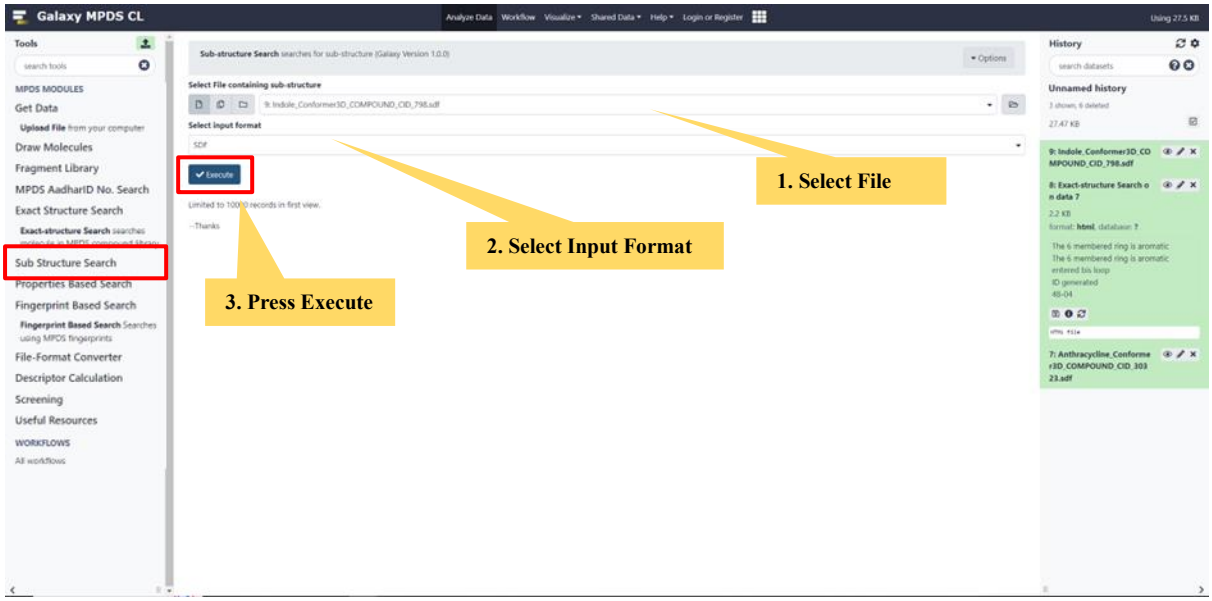
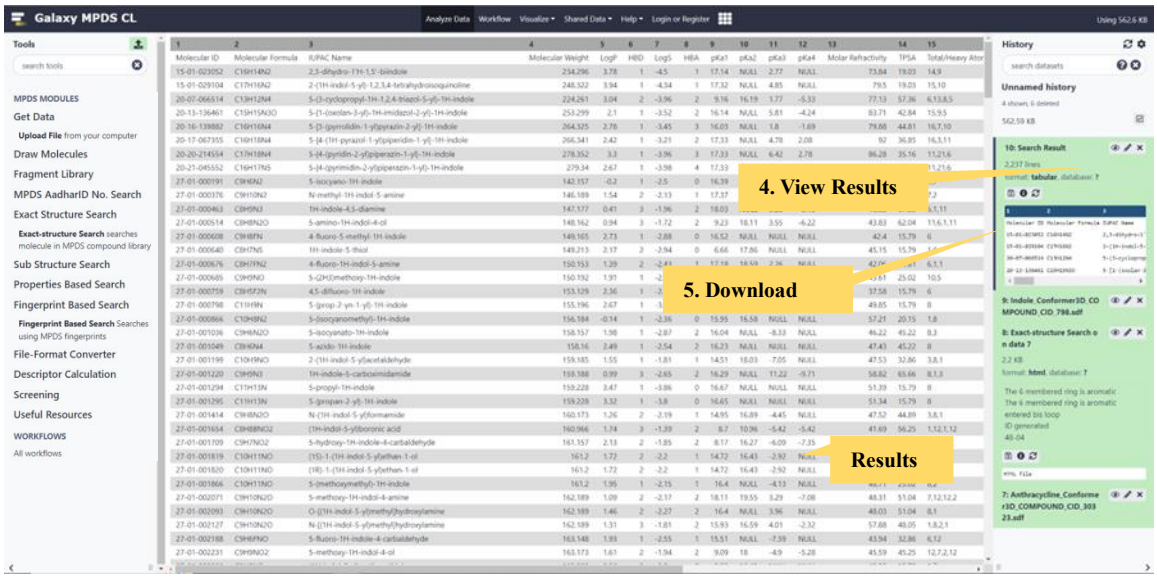


Figure 10. Steps involved in the sub-structure search in the compound library.



7. Property Based Search

The utilisation of property-based search techniques in MPDS-CL plays a pivotal role in contemporary chemical research and the field of drug discovery. The pursuit of chemical compounds possessing precise physicochemical properties or biological activities within an extensive database is the essence of property-based search. The utilisation of this methodology proves to be immensely advantageous in scenarios where the precise chemical structure or substructure of the target molecule remains elusive, yet certain distinctive attributes hold paramount significance. Through the adept utilisation of cutting-edge computational methodologies and algorithms, researchers are empowered to deftly traverse the vast expanse of chemical space, defying its inherent complexity, and deftly discern compounds that manifest the coveted attributes, encompassing solubility, potency, toxicity, and other pivotal parameters.

The implementation of property-based search methodologies assumes a paramount significance in the process of lead optimization within the realm of drug discovery. In this intricate pursuit, researchers ardently endeavour to identify and select molecules that exhibit an optimal equilibrium between efficacy and safety. Furthermore, within the realm of materials science, this cutting-edge technique plays a pivotal role in the discernment of materials possessing meticulously customized properties, thereby catering to a wide array of applications. The efficacy of property-based search resides in its capacity to expedite the identification of promising candidates amidst the extensive array of potential compounds, thus resulting in substantial time and resource savings.

7.1 Molecular Property Based Search Perform: simple or advanced queries on the compound library data

Molecular Property Based Search is a procedure for finding compounds or molecules having a particular chemical or physical qualities. It enables researchers to locate compounds having the appropriate physicochemical qualities such as molecular formula, molecular weight, LogP, molecular solubility, hydrogen bond donor/acceptor toxicity, bioactivity, ring counts, etc. can be explored by the users.

Step 1. The user may select the molecular property name.

Step 2. The users may select the operator name such as equal to, less than, or greater than.

Step 3. The users may provide the value for the selected property.

Step 4. Press Execute.

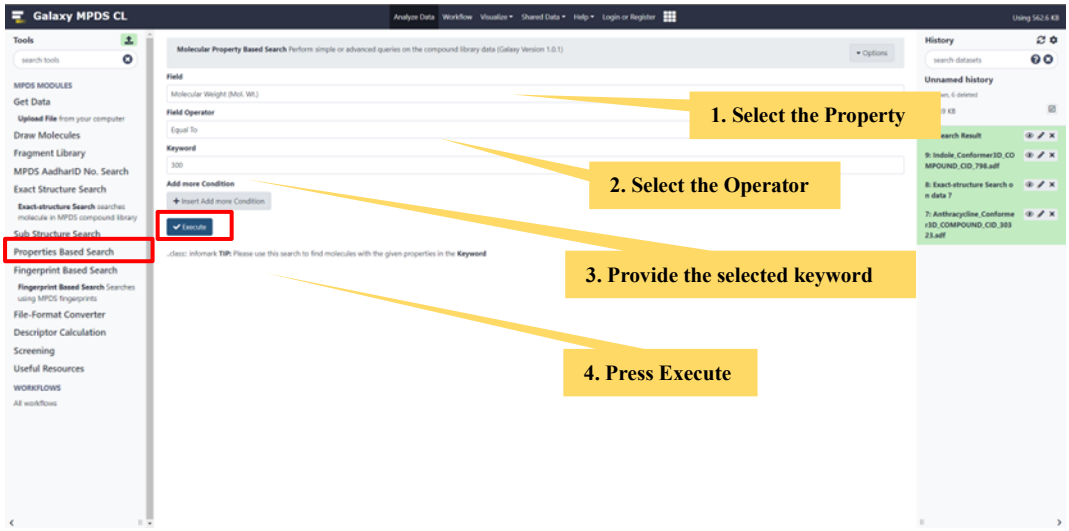


Figure 12. Steps involved to search a molecule using molecular property-based search.

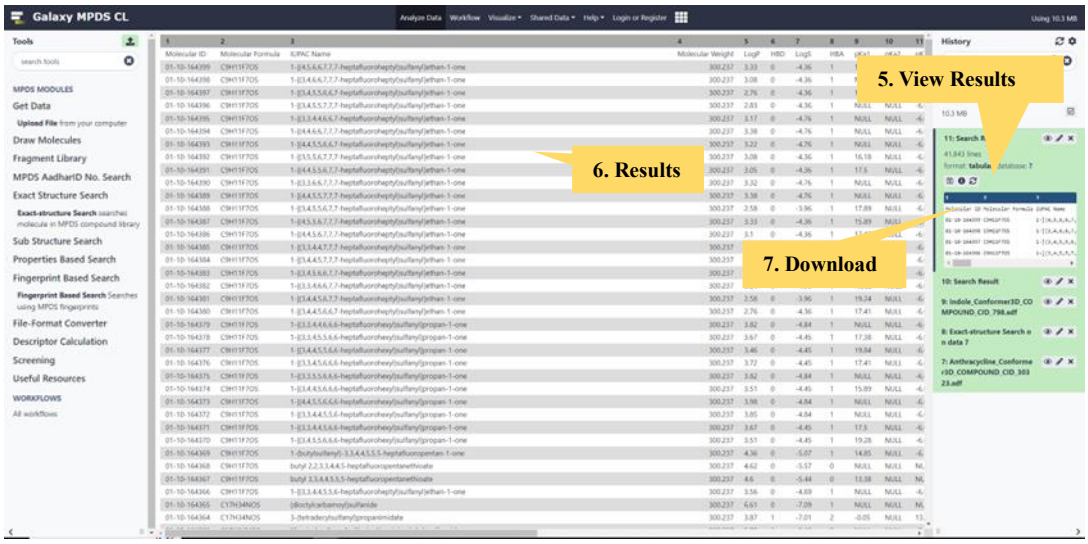


Figure 13. Results generated by the molecular property-based search.

8. Fingerprint Based Search

In MPDS-CL, the fingerprint-based search methodologies represent the production of distinctive and concise representations of chemical structures, commonly referred to as fingerprints, that effectively encapsulate crucial molecular characteristics. The acquired fingerprints are subsequently leveraged to expedite comparisons and discern compounds exhibiting analogous structural attributes or properties.

Fingerprint-based search methodologies have emerged as indispensable tools in the domains of drug discovery and materials science, as they enable the identification of molecules possessing precise functionalities or properties. The utilisation of this technology empowers researchers to effectively prioritize compounds with high potential for further investigation, thereby expediting the lead identification process to a significant extent. In addition, the utilisation of fingerprint-based search methodologies serves as a catalyst for the comprehensive investigation of the vast expanse of chemical space. This approach not only enables the elucidation of intricate connections between molecular structures and their corresponding functionalities but also plays a pivotal role in the strategic development of innovative pharmaceuticals and materials possessing specific and desirable characteristics.

8.1 Fingerprint Based Search using MPDS fingerprints

Chemoinformatics and Computational Chemistry employ the fingerprint-based search approach to discover chemicals or molecules depending on their structure or molecular similarities. When looking for compounds with similar structures in huge databases, or when looking for potential analogues or related molecules, fingerprint-based search is extremely useful. Fingerprint similarity search methods are especially useful in virtual screening if only a few unrelated ligands are known for a given target.

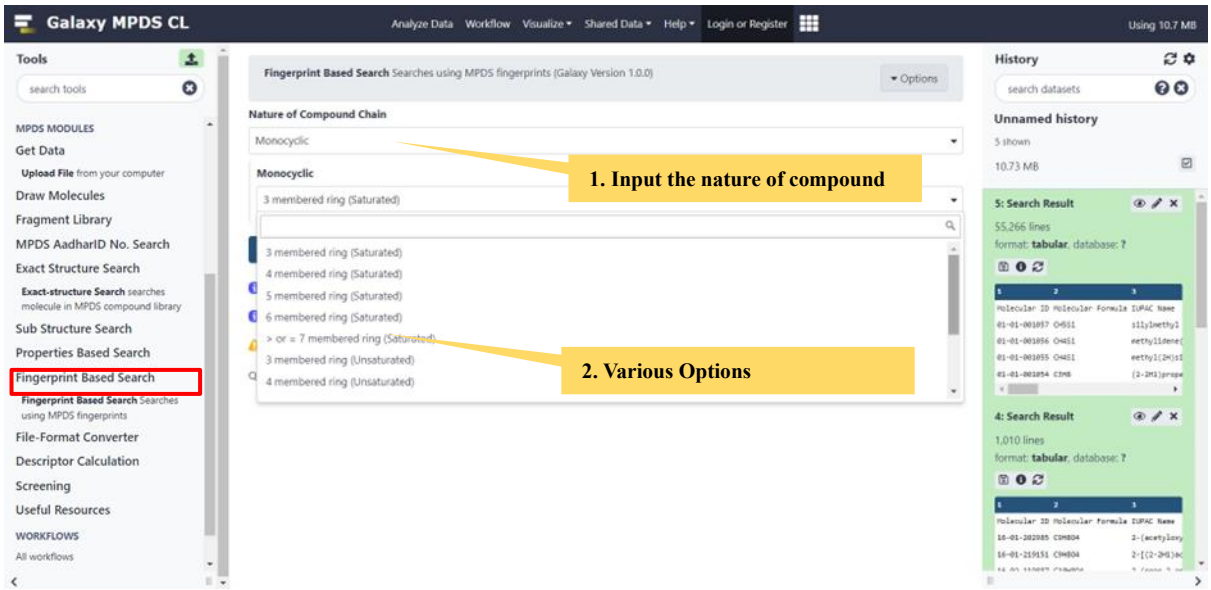


Figure 14. Various options for fingerprint based search for Monocyclic Compounds.

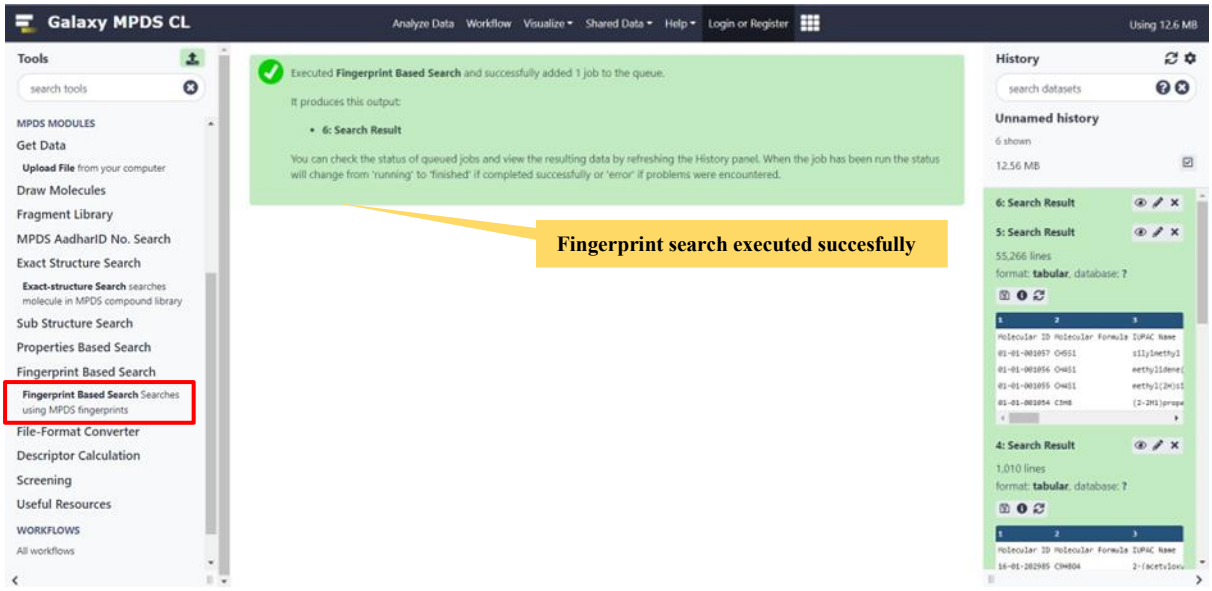


Figure 15. Execution of the fingerprint based search.

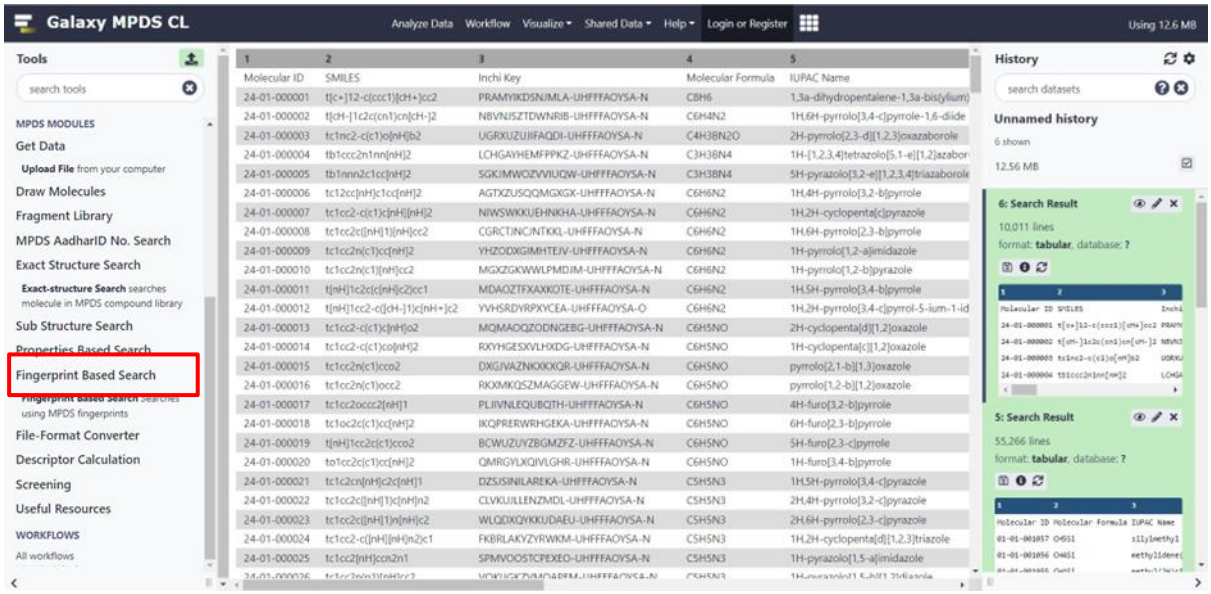


Figure 16. Result of fingerprint based search.

9. Fragment Library

In MPDS-CL, the fragment library stands as an invaluable asset within the realm of drug discovery and medicinal chemistry, encompassing a vast and all-encompassing assortment of diminutive, different molecular weight chemical fragments. Frequently, these fragments serve as diminutive substructures that hold the potential to embody drug-like properties, thereby serving as fundamental constituents for the purpose of designing and refining innovative drug candidates. The library has been meticulously curated to encompass a diverse array of chemical functionalities, shapes, and properties, thereby guaranteeing its suitability for comprehensive exploration across a multitude of target classes. Fragment-based approaches are a prominent strategy in the realm of drug discovery. These approaches entail the systematic screening of libraries containing small molecular fragments against specific biological targets. The primary objective is to identify initial hits that exhibit promising interactions with the target of interest. Subsequently, these hits can be further expanded upon and refined to ultimately yield lead compounds that possess both high potency and selectivity. In recent years, the utilisation of fragment libraries has garnered significant attention and recognition within the scientific community. This is primarily attributed to their remarkable capacity to efficiently sample chemical space, thereby empowering researchers to delve into a wider spectrum of chemical diversity. Consequently, this enhanced exploration facilitates a higher probability of uncovering exceptional lead compounds of superior quality.

The Fragment Library represents as a vital component within the realm of drug discovery and medicinal chemistry. The aforementioned is a collection of diminutive, chemically heterogeneous molecular fragments that function as fundamental units for the purpose of conceiving and advancing novel drug candidates.

9.1 Fragmenter - splits a molecule into fragments

The fragmenter as the name suggests performs the fragmentation of the input file uploaded by the user and further helps in fragment-based search and analysis. Fragmenter helps the user to split the molecue into fragment. So, per the choice, the user may use the different splitting rules viz. Recap Rule, Rotatable bonds and Rings.

Step 1. Browse the location and upload the sdf file of small molecule from your computer.

Step 2. Choose the splitting rule and various other options that are available.

Step 3. Select Output format.

Step 4. Press Execute.

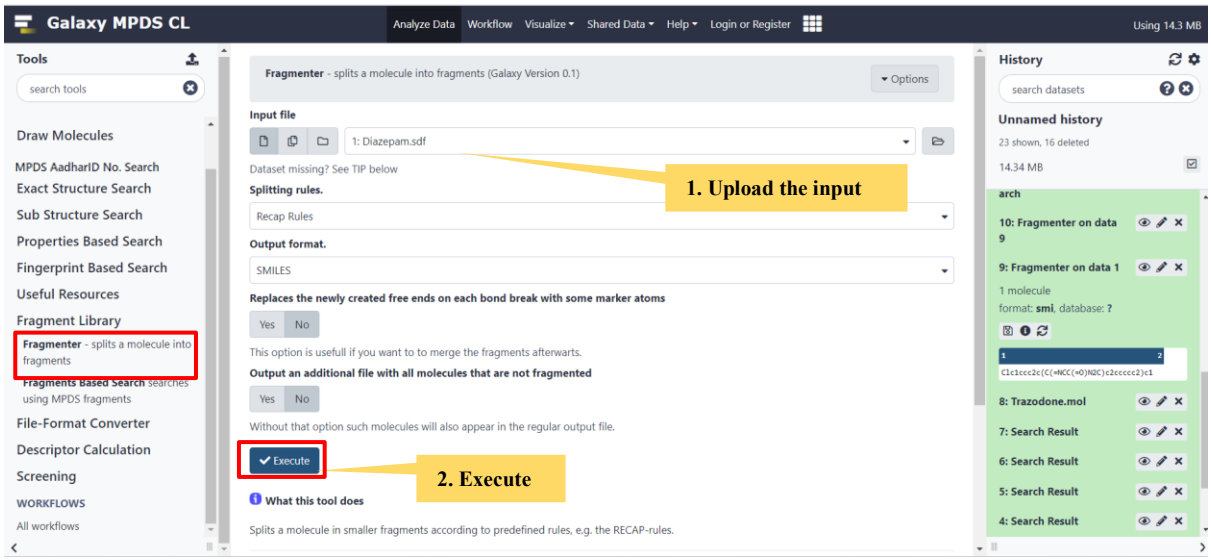


Figure 17. Showing the various options for fragmenter with splitting rules.

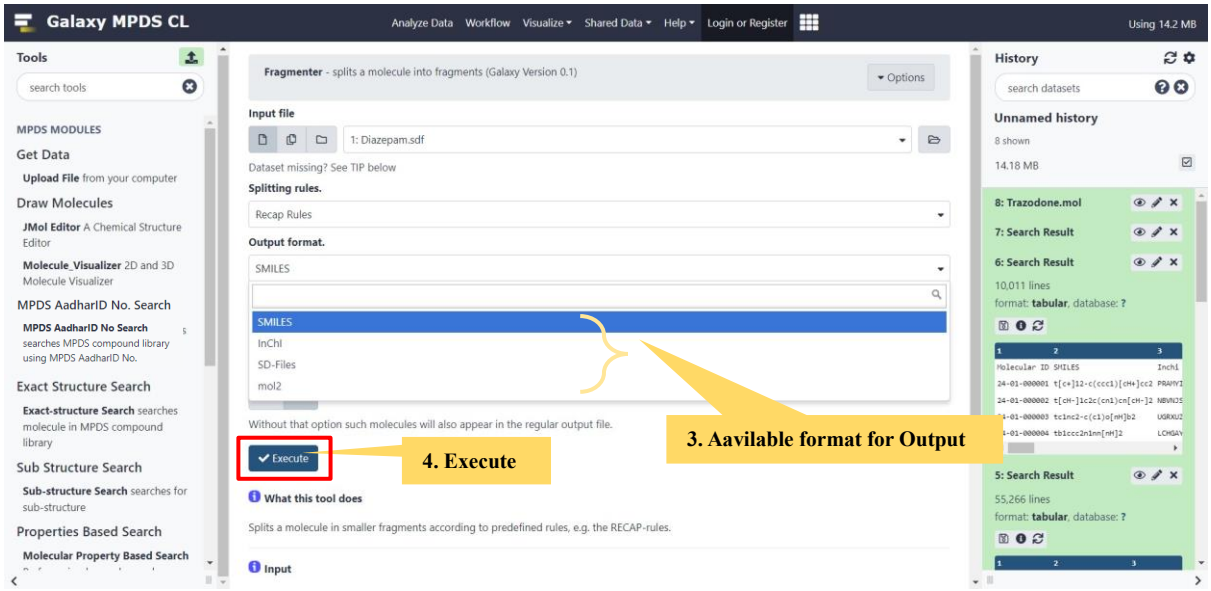


Figure 18. Showing the various options for several output format with different splitting rules.

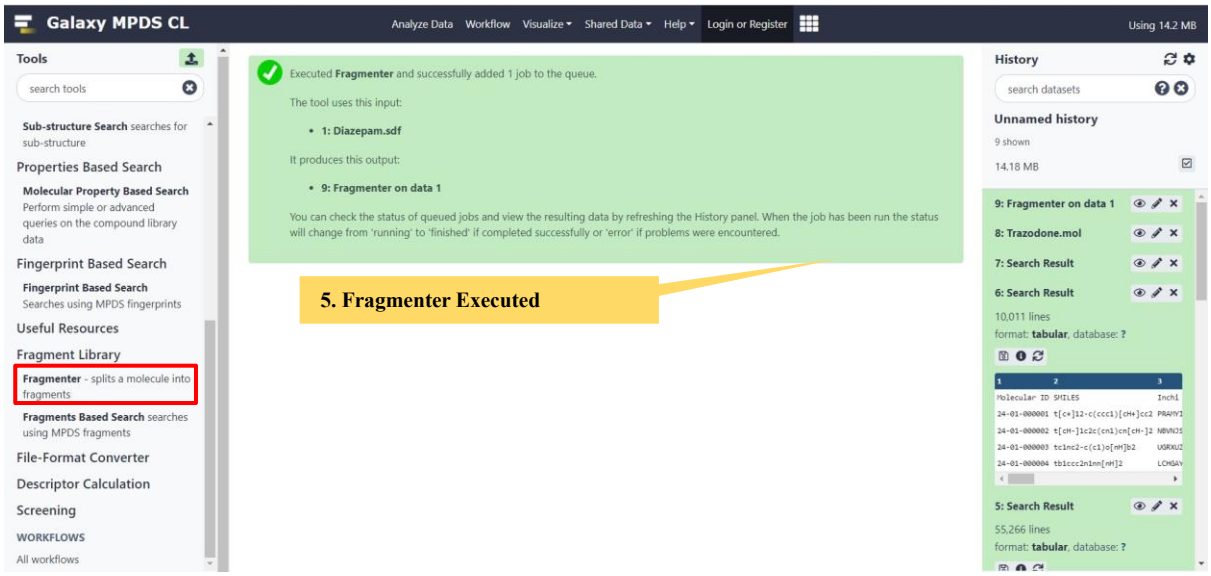


Figure 19. Fragmenter executed successfully and the results shown in history pane.

.

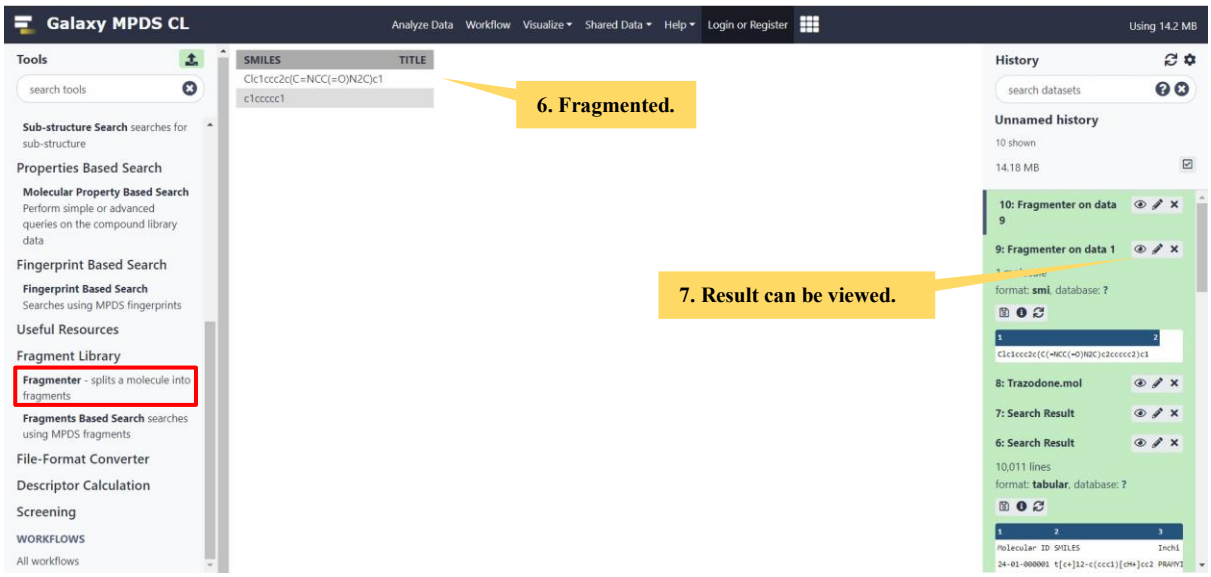


Figure 20. The molecule is fragmented and the results can be saved.

9.2 Fragments Based Search searches using MPDS fragments

The fragment-based searching is a computational technique employed in the fields of chemoinformatics and drug discovery. Its purpose is to locate and explore fragments or small molecular substructures within a vast chemical database. The process entails the fragmentation of larger molecules into smaller components, followed by a systematic search for these components within a database. This search aims to identify compounds which possess identical or comparable substructures.

In the field of computer science, a prominent technique known as Fragment Based Search (FBS) has emerged as a powerful tool for various applications. FBS involves the decomposition of a given problem into smaller fragments, which are then individually analysed

Fragment-based searching is a highly effective computational technique that finds extensive application in the domains of chemoinformatics and drug discovery. The primary objective of this system is to effectively identify and investigate minute molecular substructures or fragments within an extensive repository of chemical compounds. The procedure involves the disintegration of larger molecules into smaller constituents, subsequently pursued by a methodical exploration for said constituents within a comprehensive database. The objective of this search is to discern compounds that exhibit congruent or analogous substructures.

Step 1. The user may choose the nature of compound chain.

Step 2. The user may choose the type of fragments depending upon the choice be it momocyclic, bicyclic, tricyclic, tetracyclic, polycyclic.

Step 3. Press Execute.

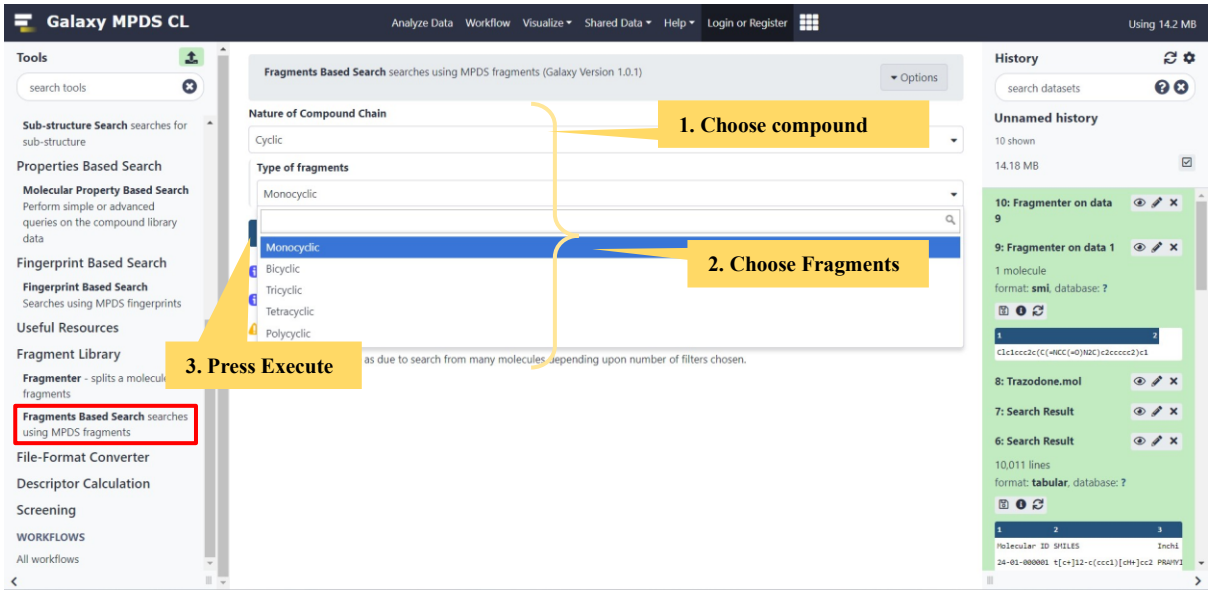


Figure 21. Showing the various search options in fingerprint-based search

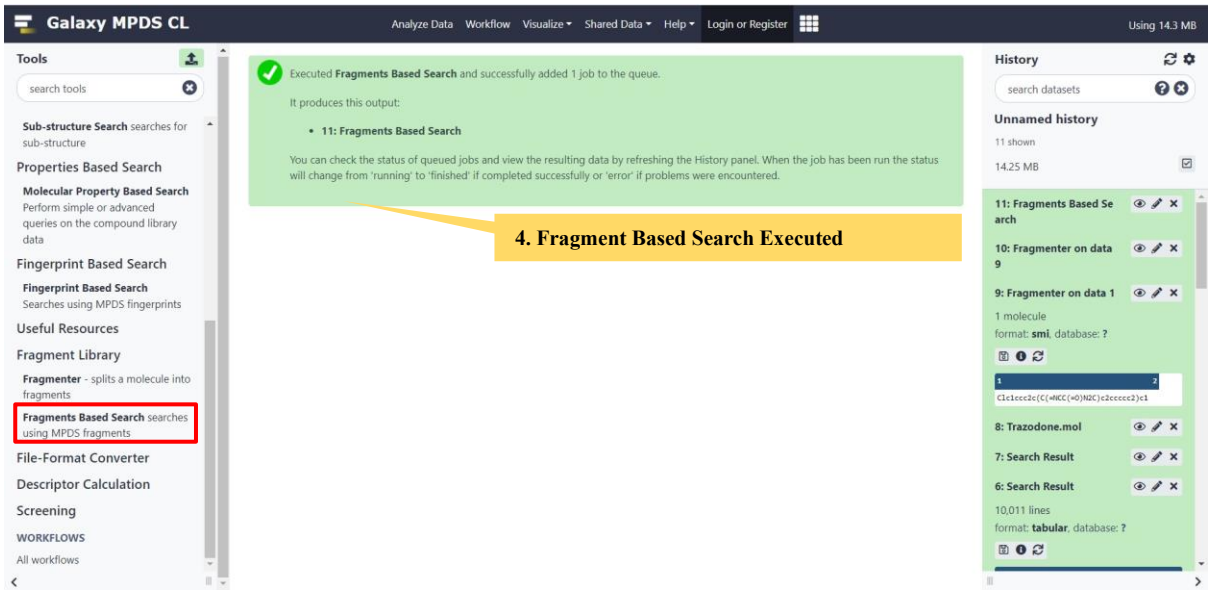


Figure 22. Fragments based search executed successfully.

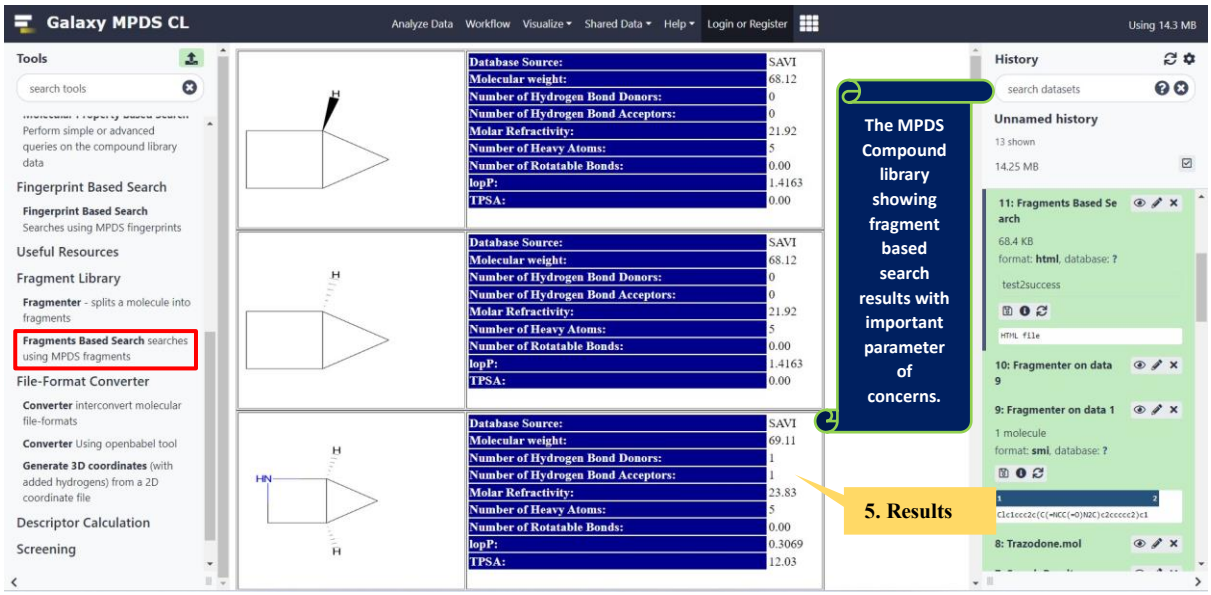


Figure 23. Results for fragment-based search.

10. File Format Converter

The utilisation of a file format converter has become an indispensable asset within a multitude of industries and disciplines, facilitating the seamless transformation of data or information from one file format to another. This highly adaptable software solution empowers users to seamlessly navigate the complexities of application, operating system, and device compatibility, facilitating effortless data exchange and utilisation. File format converters possess the remarkable ability to proficiently manage an extensive spectrum of file formats, encompassing diverse categories such as documents, images, videos, audio files, and an assortment of other digital content. They serve as a catalyst for the metamorphosis of data into formats that are optimally tailored to meet specific requirements, thereby guaranteeing the preservation of data integrity and safeguarding crucial attributes throughout the entire conversion process. In the realm of data manipulation, file format converters emerge as indispensable tools that significantly bolster productivity and efficiency. By streamlining intricate data transformations, these converters alleviate the need for laborious manual interventions, thereby saving valuable time and resources. The three tools incorporated in the MPDS-CL for employing trustworthy and fortified file format conversion tools are of paramount importance. These tools serve as guardians of data integrity, privacy, and quality, while concurrently facilitating effortless data exchange and collaboration across a multitude of platforms and systems.

10.1 Converter interconvert molecular file-formats

In MPDS compound library, the file format conversion can be done using three tools. For converting the particular file from one format to another, the user needs to first upload the file. The file-format converter combines multiple Open Babel command prompt converters, culminating in a unified and user-friendly tool. This cutting-edge software application facilitates the seamless interconversion of diverse small molecule and molecular modelling data files. The specification of the output format can be accompanied by a multitude of parameters. Various parameters are universally applicable across all tools, such as protonation state and pH. Conversely, certain parameters are exclusive to a particular output format, for instance, the exclusion of isotopes for conversion to canSMI.

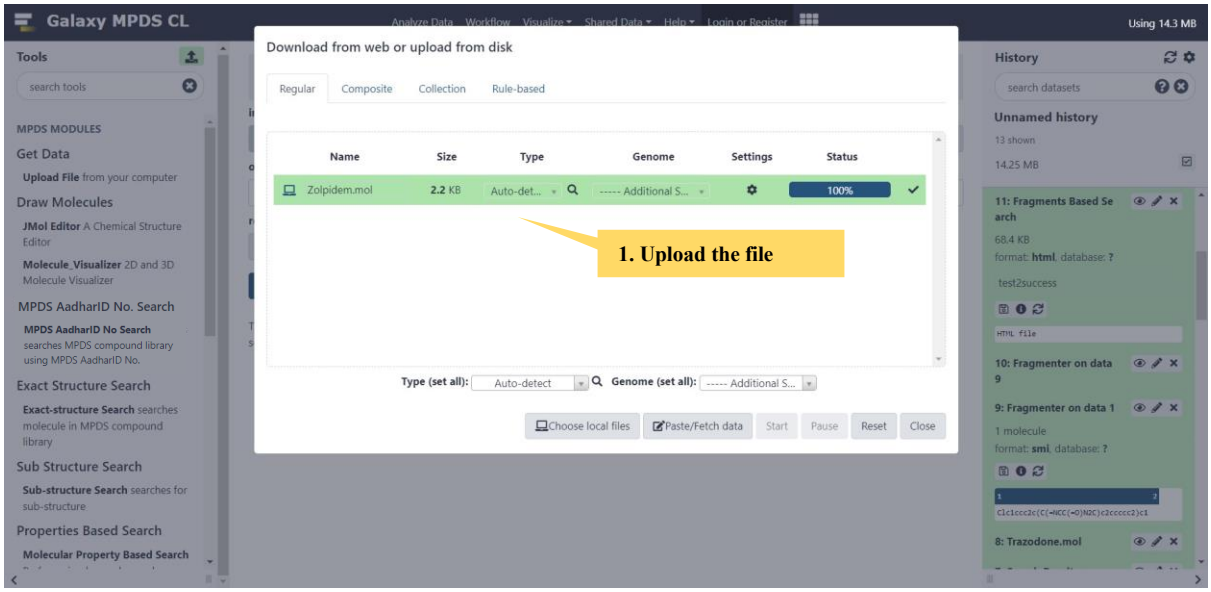


Figure24. Showing the drug Zolpidem in mol format to be converted.

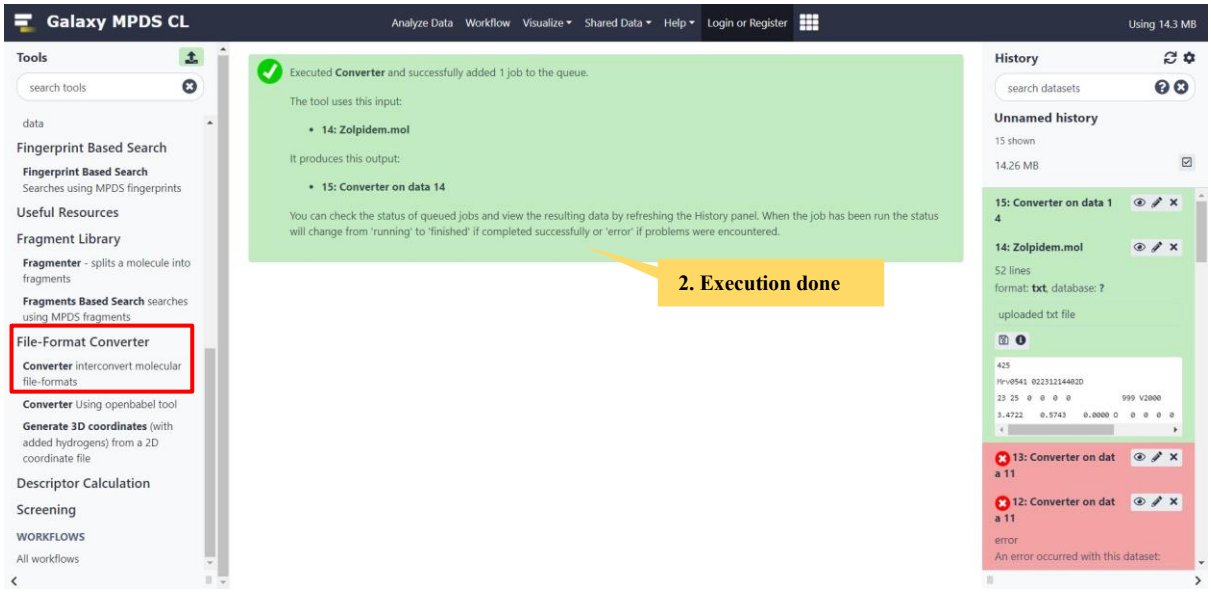


Figure 25. Execution done with green background confirming the conversion of file done successfully.

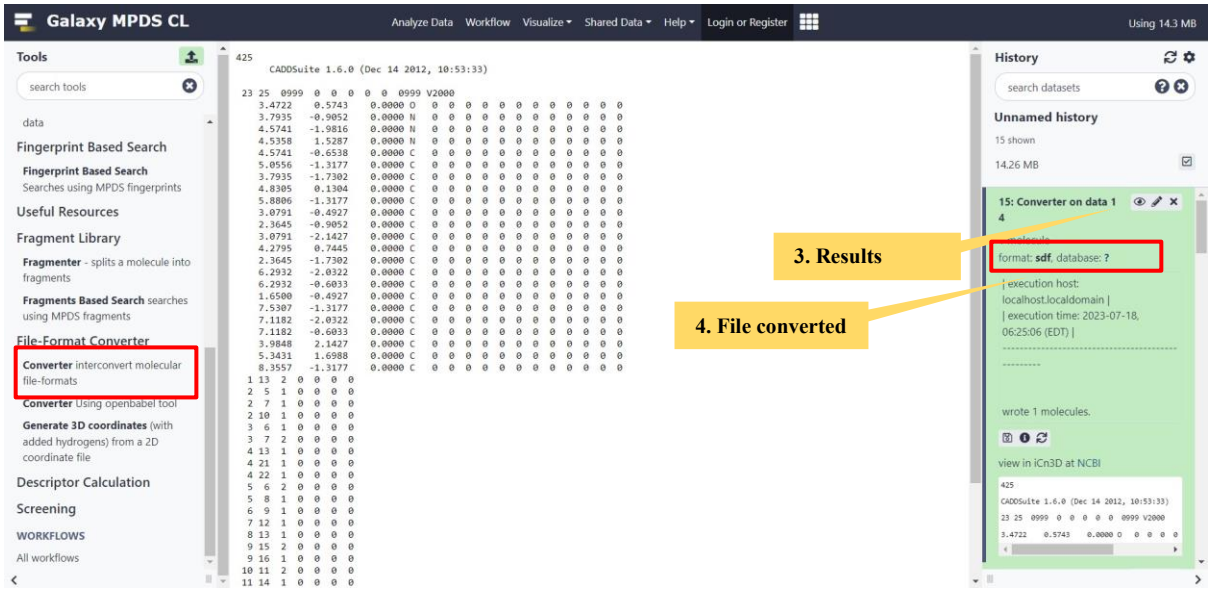


Figure 26. Showing the file is converted from mol to sdf format

10.2 Converter Using openbabel tool

In MPDS compound library, open babel has been integrated for file conversion. Open Babel which is a cutting-edge open-source chemical toolbox, stands as an indispensable resource for molecular structure manipulation, boasting its prowess as a formidable file converter. Open Babel, with its remarkable adaptability and extensive format compatibility, emerges as an indispensable asset within the realms of cheminformatics and computational chemistry. Whether the task at hand involves the transformation of a molecular representation from a two-dimensional to a three-dimensional paradigm or the seamless translation of a particular file format into another, Open Babel's prowess shines through, empowering researchers and practitioners alike. Hence, this tool plays an important tool in MPDS compound library.

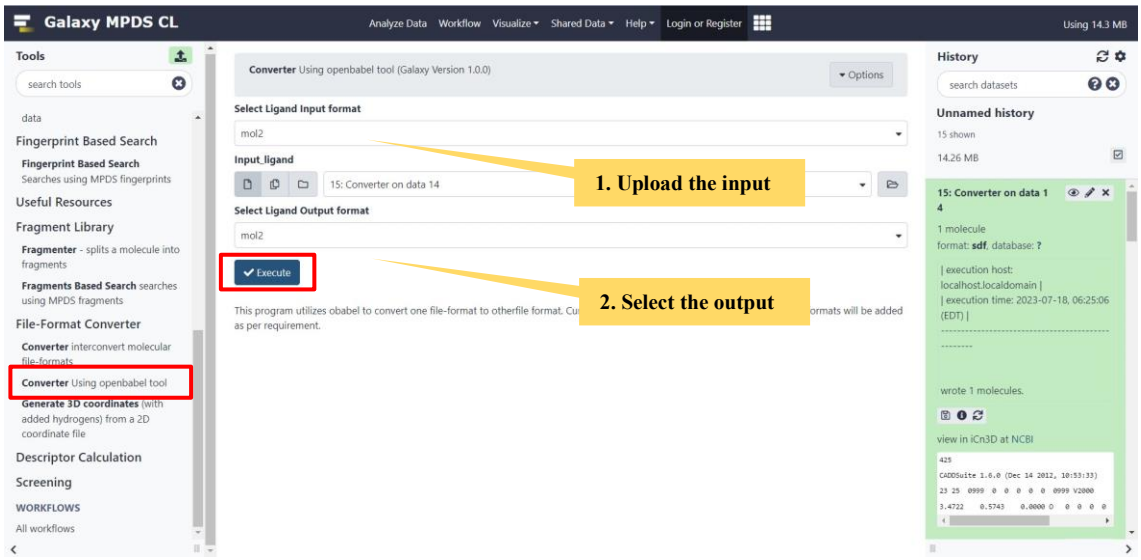


Figure 27. Showing options to upload the input file and to download.

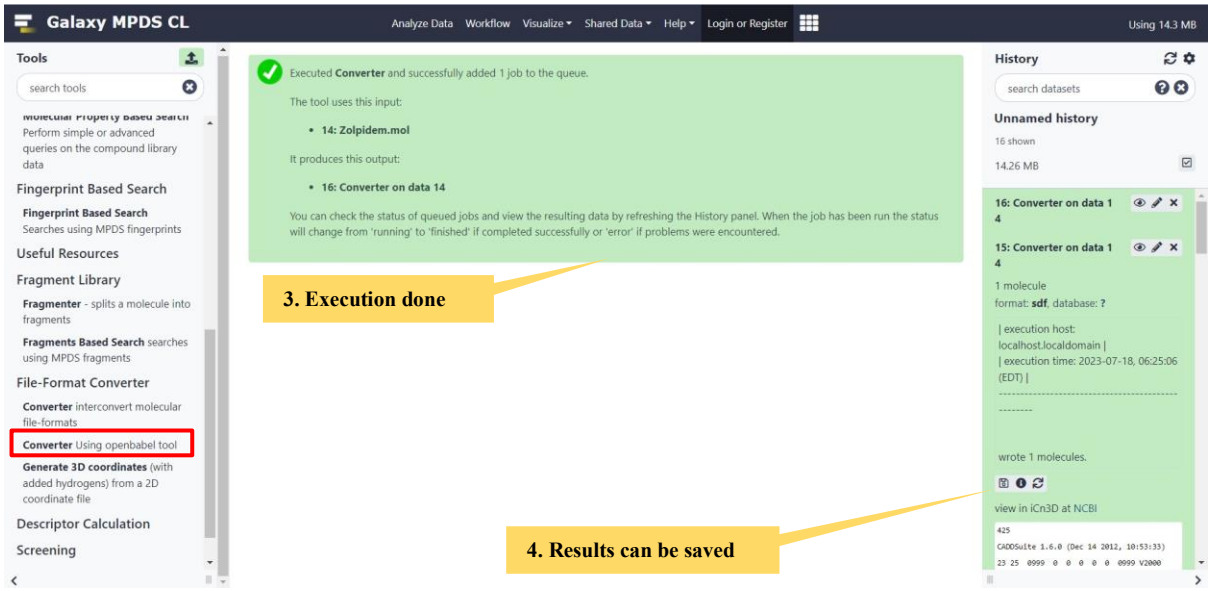


Figure 28. Showing the various steps for Converter Using openbabel tool.

10.3 Generate 3D coordinates (with added hydrogens) from a 2D coordinate file

The generation of three-dimensional (3D) coordinates for molecules is a fundamental procedure in the field of computational chemistry and molecular modelling. The task at hand encompasses the transformation of two-dimensional depictions of chemical structures into their corresponding three-dimensional conformations. There exist multiple methodologies to accomplish this objective, among which energy minimization stands out as a prevalent approach.

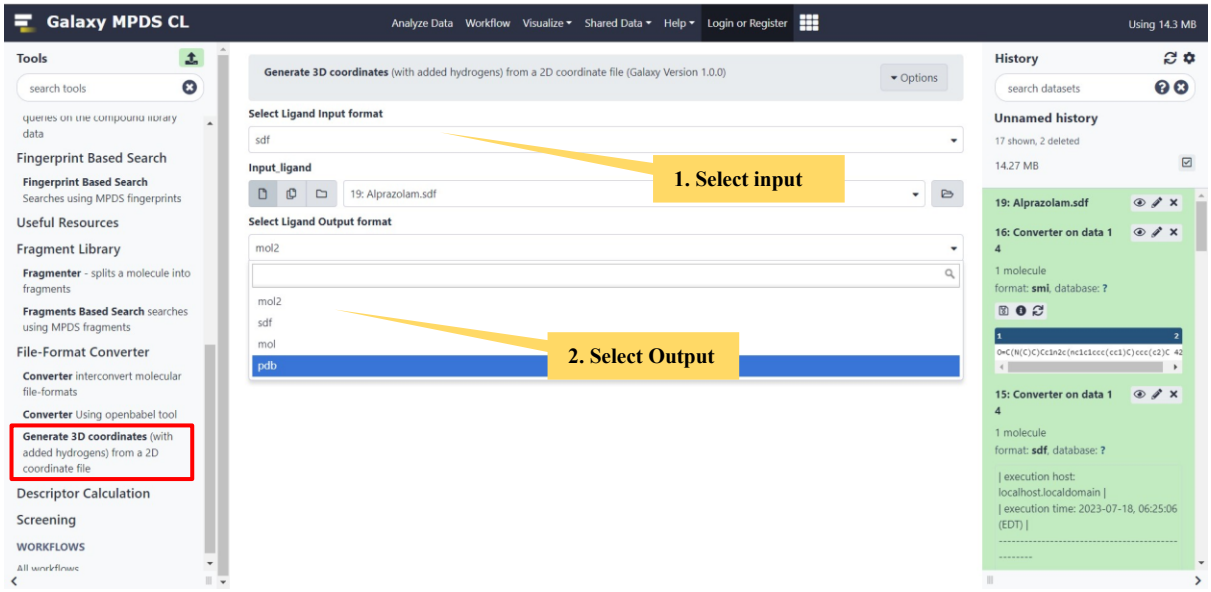


Figure 29. Showing the various options for the 3D coordinate generation.

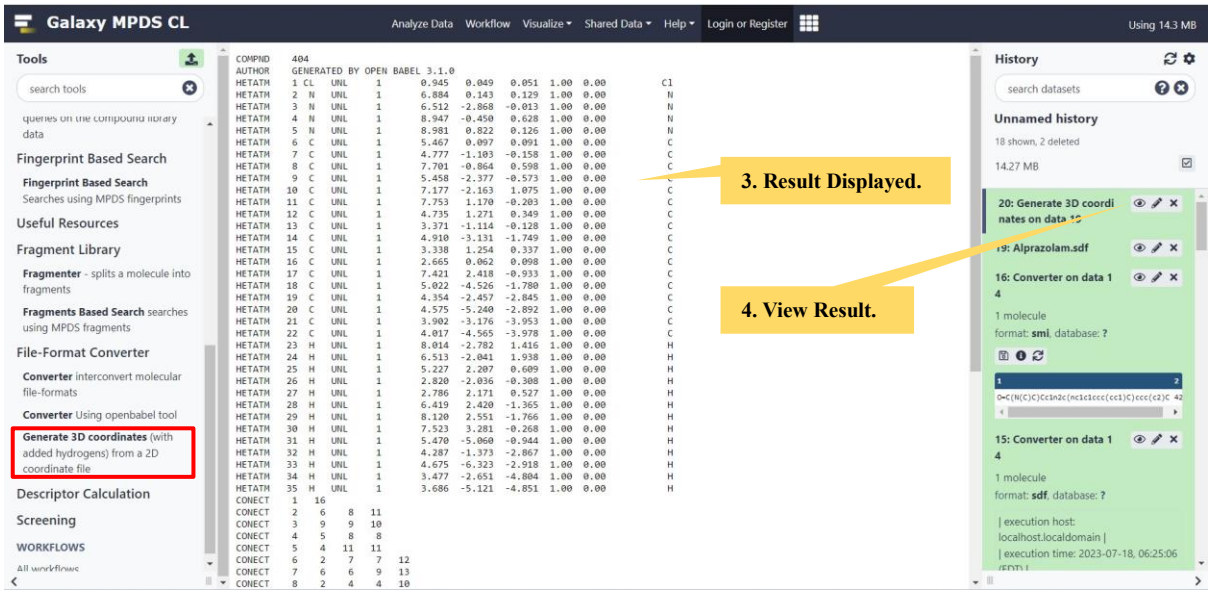


Figure 30. Showing the 3D coordinates using the Generate 3D coordinate module.

11. Descriptors Calculation

Molecular descriptors, both numerical and categorical in nature, serve as invaluable tools for the purpose of comparing and categorising molecules according to their distinct properties. These descriptors, in turn, enable the construction of predictive models that find application in a diverse range of domains, including but not limited to the prediction of drug efficacy, toxicity, and pharmacokinetics.

The realm of molecular descriptors encompasses a diverse array of calculable entities, including but not limited to topological, geometrical, electrostatic, and quantum mechanical descriptors. Molecular topology and connectivity can be effectively characterised by employing topological descriptors, which encompass crucial features such as molecular weight, the count of atoms, and the count of bonds. These descriptors serve as valuable indicators of the structural properties and connectivity patterns exhibited by molecules. Geometrical descriptors, encompassing molecular volume, surface area, and shape, serve as fundamental metrics that encapsulate the molecular dimensions and spatial configuration.

11.1 Padel Descriptors

In MPDS compound library, Padel descriptor tool has been integrated to calculate the descriptors. Padel Descriptors, a ubiquitous software tool in the realm of cheminformatics and drug discovery, stands as a prominent solution for the generation of molecular descriptors. Its extensive usage and adoption within the scientific community attest to its significance and efficacy in this field. This cutting-edge software has been meticulously engineered to perform intricate calculations, generating an extensive array of descriptors tailored specifically for the analysis of diminutive organic compounds. The extraction of diverse molecular features and properties from chemical structures is facilitated by Padel Descriptors, which leverages the open-source chemistry library CDK (Chemistry Development Kit).

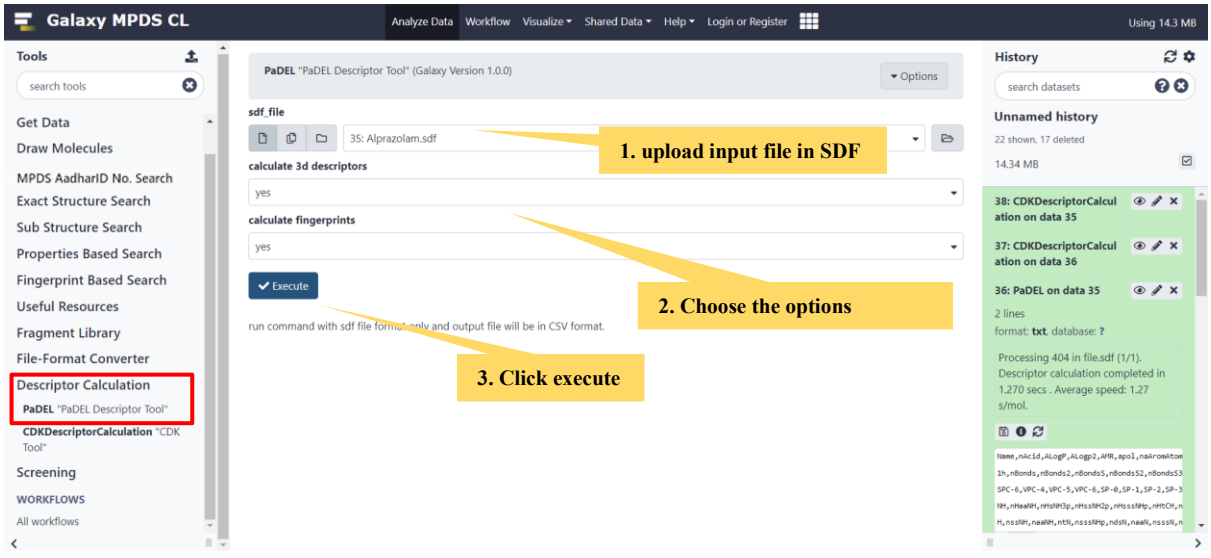


Figure 31. Showing the Padel descriptor tool with several options.

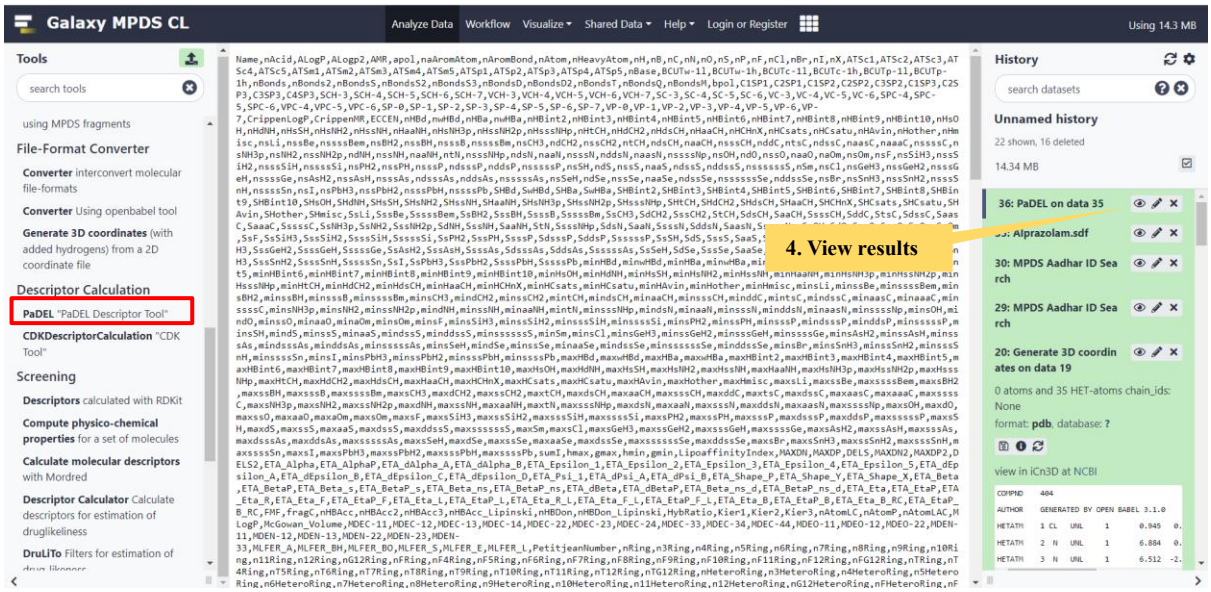


Figure 32. Showing the results of Padel descriptors.

11.2 CDK Descriptor Tool

In MPDS compound library, the CDK Descriptor Tool, which is a robust software tool employed in the domains of cheminformatics and computational chemistry, stands as a testament to its immense power and utility. The aforementioned is based on Java library that operates under an open-source licence, offering a comprehensive array of molecular descriptors and algorithms. These resources are specifically designed to facilitate the analysis and characterization of chemical compounds. The descriptors serve as quantitative representations of chemical properties and molecular features, playing a pivotal role in the realm of virtual screening, drug design, and various molecular modelling endeavours.

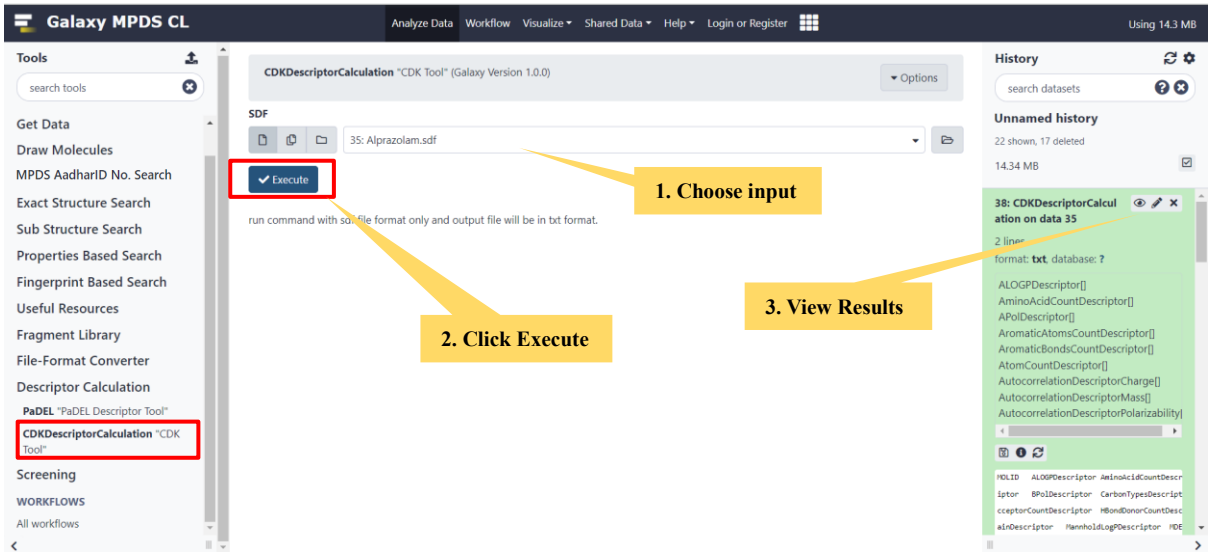


Figure 33. Showing the various options for CDK descriptor calculation.

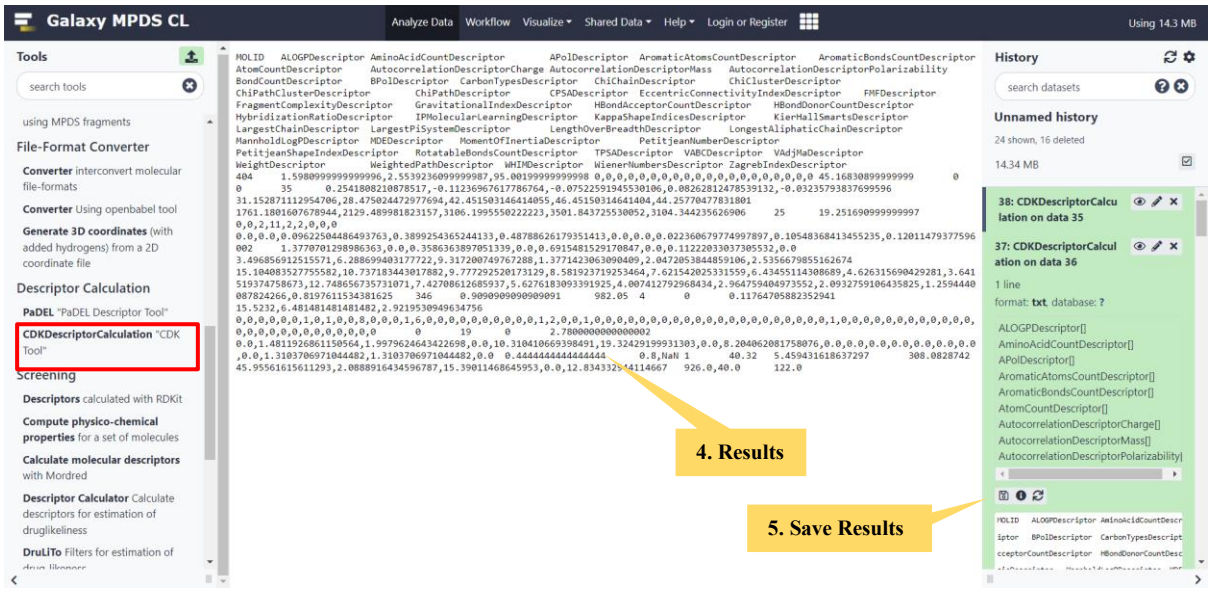


Figure 34. Showing the results for CDK descriptor calculation.

12 Screening

Screening is an essential procedure employed across diverse disciplines such as medicine, chemistry, and technology, with the aim of discerning and assessing prospective candidates or entities from a vast array of available options. It functions as an effective and methodical approach to selectively screen and rank resources, including drug compounds, chemical libraries, job applicants, or potential innovations. Screening is a widely utilised practise in medical environments, aimed at identifying initial indications of diseases or potential risk factors within specific populations. This proactive approach facilitates prompt interventions and preventive measures. The utilisation of virtual or high-throughput screening in the field of drug discovery enables researchers to systematically investigate extensive chemical spaces and discern compounds that exhibit desired properties, thereby facilitating subsequent testing and evaluation. The utilisation of effective screening methods has the potential to result in substantial progress and innovative discoveries, rendering it an essential instrument in contemporary research and decision-making practises spanning various fields.

The objective behind creating this tool is to develop a user-friendly interface that allows user to selectively assess various druglikeness filters for any given molecule. Well known filters like Lipinski, Veber, Ghose, QED etc., are incorporated here in order to estimate the druglikeness for any one or more molecules. Other filters like BCS, structural alerts and natural product likeness tools are also deployed to analyze the multifarious aspects of any given molecule.

12.1 Descriptors calculated with RDkit:

RDKit is a popular cheminformatics tools with multi-functionalities. One of the applications of this tool is to compute molecular descriptors and this function is incorporated under the 'Screening' module. The user can provide the molecule in .sdf format.

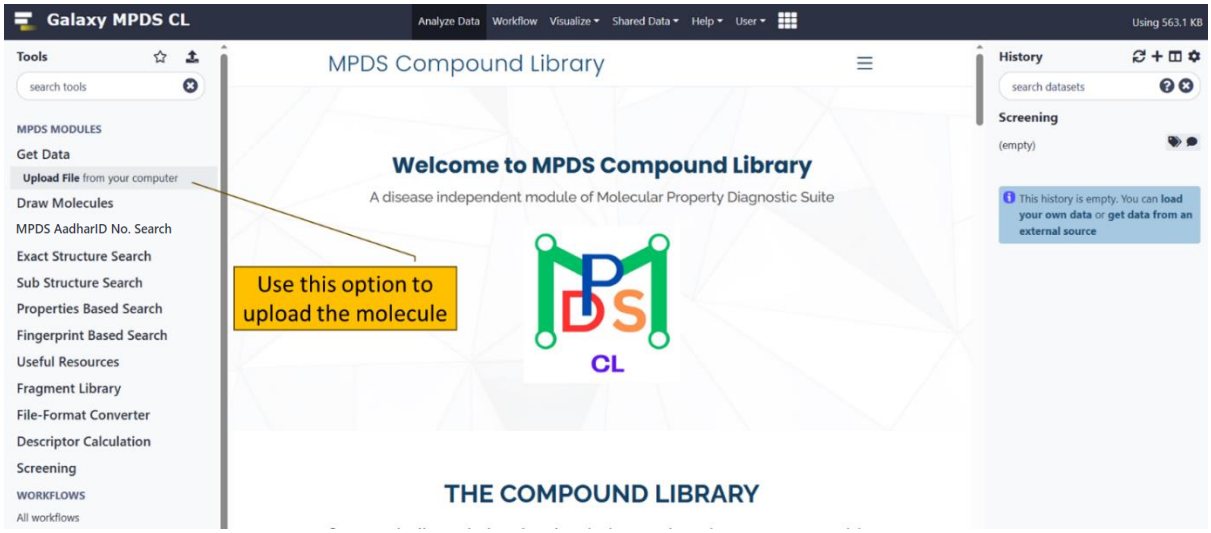


Figure35. Select the appropriate options as shown in the figure below.

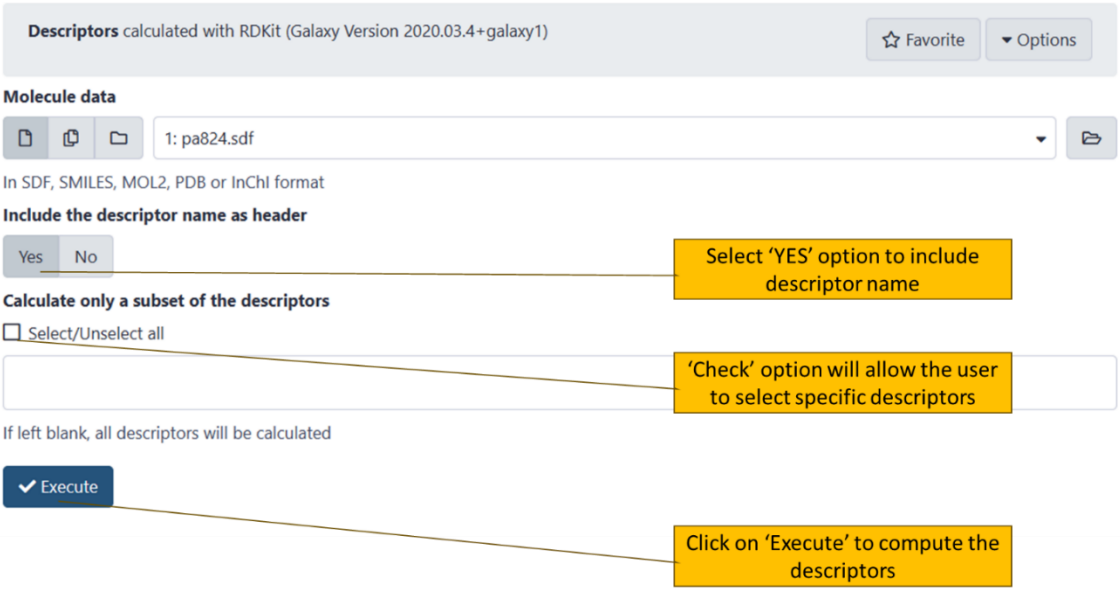


Figure 36. Options for computing the molecular descriptors using RDKit tool.

Figure 37. Resultant page showing the descriptors computed using RDKit tool.

12.2 Compute physico-chemical properties:

The determination of physico-chemical properties for a given set of molecules constitutes a pivotal undertaking within the realm of contemporary cheminformatics and drug discovery. The properties discussed herein encompass a diverse array of characteristics, spanning from molecular weight and lipophilicity to hydrogen bonding potential, solubility, and numerous others. These properties wield substantial influence over the behaviour and interactions of a compound within biological systems. Through the adept utilisation of cutting-edge computational tools and sophisticated software libraries such as RDKit, chemists and researchers are empowered to seamlessly and expediently compute a diverse range of properties pertaining to a given ensemble of molecules. The aforementioned process not only facilitates the acquisition of invaluable insights pertaining to the chemical landscape encompassed by the compounds, but also serves as a catalyst for the prioritisation and selection of potential drug candidates or chemical entities that warrant further investigation.

OpenBabel is deployed in this tool to compute the molecular descriptors.

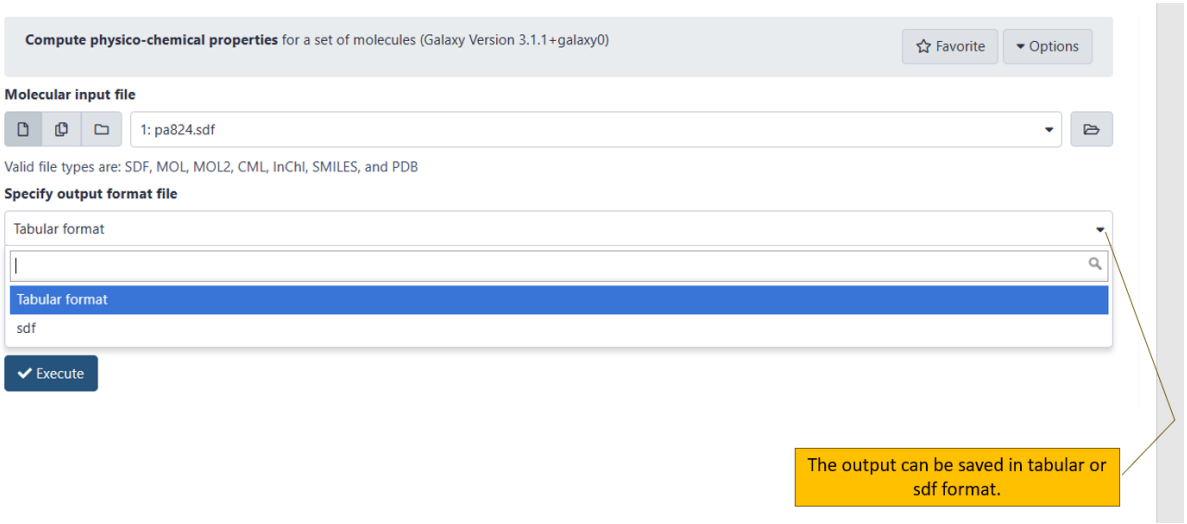


Figure 38. Options for computing the molecular descriptors using OpenBabel tool.

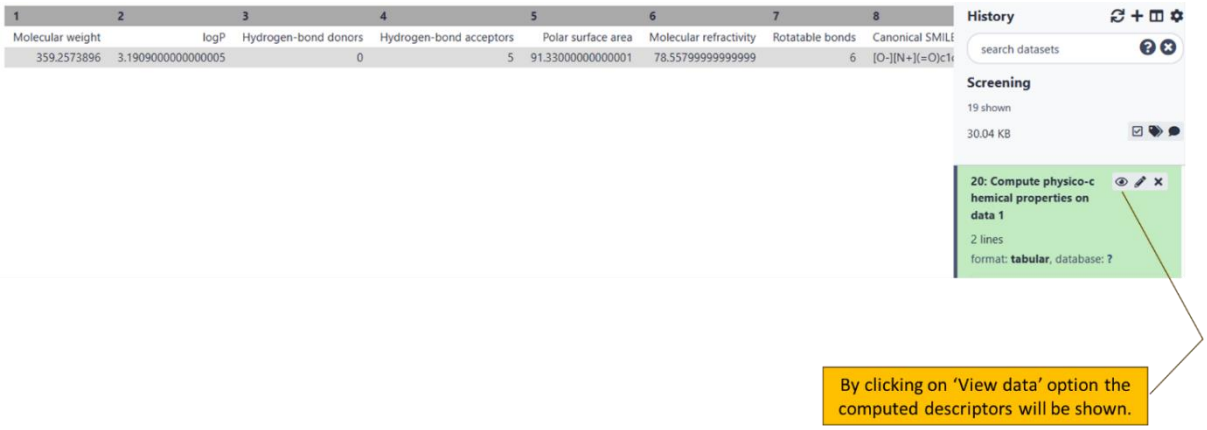


Figure 39. Resultant page showing the descriptors computed using OpenBabel tool.

12.3. Compute molecular descriptors with Modred.

Mordred, an influential Python library, has been meticulously crafted to facilitate the computation of molecular descriptors, thereby assuming a pivotal position within the realm of cheminformatics and computational chemistry. Molecular descriptors, as quintessential numerical representations of molecular properties, bestow upon us invaluable insights into the intricate structure and dynamic behaviour of compounds. Mordred, an advanced computational tool, presents a comprehensive array of descriptors that encompass diverse facets, including but not limited to 1D and 2D properties, constitutional characteristics, topological attributes, electronic properties, and geometrical features. The utilisation of Mordred empowers researchers and chemists to effectively derive an extensive repertoire of descriptors encompassing a wide spectrum of molecules, ranging from diminutive organic compounds to intricate biomolecules.

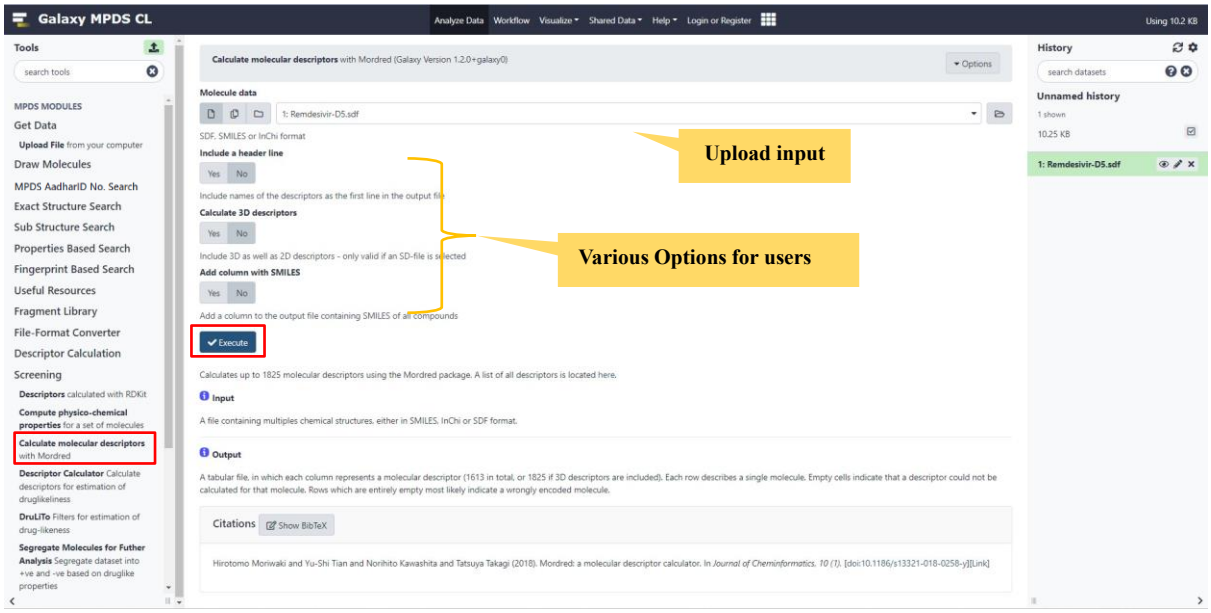


Figure 40. Showing various options for molecular descriptors with Modred**.**

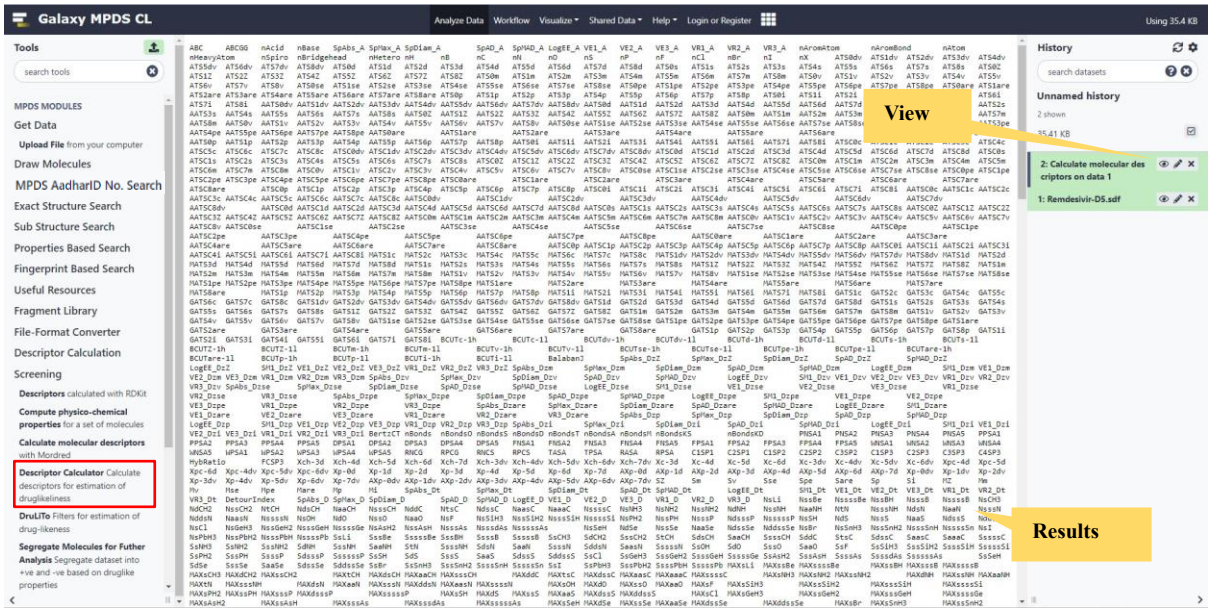


Figure 41. Showing various options for molecular descriptors with Modred**.**

12.4. Descriptor calculator:

The Descriptor Calculator stands as a pivotal instrument in the realm of drug discovery, assuming a paramount position in the estimation of the druglikeness of chemical compounds. The proposed methodology leverages a comprehensive array of molecular descriptors to assess a wide range of physicochemical properties and structural attributes inherent in molecules. These descriptors encompass crucial factors including molecular weight, lipophilicity, hydrogen bonding propensity, and polar surface area, among numerous others. By leveraging the power of computational analysis, researchers are able to evaluate the degree of concordance between a compound's properties and the characteristic features commonly observed in established pharmaceutical agents. The estimation of druglikeliness holds paramount importance during the nascent phases of drug development, serving as a pivotal tool in sieving and ranking potential drug candidates. This indispensable process substantially enhances the efficacy of lead optimisation, thereby streamlining the overall drug discovery endeavour.

Estimation of druglikeness of a molecule can be done based on the several rule of thumb such as Lipinski, Veber, Ghose, etc., In the tool, various properties can be computed based on which further druglikeness can be estimated.

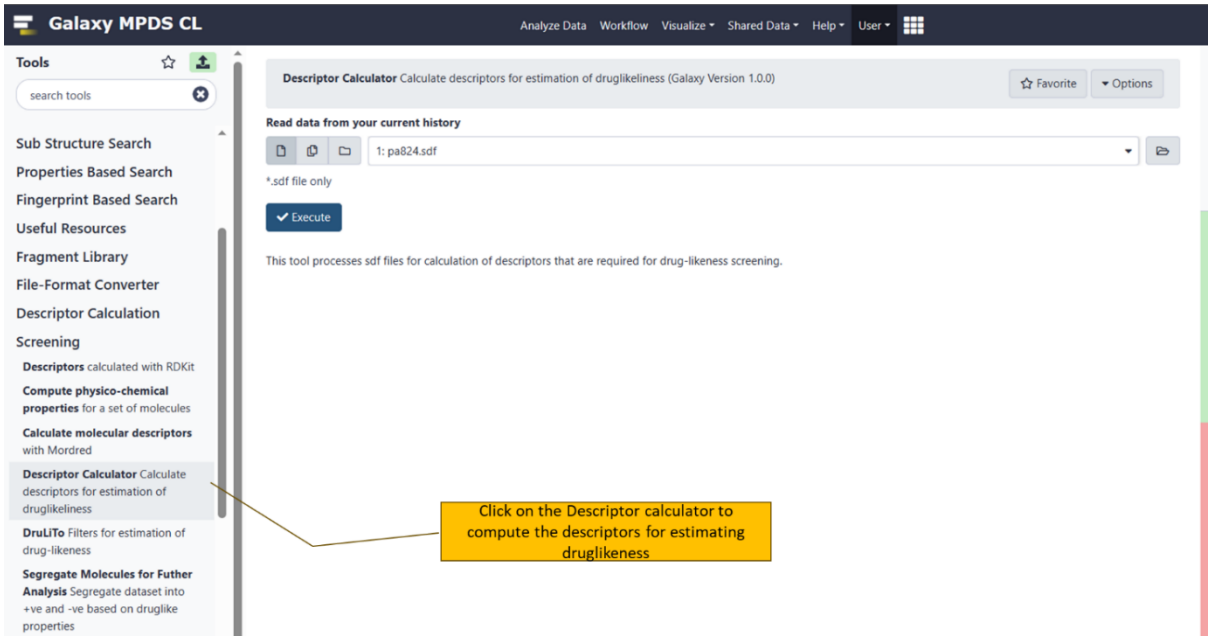


Figure 42. Options for estimating the druglikeness.

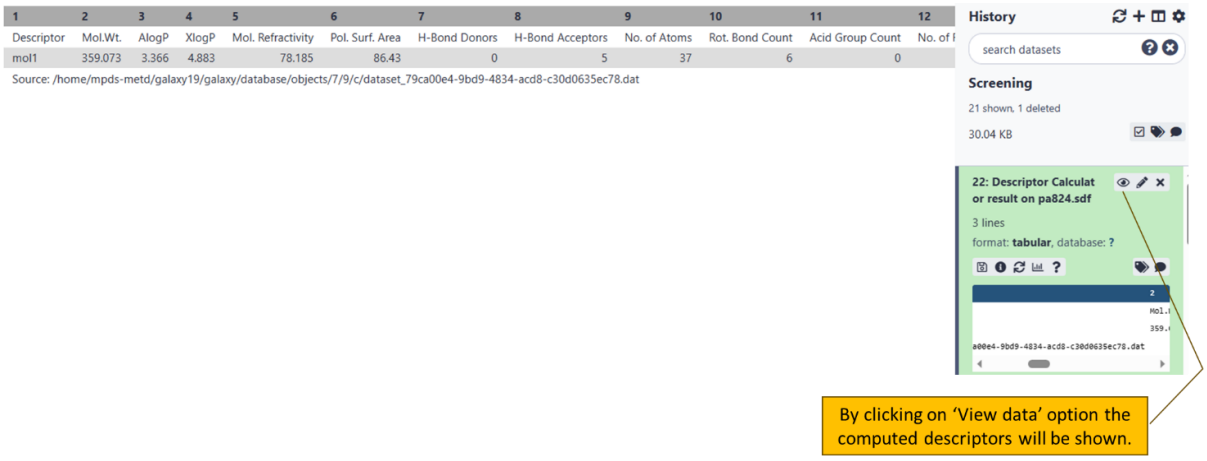


Figure 43. Resultant page showing the descriptors required for assessing druglikeness.

12.5. DruLiTo

Various druglikeness rules such like Lipinski, Veber, QED, etc., can be assessed for any given molecule using this tool. The user has to provide the result (molecular descriptor file) generated through the previous tool, i.e., 'Descriptor Calculator' as input to the current tool and remaining options is as shown in the below figure. User can check the druglikeness filters either for one filter or multiple filters in one go.

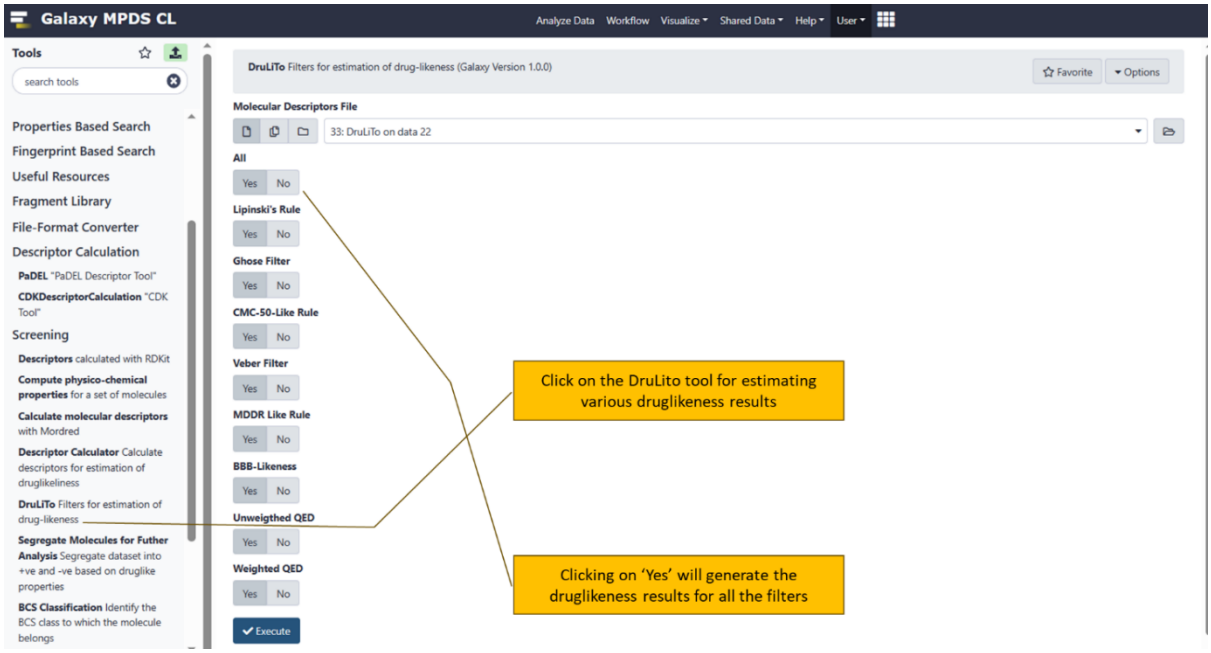


Figure 44. Options for assessing various druglike filters for any molecule using DruLiTo tool.

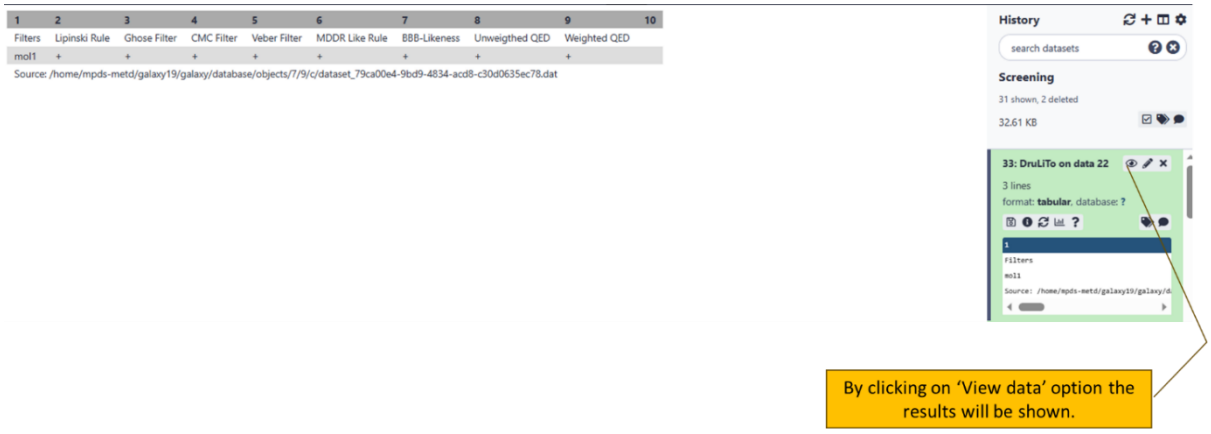


Figure 45. Resultant page showing the output from the DruLiTo tool .

As shown the above figure the molecule doesn't violate any of the rules as the results for all the filters is shown with '+' sign. If any filter is violated it will be shown with '-' sign. In this way the user can check for the druglikeness filters for one or multiple molecules through this tool.

12.6. Segregate molecules for further analysis:

Multiple molecules in an .sdf file can be segregated into positive or negative ligands based on the druglikeness filters of DruLito tool. The purpose of this analysis is to filter out the molecules which violates the druglikeness rules, which will help to keep only those molecules that doesn't violate any

rules and thus can be helpful in screening of the large number of molecules for any further ligand-based drug discovery process.

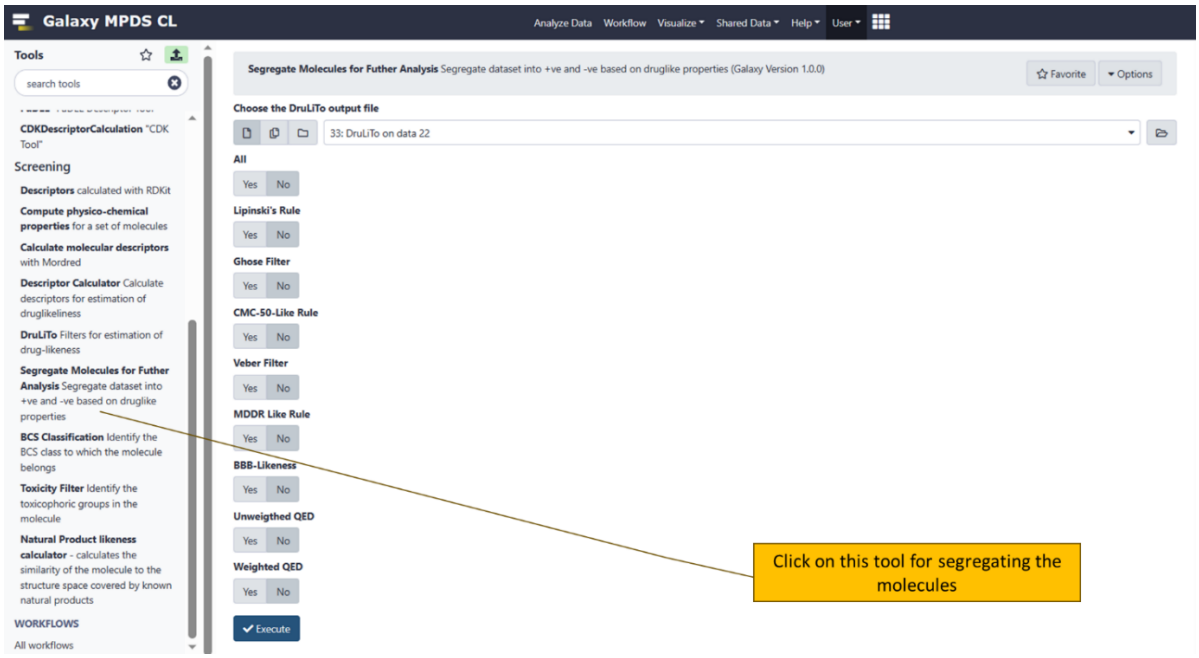


Figure 46. Options for segregating out molecules using DruLiTo tool.

12.7. BCS

Biopharmaceutical Classification System (BCS) is a system used in the pharmaceutical industry to categorize drugs based on their solubility and permeability features. The BCS classifies drugs into four classes (I to IV) based on two key parameters namely,

1. Solubility:
    a. Class I: High solubility - Drugs that are highly soluble in water.
    b. Class II: Low solubility - Drugs that have low solubility in water.
2. Permeability:
    a. Class I: High permeability - Drugs that are highly permeable through the intestinal membrane.
    b. Class II: Low permeability - Drugs that have low permeability through the intestinal membrane.
    c. Class III: High solubility but low permeability - Drugs that are highly soluble but have limited permeability.
    d. Class IV: Low solubility and low permeability - Drugs that have both low solubility and low permeability.

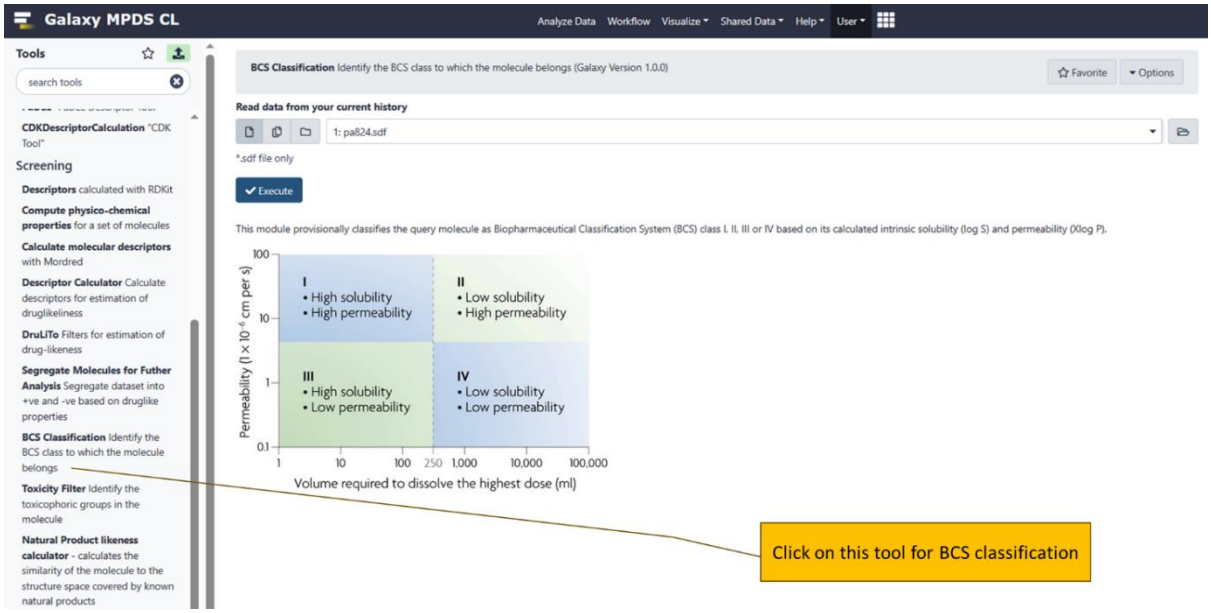


Figure 47. Options for assessing BCS for any molecule.

1	2	3	4	5	6
Molecule/Descriptor	logS	XlogP	BCS Class	Solubility	Permeability
mol1	-5.072	4.883	II	Low	High

BCS class	Solubility	Permeability
I	High	High
II	Low	High
III	High	Low
IV	Low	Low

Figure 48. Resultant page showing the output from the BCS tool .

Based on the BCS it is found that the molecule (PA824) belongs to Class II in BCS category.

12.8. Toxicity Filter:

This tool identifies and highlights the structural alerts or unwanted toxicophoric moieties in the submitted query molecule and renders a downloadable image and summary file. The complete set of results of the processed dataset can be downloaded as a compressed file using the link (Download All Results Here) on the page.

The file named "MPDS_ToxFilterResults_summary.txt" (default output file name) present in the folder provides a summary of results in a text format for all the molecules processed from the input dataset. This file contains the serial number of the molecule, the structural alerts (if present), and the number of times a specific alert occurred in the target molecule ("Occurrence count").

For molecules devoid of any structural alert, "No structural alerts found!" message would be displayed.

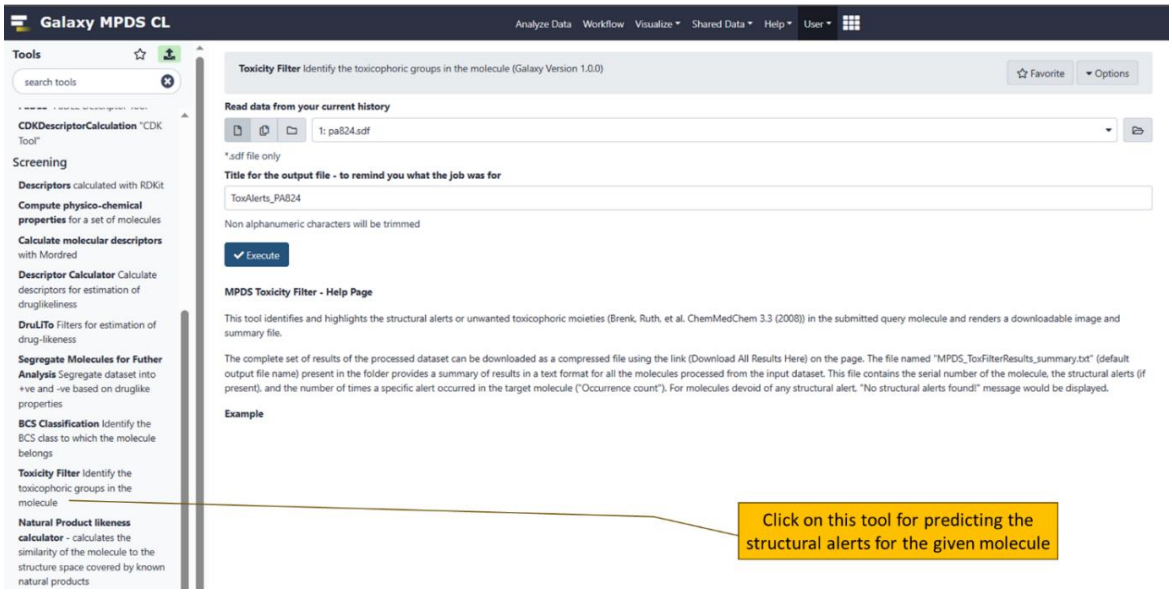


Figure 49. Options for predicting structural alerts for any molecule.

The summary of the results will be generated as shown below.

```
##################################################
#       Summary of Toxicity Filter results:      #
#       Date: Thu Jul 20 00:56:04 EDT 2023       #
##################################################

Molecule 1
Structural Alert found: nitro_group ([N+](=O)[O-])
Occurrence count: 1
Structural Alert found: Oxygen-nitrogen_single_bond ([OR0,NR0][OR0,NR0])
Occurrence count: 1
```

Figure 50. Resultant page showing the output from the Toxicity filter tool.

12.9. Natural Product likeness calculator

The Natural-Product-Likeness Scorer calculates the Natural Product (NP)-likeness of a molecule, i.e., the similarity of the molecule to the structure space covered by known natural products. The more positive the score, the higher the NP-likeness and vice versa.

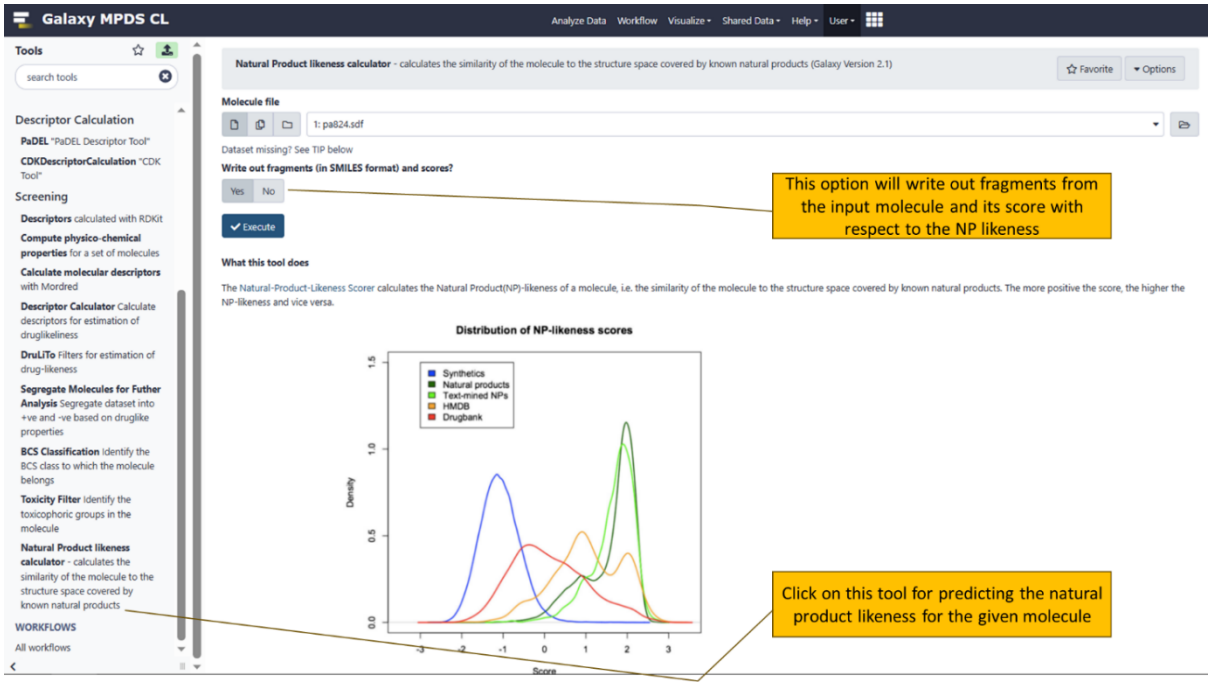


Figure 51. Options for predicting NP likeness for any molecule.

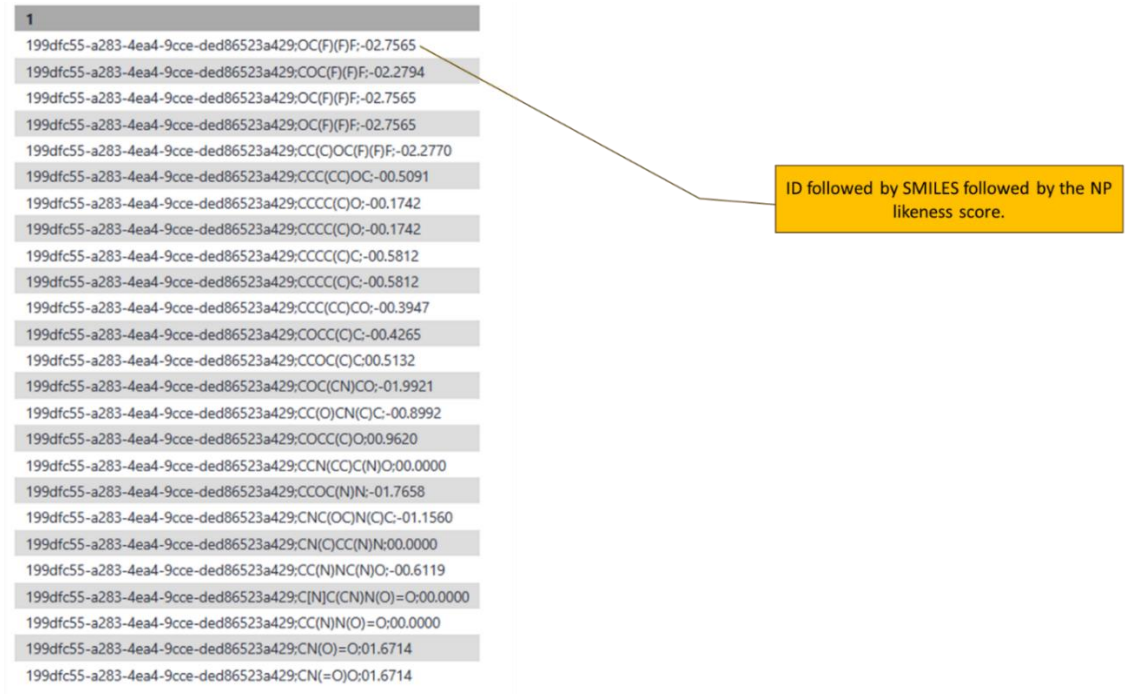


Figure 52. Resultant page showing the output from the NP likeness tool.

The result of the NP likeness tool is shown in the above figure, where the fragments from the molecule with their NP likeness score is mentioned. This result is obtained only if 'Write out fragments and score' is selected. On the other hand, if this option is not selected, then the output will be a .sdf file with a tag like **"> <NATURAL_PRODUCT_LIKENESS_SCORE>**

-00.6910" will be generated.

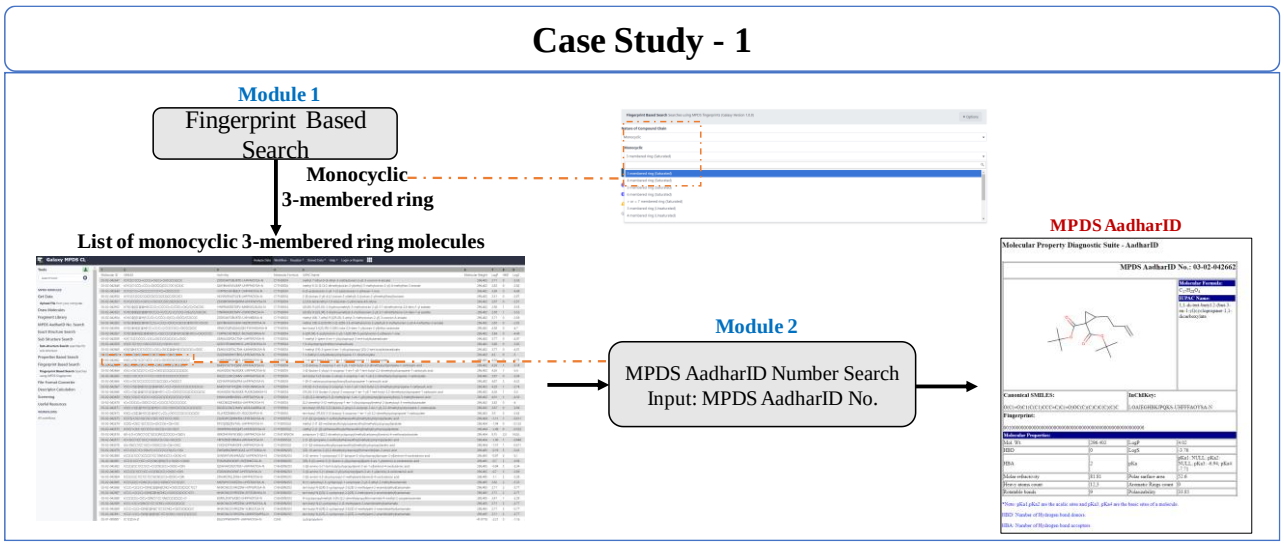


Figure 1: Workflow on case study 1 on the finger-based search and generation of MPDS AadharID containing the physicochemical properties of the compound.

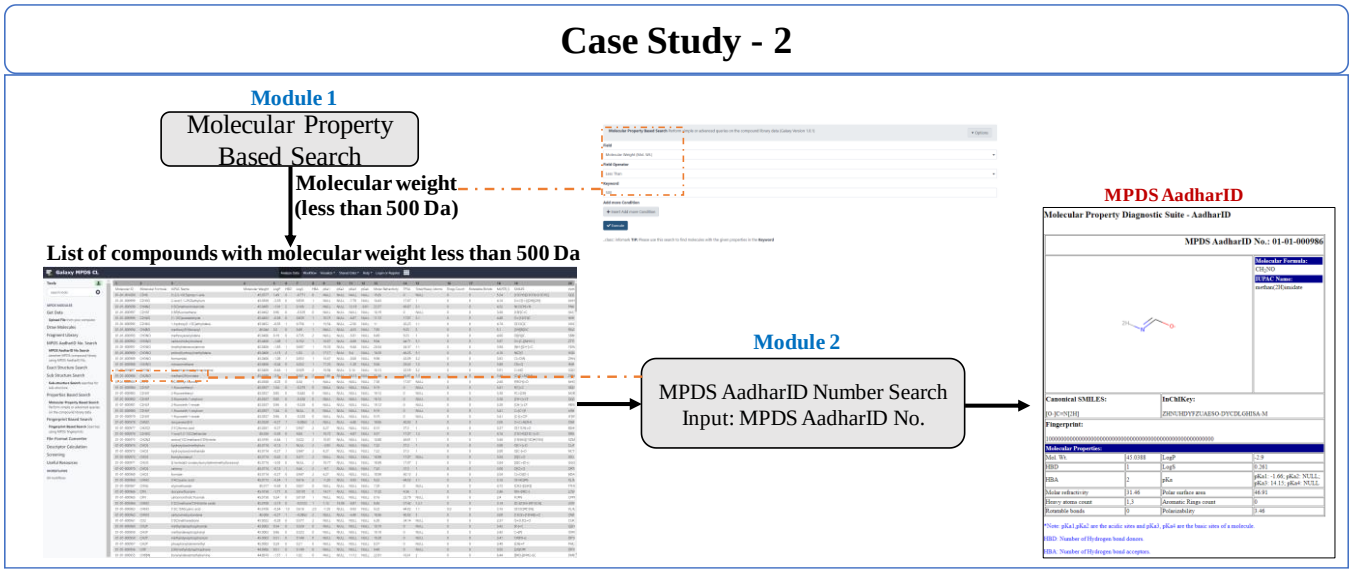


Figure 2: Workflow on case study 2 on the Molecular property-based search and generation of MPDS AadharID containing the physicochemical properties of the compound with molecular properties less than 500 Da.

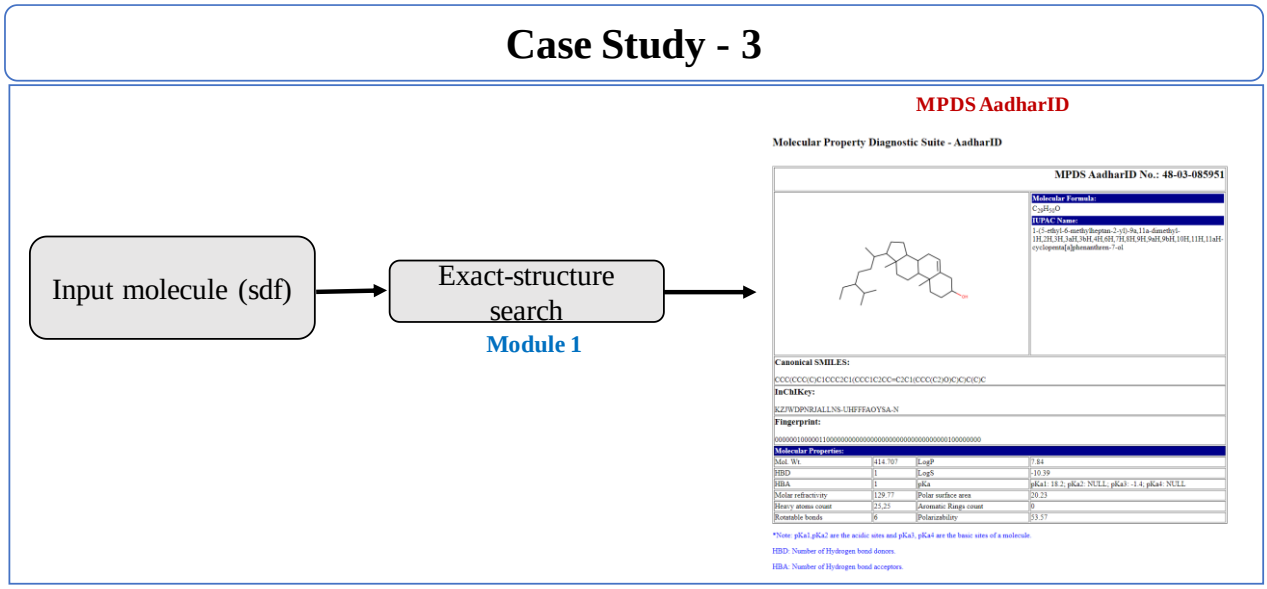


Figure 3: Workflow of case study 3 on generation of MPDS AadharID for a compound through exact search.ss

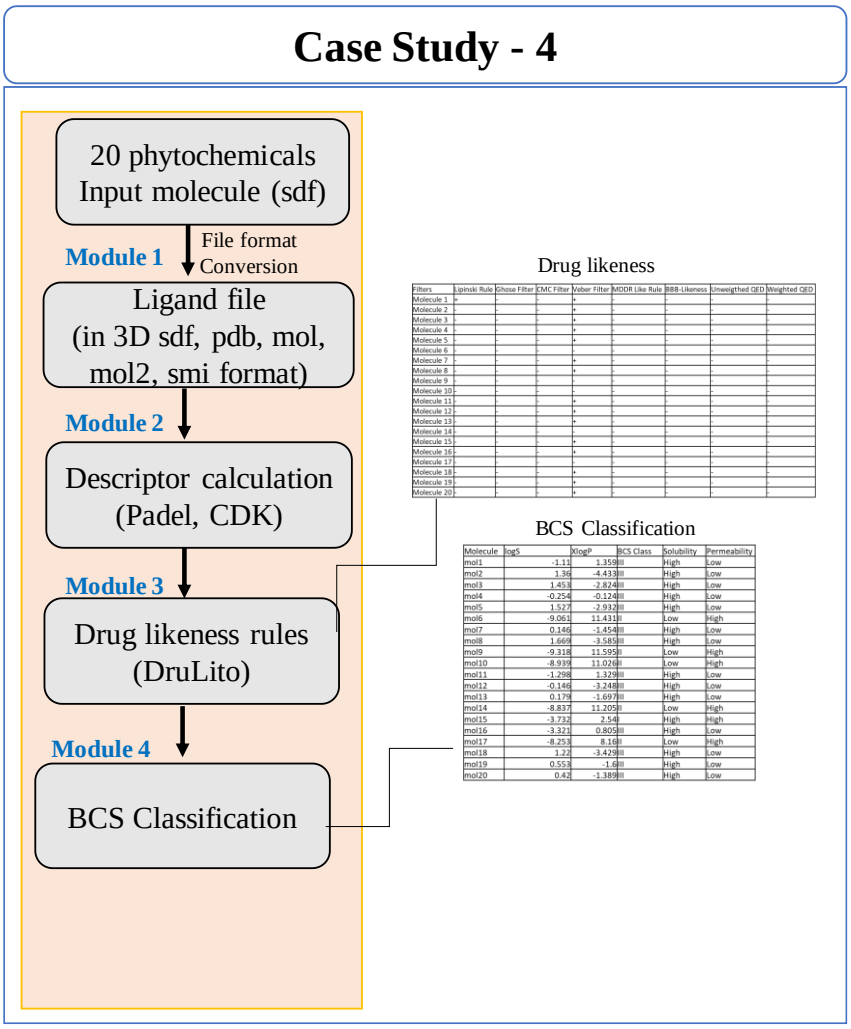


Figure 4: Workflow of case study 4 on screening phytochemicals using the modules of the MPDS-Compound Library

Case Study-5

Objective: Using the MPDS-CL, Analysis of FDA Approved Antiviral Drugs.

Molecules used for investigation: For this case study, the four Antiviral FDA Approved Drugs were taken into consideration:

1. Acyclovir
2. Oseltamivir
3. Ganciclovir
4. Sofosbuvir

Brief Description of the case study:

The four antiviral FDA approved drugs were taken and using the MPDS-CL modules, we performed:

1. Exact structure Search
2. Fragment based search using the fragment library
3. Descriptor Calculations
4. Screening thereby calculated
5. Toxicity Filter- This calculated the toxicophoric group present in the molecule
6. Natural Product Likeliness score

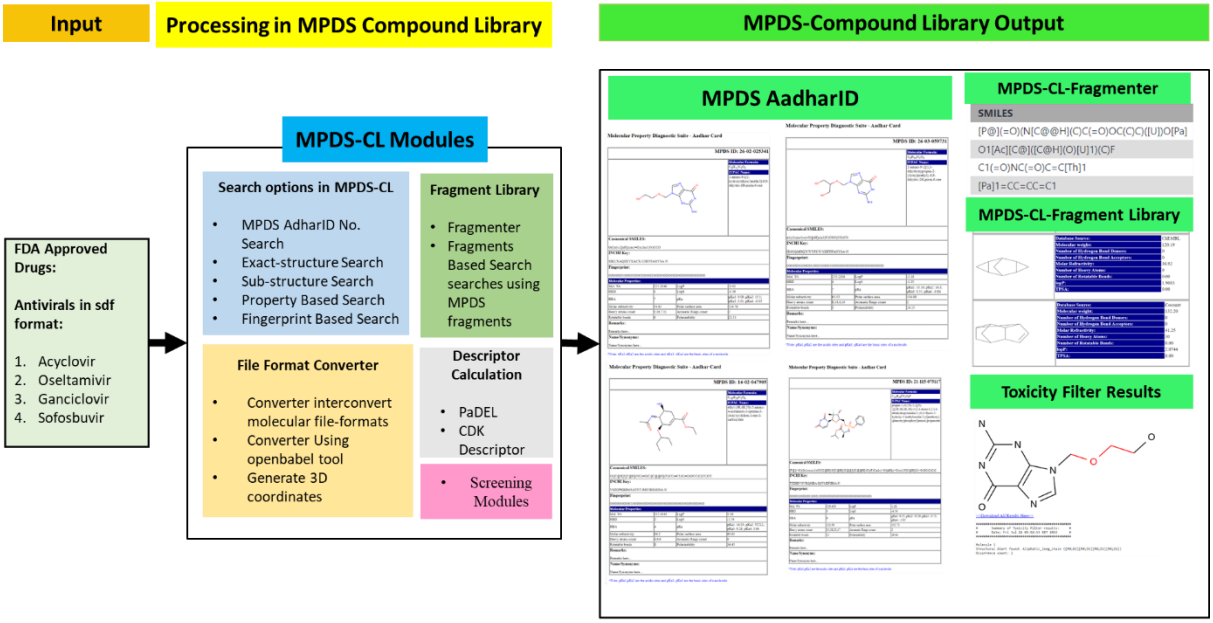


Figure 5: Case study of four antivirals showing the MPDS AdharID and associated analysis performed in MPDS-CL

References

1. Gaur, A. S., Bhardwaj, A., Sharma, A., John, L., Vivek, M. R., Tripathi, N., & Sastry, G. N. (2017). Assessing therapeutic potential of molecules: molecular property diagnostic suite for tuberculosis (MPDSTB) . Journal of Chemical Sciences, 129, 515-531.
2. Nagamani, S., Gaur, A. S., Tanneeru, K., Muneeswaran, G., Madugula, S. S., Consortium, M. P. D. S., ... & Sastry, G. N. (2017). Molecular property diagnostic suite (MPDS): Development of disease-specific open source web portals for drug discovery. SAR and QSAR in Environmental Research, 28(11), 913-926.
3. Gaur, A. S., Nagamani, S., Tanneeru, K., Druzhilovskiy, D., Rudik, A., Poroikov, V., & Sastry, G. N. (2018). Molecular property diagnostic suite for diabetes mellitus (MPDSDM): an integrated web portal for drug discovery and drug repurposing. Journal of Biomedical Informatics, 85, 114-125.
4. Badrinarayan, P., & Sastry, G. N. (2012). Virtual screening filters for the design of type II p38 MAP kinase inhibitors: a fragment based library generation approach. Journal of Molecular Graphics and Modelling, 34, 89-100.
5. Gaur, A. S., John, L., Kumar, N., Vivek, M. R., Nagamani, S., Mahanta, H. J., & Sastry, G. N. (2022). Towards systematic exploration of chemical space: building the fragment library module in molecular property diagnostic suite. Molecular Diversity, 1-10.
6. John, L., Soujanya, Y., Mahanta, H. J., & Narahari Sastry, G. (2022). Chemoinformatics and machine learning approaches for identifying antiviral compounds. Molecular Informatics, 41(4), 2100190.
7. John, L., Mahanta, H. J., Soujanya, Y., & Sastry, G. N. (2023). Assessing machine learning approaches for predicting failures of investigational drug candidates during clinical trials. Computers in Biology and Medicine, 153, 106494.
8. Jamir, E., Sarma, H., Priyadarsinee, L., Kiewhuo, K., Nagamani, S., & Sastry, G. N. (2022). A structure-based drug repurposing approach by considering the twenty four SARS-CoV2 Targets: A consensus scoring approach.
9. Jamir, E., Sarma, H., Priyadarsinee, L., Nagamani, S., Kiewhuo, K., Gaur, A. S., & Sastry, G. N. (2022). Applying polypharmacology approach for drug repurposing for SARS-CoV2. Journal of Chemical Sciences, 134(2), 57.
10. Jamir, E., Kiewhuo, K., Priyadarsinee, L., Sarma, H., Nagamani, S., & Sastry, G. N. (2022). Structure-function relationships among selected human coronaviruses.
11. O'Boyle, N. M., Banck, M., James, C. A., Morley, C., Vandermeersch, T., & Hutchison, G. R. (2011). Open Babel: An open chemical toolbox. Journal of cheminformatics, 3(1), 1-14.

12. Bento, A. P., Hersey, A., Félix, E., Landrum, G., Gaulton, A., Atkinson, F., & Leach, A. R. (2020). An open source chemical structure curation pipeline using RDKit. Journal of Cheminformatics, 12, 1-16.
13. Ertl, P., Roggo, S., & Schuffenhauer, A. (2008). Natural product-likeness score and its application for prioritization of compound libraries. Journal of chemical information and modeling, 48(1), 68-74.
14. Yap, C. W. (2011). PaDEL-descriptor: An open source software to calculate molecular descriptors and fingerprints. Journal of computational chemistry, 32(7), 1466-1474.
15. Steinbeck, C., Han, Y., Kuhn, S., Horlacher, O., Luttmann, E., & Willighagen, E. (2003). The Chemistry Development Kit (CDK): An open-source Java library for chemo-and bioinformatics. Journal of chemical information and computer sciences, 43(2), 493-500.
16. Afgan, E., Nekrutenko, A., Grüning, B. A., Blankenberg, D., Goecks, J., Schatz, M. C., & Briggs, P. J. (2022). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update. Nucleic acids research.
17. Singh Gaur, A., Nagamani, S., Priyadarsinee, L., Mahanta, H. J., Parthasarathi, R., & Sastry, G. N. (2023). Galaxy for open-source computational drug discovery solutions. *Expert Opinion on Drug Discovery*, *18*(6), 579-590.
18. Blankenberg, D., Von Kuster, G., Bouvier, E., Baker, D., Afgan, E., Stoler, N., ... & Nekrutenko, A. (2014). Dissemination of scientific software with Galaxy ToolShed. *Genome biology*, *15*(2), 1-3.
19. Bickerton, G.R., *et al.* (2012) Quantifying the chemical beauty of drugs. *Nat. Chem.*, **4**, 90-98.
20. Brenk, R., *et al.* (2007) Lessons learnt from assembling screening libraries for drug discovery for neglected diseases. *ChemMedChem*, **3**, 435-444.
21. Ghose, A.K., Viswanadhan, V.N. and Wendoloski, J.J. (1999) A knowledge-based approach in designing combinatorial or medicinal chemistry libraries for drug discovery. 1. A qualitative and quantitative characterization of known drug databases. *J. Com. Chem.*, **1**, 55-68.
22. Kerns, E. and Di, L. (2008) *Drug-like properties: concepts, structure design and methods: from ADME to toxicity optimization*. Academic Press.
23. Lipinski, C.A., *et al.* (1997) Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv. Drug Delivery Rev.*, **23**, 3-25.
24. Oprea, T.I. (2000) Property distribution of drug-related chemical databases*. *J. Comput-Aided. Mol. Des.*, **14**, 251-264.
25. Steinbeck, C., *et al.* (2003) The Chemistry Development Kit (CDK): An open-source Java library for chemo-and bioinformatics. *J. Chem. Inf. Comput. Sci.*, **43**, 493-500.
26. Veber, D.F., *et al.* (2002) Molecular properties that influence the oral bioavailability of drug candidates. *J. Med. Chem.*, **45**, 2615-2623.

27. Yusof, I. and Segall, M.D. (2013) Considering the impact drug-like properties have on the chance of success. *Drug Discov. Today*, 'in press'.
28. Hirotomo Moriwaki and Yu-Shi Tian and Norihito Kawashita and Tatsuya Takagi (2018). Mordred: a molecular descriptor calculator. In Journal of Cheminformatics, 10 (1). [doi:10.1186/s13321-018-0258-y.
29. Ertl, P., Roggo, S., & Schuffenhauer, A. (2008). Natural product-likeness score and its application for prioritization of compound libraries. Journal of chemical information and modeling, 48(1), 68-74.
30. Moriwaki, H., Tian, Y. S., Kawashita, N., & Takagi, T. (2018). Mordred: a molecular descriptor calculator. *Journal of cheminformatics*, *10*(1), 1-14.
31. O'Boyle, N. M., Morley, C., & Hutchison, G. R. (2008). Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit. *Chemistry Central Journal*, *2*(1), 1-7.
32. Mendez, D., Gaulton, A., Bento, A. P., Chambers, J., De Veij, M., Félix, E., ... & Leach, A. R. (2019). ChEMBL: towards direct deposition of bioassay data. Nucleic acids research, 47(D1), D930-D940.
33. Lelong, S., Zhou, X., Afrasiabi, C., Qian, Z., Cano, M. A., Tsueng, G., ... & Wu, C. (2022). BioThings SDK: a toolkit for building high-performance data APIs in biomedical research. *Bioinformatics*, *38*(7), 2077-2079.
34. Wahl, J., & Sander, T. (2022). Fully automated creation of virtual chemical fragment spaces using the open-source library OpenChemLib. *Journal of Chemical Information and Modeling*, *62*(9), 2202-2211.

Annexure

The list of useful Chemoinformatics tools is given below:

1. RDKit: RDKit is an open-source cheminformatics toolkit that provides a wide range of functionalities for molecular modeling and drug discovery. It allows users to work with molecular structures, fingerprints, substructure searching, and more.

2. Open Babel: Open Babel is an open-source chemical toolbox designed to speak the many languages of chemical data. It supports conversions between various chemical file formats and offers a range of molecular descriptor calculations.

3. ChemAxon: ChemAxon provides a suite of software tools and libraries for cheminformatics applications. Some of their products include JChem, Marvin, and Instant JChem, which offer functionalities such as chemical structure drawing, structure-based property prediction, and data management.

4. KNIME: KNIME is an open-source data analytics platform that includes various cheminformatics extensions. It allows users to perform data manipulation, molecular descriptor calculation, virtual screening, and more through a user-friendly graphical interface.

5. Cheminformatics Toolkit (CITK): CITK is a collection of cheminformatics algorithms and tools provided by the National Cancer Institute (NCI) to support drug discovery and cheminformatics research.

6. AutoDock: AutoDock is a popular molecular docking software used for predicting the binding modes of small molecules to proteins. It is widely used in virtual screening and structure-based drug design.

7. Pybel: Pybel is a Python wrapper for the Open Babel toolkit, allowing users to access Open Babel's functionalities and integrate them into Python scripts and workflows.

8. ChemDoodle: ChemDoodle is a chemical drawing and visualization software that allows users to create high-quality chemical structures and 2D/3D representations.

9. MOE (Molecular Operating Environment): MOE is a comprehensive software package for molecular modeling, visualization, and analysis. It is widely used in medicinal chemistry, protein modeling, and structure-based drug design.

10. ChemMine Tools: ChemMine Tools is a web-based platform that offers a wide range of cheminformatics utilities, including similarity searching, substructure searching, and molecular property calculation.

11. Cinfony: A Unifying Application Programming Interface (API) for Multiple Cheminformatics Toolkits. The software, being an open-source solution, offers users the opportunity to readily install it on their Windows or Linux operating systems without incurring any financial costs.

12. ChemDoodle: It is an exemplary JavaScript library that empowers users with the ability to create stunning 2D chemical graphics. With its robust features and intuitive interface, ChemDoodle revolutionises the field of chemical visualisation. This cutting-edge library seamlessly integrates into web applications, providing a seamless user experience. By harnessing the power of JavaScript, ChemDoodle empowers developers to effortlessly generate high-quality chemical.

13. OSRA: The Optical Structure Recognition Application (OSRA) is an advanced software tool that leverages advanced machine learning techniques to seamlessly convert intricate graphical depictions of chemical structures into standardised SMILES or SD files.

14. CDKjs, an Open Source modular Java libraries for Cheminformatics and exquisite JavaScript rendition of the Chemistry Development Kit (CDK) library, stands as a testament to the remarkable progress made in the realm of computational chemistry. This cutting-edge implementation empowers developers with the ability to harness the power of CDK's robust functionality within the JavaScript ecosystem.

15. ChemmineR: It is a cheminformatics package for analyzing drug-like small molecule data in R.

16. Enalos Nodes for KNIME: Enalos Nodes are chemoinformatics tools developed by NovaMechanics Ltd. Novamechanics Ltd is an in silico drug design company committed to the computer aided design of small molecule medicines for a very wide range of target classes.

Useful Resources provided on MPDS-CL

The list of useful resources provided in the MPDS-CL is as follows:

1. ChEMBL
2. mychem.info
3. OpenChemLib

Frequently Asked Questions (FAQs)

1. What is a compound library?

A compound library, in the realm of drug discovery and chemical biology, refers to a meticulously curated collection of diverse small molecules that are systematically screened for their potential to interact with biological targets. These libraries serve as invaluable resources for researchers seeking novel therapeutic agents or probes for studying biological processes.

A compound library represents an assemblage of an extensive array of chemical compounds, meticulously arranged and preserved, serving as a valuable resource for a multitude of scientific investigations and the pursuit of novel therapeutic agents. These libraries serve as invaluable assets for scientists, researchers, and pharmaceutical enterprises in their quest to discern promising drug candidates, explore intricate biological pathways, and expedite the drug development trajectory.

2. How are compounds selected for inclusion in a compound library?

The selection of compounds within a library is commonly guided by various criteria, including but not limited to chemical diversity, synthetic accessibility, and established or anticipated biological activities. The primary objective entails the incorporation of a diverse array of molecular architectures, encompassing a wide spectrum of chemical space, thereby encompassing a multitude of potential target interactions.

3. What are the different types of compound libraries available?

Compound libraries encompass a diverse array of categories, each serving a distinct purpose within the realm of drug discovery. Natural product libraries, for instance, comprise compounds sourced from nature itself, while focused libraries revolve around compounds meticulously crafted to align with specific target classes or biological functions. On the other hand, diversity-oriented libraries prioritise chemical diversity, thereby maximising the potential for novel discoveries. Lastly, fragment libraries consist of diminutive molecular weight compounds that prove invaluable in fragment-based drug discovery endeavours.

4. What are the main applications of compound libraries in drug discovery?

Compound libraries are of utmost importance in the multifaceted realm of drug discovery, as they serve as indispensable tools throughout its various stages. These stages encompass target identification and validation, hit identification, lead optimisation, and the elucidation of structure-activity relationships (SAR). In the realm of high-throughput screening (HTS) campaigns, these tools find extensive utility in the identification of compounds that effectively modulate targeted biological entities.

5. How are compound libraries screened for potential drug candidates?

High-throughput screening (HTS) represents a widely employed approach for the assessment of compounds within a library for targeted biological assays. In the course of this intricate procedure, an extensive array of compounds, ranging from thousands to millions, are subjected to swift and rigorous testing with the aim of discerning those that manifest the coveted activity against the designated target of interest.

6. Can I access publicly available compound libraries for research purposes?

Yes, access to compound libraries is available for research purposes through a number of public databases and projects. To encourage cooperation and speed up scientific advancement, academic institutions, governmental bodies, and non-profit organisations frequently make these resources available.

7. What is chemical space?

In chemistry, "chemical space" is the theoretical region that includes all conceivable chemical compounds, each of which has its own unique structure and set of attributes. It's meant to symbolise the incredible variety of possible chemical entities.

8. How does chemical space sparsity impact drug discovery?

The sparsity of chemical space poses a significant constraint on the breadth of compound diversity accessible for screening purposes, thereby potentially impeding the identification of highly promising drug candidates. Furthermore, the potential exists for the impeding of the identification of compounds possessing distinctive mechanisms of action, thereby limiting the advancement of groundbreaking pharmaceuticals.

9. What are the challenges in exploring chemical space?

Exploration of chemical space poses a formidable challenge due to its vast expanse, which is believed to encompass an astronomical multitude of compounds. The sheer magnitude of this expanse renders it virtually unfeasible to empirically investigate the entirety of potential compounds.

10. Why is the exploration of chemical space important?

The exploration of chemical space plays a pivotal role in the realms of drug discovery and materials development. The utilisation of machine learning techniques holds immense potential in facilitating the

emergence of groundbreaking drug candidates and the unearthing of novel materials boasting unparalleled properties and functionalities.

11. How can we deal with compound redundancy?

In order to address the issue of compound redundancy, researchers can employ compound curation algorithms to effectively merge or eliminate duplicate compounds within databases. The utilisation of data clustering techniques presents a valuable approach in the identification of representative compounds and the mitigation of redundancy.

12. What is the role of machine learning in exploring chemical space?

Predicting compound characteristics, discovering novel structures, and optimising chemistry processes are all areas where machine learning plays an important role in expanding our understanding of the chemical universe. It can direct experiments and allocate money to the most promising regions of the chemical universe.

MOLECULAR PROPERTY DIAGNOSTIC SUITE -COMPOUND LIBRARY

(MPDS-CL)

http://mpds.neist.res.in:8086/

(SCRIPTS, USED AND DEPOSITED IN GITHUB)

Table of Contents

1. generate.py: To generate the fingerprints from canonical smiles
2. checknature.py:To check all the bonds in a given ring are pure aromatic, pure aliphatic or mixed.
3. mono3to19.py: To check the molecules belongs to class 03 to 19 (If it is monocyclic group) based on the size of the ring and bond types. Along with this, checks the fused rings and neighbors.
4. bi_cl21_to37.py: To check the molecules belongs to which class from class 21 to class 37 (If it is bicyclic group) based on ring count and number of carbon atoms in each ring.
5. tr_cl38_to46.py: To check the molecules belongs to which class from class 28 to class 46(if it is tricyclic group) based on ring count and number of carbon atoms in each ring. Along with this, checks the fused and connected rings
6. tt_cl47_to50.py: To check the molecules belongs to which class from class 28 to class 46(if it is tetracyclic group) based on ring count and number of carbon atoms in each ring
7. multiple_cl20.py: To check the molecules belongs to class 20(If it is mono cyclic with multiple element group in a ring) based on ring count and number of carbon atoms in each ring
8. po_cl51.py: To check the molecules belongs to which class from class 51(if it is pentacyclic group) based on ring count and number of carbon atoms in each ring.

MPDS-CL SCRIPTS

Following is the list of python scripts employed in MPDS-CL:

1. generate.py: To generate the fingerprints from canonical smiles

```
##############################################################################
# MPDS-CL: 1. generate.py
# DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA
##############################################################################

import re,sys,os,getopt
from openbabel import pybel
import rdkit as rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
import checknature as ck
import mono_3to19 as mono3to19
import bi_cl21to37 as bi21to37
import tr_cl38to46 as tr38to46
import tt_cl47to50 as tt47to50
import po_cl51 as po51
import multiple_cl20 as multiple20
import datamol as dm
patt1=pybel.Smarts("([R])")
unsat_patt=pybel.Smarts("[#6]=,#[#6]")
patt3=pybel.Smarts("[a;R]")
patt4=pybel.Smarts("[n,o,s,p;R]")
patt5=pybel.Smarts("[c]:[!c]")
patt6=pybel.Smarts("[R]=[R]")
patt7=pybel.Smarts("[R]#[R]")
patt8=pybel.Smarts("[C,S,N,O,P,A;R]")
conpat=pybel.Smarts("[R]!@[R]")
fusedpat=pybel.Smarts("[R2]")
conpat_ali=pybel.Smarts("[R&A]!@[R&A]")
fusedpat_ali=pybel.Smarts("[R2&A]")
conpat_aro=pybel.Smarts("[R&a]!@[R&a]")
fusedpat_aro=pybel.Smarts("[R2&a]")
fusedpat_mx=pybel.Smarts("[R&a][R&A]")
conpat_mx=pybel.Smarts("[R&a]!@[R&A]")
carbon_patt1=pybel.Smarts("[C,c]")
carbon_patt2=pybel.Smarts("[#6]")
ringpat=pybel.Smarts("[R]")
trans=pybel.Smarts("[#21,#22,#23,#24,#25,#26,#27,#28,#29,#30,#39,#40,#41,#42,#43,#44,#45,#46,
#47,#48,#57,#58,#59,#60,#61,#62,#63,#64,#65,#66,#67,#68,#69,#70,#71,#72,#73,#74,#75,#76,#77,#
78,#79,#80,#89,#90,#91,#92,#93,#94,#95,#96,#97,#98,#99,#100,#101,#102,#103,#104,#105,#106,#
107,#108,#109,#110,#111,#112]")
```

```
def fp_gen(input_smiles,mol_wt):
	smiles = input_smiles
	mol_wt = mol_wt
	mol_wt_1 = str(mol_wt)
	o1=open('/home/mpds-metd/galaxy19/galaxy/tools/mpds-
tools/module_mpds_databases/exactsearch/CL_56_FP.txt', "a+")
	o1.truncate(0)
	m=Chem.MolFromSmiles(smiles)
	if m is None:
		print('M is none')
		m2=dm.to_mol(smiles,sanitize=False)
		with dm.without_rdkit_log():
			fixed_mol=dm.fix_valence_charge(m2)
			Chem.SanitizeMol(fixed_mol)
			m=Chem.MolToSmiles(fixed_mol)
	fp_list=['0']*56
	num=[]
	mymol=pybel.readstring("smi",smiles)
	check=isinstance(m,rdkit.Chem.rdchem.Mol)
	result1=patt1.findall(mymol)
	result3=patt3.findall(mymol)
	result4=patt4.findall(mymol)
	result5=patt5.findall(mymol)
	result6=patt6.findall(mymol)
	result7=patt7.findall(mymol)
	result8=patt8.findall(mymol)
	conpat_res=conpat.findall(mymol)
	fusedpat_res=fusedpat.findall(mymol)
	carbon_res1=carbon_patt1.findall(mymol)
	carbon_res2=carbon_patt2.findall(mymol)
	ring_res=ringpat.findall(mymol)
	trans_res=trans.findall(mymol)
	number1=''
#Class 2
	if len(carbon_res1)==0:
		fp_list[1]='1'
	if len(carbon_res2)==0:
		fp_list[1]='1'
#FP for class 1 and class 2
	elif (m!=None):
		result1=patt1.findall(mymol)
		unsat_res=unsat_patt.findall(mymol)
		result3=patt3.findall(mymol)
		result4=patt4.findall(mymol)
		result5=patt5.findall(mymol)
		result6=patt6.findall(mymol)
		result7=patt7.findall(mymol)
		result8=patt8.findall(mymol)
		conpat_res=conpat.findall(mymol)
		fusedpat_res=fusedpat.findall(mymol)
		ring_res=ringpat.findall(mymol)
```

```
        inorganic1_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSmarts(
"[C,c]"))

        inorganic2_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSmarts(
"[#6]"))

        trans_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSmarts("[#21,
#22,#23,#24,#25,#26,#27,#28,#29,#30,#39,#40,#41,#42,#43,#44,#45,#46,#47,#48,#57,#58,#59,#60,#
61,#62,#63,#64,#65,#66,#67,#68,#69,#70,#71,#72,#73,#74,#75,#76,#77,#78,#79,#80,#89,#90,#91,#9
2,#93,#94,#95,#96,#97,#98,#99,#100,#101,#102,#103,#104,#105,#106,#107,#108,#109,#110,#111,#
112]"))
#Class 1
                if (len(result1)==0):
                        fp_list[0]='1'
                if (len(result1)>0):
                        bis = m.GetSubstructMatches(Chem.MolFromSmarts('[R]!@[!R]'))
                        bis2 = m.GetSubstructMatches(Chem.MolFromSmarts('[R]!@[R]'))
#Class 3 to 19

        fp_list=mono3to19.checkmono(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,conpat,conpat_r
es)
#Class 20 to 36
                        fp_list=bi21to37.checkbi(bis,bis2,o1,smiles,fp_list,mymol,m)
#Class 37 to 45

        fp_list=tr38to46.checktr(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,conpat,co
npat_res)
#Class 46 to 49

        fp_list=tt47to50.checktt(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,conpat,co
npat_res)
#Class 50

        fp_list=po51.checkpo(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,conpat,conp
at_res)
#Class 51

        fp_list=multiple20.checkspecial(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,co
npat,conpat_res)
#Class 2
                if len(carbon_res1)==0:
                        fp_list[1]='1'
                if len(carbon_res2)==0:
                        fp_list[1]='1'
                if len(inorganic1_rdk)==0:
                        fp_list[1]='1'
                if len(inorganic2_rdk)==0:
                        fp_list[1]='1'
#Class 52
                if len(trans_res)==1:
```

```
                        fp_list[51]='1'
                if len(trans_rdk)==1:
                        fp_list[51]='1'
#Class 53
                if len(trans_res)==2:
                        fp_list[52]='1'
                if len(trans_rdk)==2:
                        fp_list[52]='1'
#Class 54
                if len(trans_res)>=3:
                        fp_list[53]='1'
                if len(trans_rdk)>=3:
                        fp_list[53]='1'
#Class 55
                if float(mol_wt)>=750.00 and float(mol_wt)<=1200.99:
                        fp_list[54]='1'
                        fp_final = "".join(fp_list)
#Class 56
                if float(mol_wt)>=1201.00:
                        fp_list[55]='1'
                        fp_final = "".join(fp_list)
                fp_final = "".join(fp_list)
#               print("Done")
                o1.write(fp_final+ "\t" + mol_wt_1)
#               print(fp_final)
        else:
                pass
```

2. checknature.py:To check all the bonds in a given ring are pure aromatic, pure aliphatic or mixed.

```
################################################################################
# MPDS-CL: 2. checknature.py
#          DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA
################################################################################

import re
def isRingAromatic(mol, bondRing):
    for id in bondRing:
      if not mol.GetBondWithIdx(id).GetIsAromatic():
        return False
    return True
#Check the aliphatic or aromatic or mixed type of molecules
def checknature(m,z,result3,result4,result5,result6,result7,result8):
        ri = m.GetRingInfo()
        pure_arom=0
        pure_alip=0
        mixed_arom=0
        if isRingAromatic(m, ri.BondRings()[0]) == True:
#Pure-arom
                if len(result3)>=1 and len(result8)==0:
                        pure_arom=1
                if len(result4)>=1 and len(result8)==0:
                        pure_arom=1
                if re.search(r'(c)',z) and re.search('^((?!C|N|O|P|S).)*$',z) and len(result8)==0:
                        pure_arom=1
                if len(result5)>=1 and len(result8)==0:
                        pure_arom=1
#Mixed-arom
                if len(result3)>=1 and len(result8)>=1:
                        mixed_arom=1
#                       print("Found mixed ring")
                if len(result4)>=1 and len(result8)>=1:
                        mixed_arom=1
#                       print("Found mixed ring")
#Pure-ali
                if len(result3)==0 and len(result4)==0 and len(result6)>=0 and len(result7)>=0 and
len(result8)>=1 and re.search('^((?!c|n|o|s|p).)*$',z):
                        pure_alip=1
        else:
                #Pure-arom
                if len(result3)>=1 and len(result8)==0:
                        pure_arom=1
                if len(result4)>=1 and len(result8)==0:
                        pure_arom=1
```

```
                if re.search(r'(c)',z) and re.search('^((?!C|N|O|P|S).)*$',z) and len(result8)==0:
                        pure_arom=1
                if len(result5)>=1 and len(result8)==0:
                        pure_arom=1
#Mixed-arom
                if len(result3)>=1 and len(result8)>=1:
                        mixed_arom=1
                        print("Found mixed ring")
                if len(result4)>=1 and len(result8)>=1:
                        mixed_arom=1
                        print("Found mixed ring")
#Pure-ali
                if len(result3)==0 and len(result4)==0 and len(result6)>=0 and len(result7)>=0 and
len(result8)>=1 and re.search('^((?!c|n|o|s|p).)*$',z):
                        pure_alip=1
        return pure_alip,pure_arom,mixed_arom
```

3. mono3to19.py: To check the molecules belongs to class 03 to 19 (If it is monocyclic group) based on the size of the ring and bond types. Along with this, checks the fused rings and neighbours.

```
################################################################################
```

MPDS-CL: 3. mono3to19.py

DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA

```
################################################################################
```

```
import re,sys,os,getopt
from openbabel import pybel
import rdkit as rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import rdmolops
import checknature as ck
import ringcount as rc
import cleansmiles as cs
def checkmono(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,conpat,conpat_res):
        ssr = Chem.GetSymmSSSR(m)
   # Loop over each ring and print its properties
        for i, ring_atoms in enumerate(ssr):
                three_ring_type = 0
                four_ring_type = 0
                five_ring_type = 0
                six_ring_type = 0
                ge_seven_ring_type = 0
                ring_size = len(ring_atoms)
                if ring_size == 3:
                        three_ring_type = 1
                if ring_size == 4:
                        four_ring_type = 1
                if ring_size == 5:
                        five_ring_type = 1
                        print("found 5")
                if ring_size == 6:
                        six_ring_type = 1
                if ring_size >= 7:
                        ge_seven_ring_type = 1
                else:
                        ring_type = 'multi-membered ring'
     # Loop through the rings and check if they are saturated, unsaturated, or aromatic
                aromatic = all([m.GetAtomWithIdx(atom).GetIsAromatic() for atom in ring_atoms])
                three_mem_aro=0
                four_mem_aro=0
                five_mem_aro=0
                six_mem_aro=0
```

```
                ge_seven_aro=0
                if aromatic:
                        if three_ring_type == 1:
                                fp_list[3] = '1'
                                three_mem_aro=1
                                print("The 3 membered ring is aromatic")
                        if four_ring_type == 1:
                                fp_list[5] = '1'
                                four_mem_aro=1
                                print("The 4 membered ring is aromatic")
                        if five_ring_type == 1:
                                fp_list[8] = '1'
                                five_mem_aro=1
                                print("The 5 membered ring is aromatic")
                        elif six_ring_type == 1:
                                fp_list[14] = '1'
                                six_mem_aro=1
                                print("The 6 membered ring is aromatic")
                        elif ge_seven_ring_type == 1:
                                fp_list[18] = '1'
                                ge_seven_aro=1
                                print("The macro membered ring is aromatic")
                num_single_bonds = 0
                num_double_bonds = 0
                for j in range(ring_size):
                        bond = m.GetBondBetweenAtoms(ring_atoms[j], ring_atoms[(j + 1) %
ring_size])
                        if bond.GetBondType() == Chem.BondType.SINGLE:
                                num_single_bonds += 1
                        if bond.GetBondType() == Chem.BondType.DOUBLE:
                                num_double_bonds += 1
                if num_single_bonds >= 1:
                        if three_ring_type == 1:
                                fp_list[2] = '1'
                        if four_ring_type == 1:
                                fp_list[4] = '1'
                        if five_ring_type == 1:
                                fp_list[6] = '1'
                        if six_ring_type == 1:
                                fp_list[12] = '1'
                        if ge_seven_ring_type == 1:
                                fp_list[17] = '1'
                if num_double_bonds == 1:
                        if three_ring_type == 1:
                                fp_list[3] = '1'
                        if four_ring_type == 1:
                                fp_list[5] = '1'
                        if five_ring_type == 1:
                                fp_list[7] = '1'
                        if six_ring_type == 1:
                                fp_list[13] = '1'
```

```
                        if ge_seven_ring_type == 1:
                                fp_list[18] = '1'
                if num_double_bonds ==2 and six_ring_type==1:
                                fp_list[13]='1'
                if num_double_bonds >= 2 and ring_size >= 7:
                        if ge_seven_ring_type==1:
                                fp_list[18]='1'

#Check if none of the BITS are 1
#       check_2to13="".join([e for e in fp_list[1:12]])
#       if check_2to13.find('1')==-1:
#               Three_mem_sat_ring=pybel.Smarts("[R]1[R][R]1")
#               Three_mem_sat_ring2=pybel.Smarts("[#6;r3]!=[#6;r3]")
        Three_mem_sat_ring=pybel.Smarts("[CH2]1[CH2][CH2]1")
        Three_mem_sat_ring2=pybel.Smarts("[!a;R;X4]1@[!a;R;X4]@[!a;R;X4]1")
        Three_mem_sat_ring3=pybel.Smarts("[!a;#6;r3]=[!a;#6;r3]")
        Four_mem_sat_ring=pybel.Smarts("[CH2]1[CH2][CH2][CH2]1")
        Four_mem_sat_ring2=pybel.Smarts("[!a;R;X4]1@[!a;R;X4]@[!a;R;X4]@[!a;R;X4]1")
        Four_mem_sat_ring3=pybel.Smarts("[!a;#6;r4]=[!a;#6;r4]")
#               Four_mem_sat_ring=pybel.Smarts("[R]1[R][R][R]1")
#               Four_mem_sat_ring2=pybel.Smarts("[#6;r4]!=[#6;r4][r4!a]")
#               Five_mem_sat_ring=pybel.Smarts("[A;!a;R1]1[A;!a;R1][A;!a;R1][A;!a;R1][A;!a;R1]1")
        Five_mem_sat_ring=pybel.Smarts("[CH2]1[CH2][CH2][CH2][CH2]1")
        Five_mem_sat_ring2=pybel.Smarts("[!a;R;X4]1@[!a;R;X4]@[!a;R;X4]@[!a;R;X4]@[!a;R;X4]1"
)
        Five_mem_sat_ring3=pybel.Smarts("[!a;#6;r5]=[!a;#6;r5]")
        Six_mem_sat_ring=pybel.Smarts("[CH2]1[CH2][CH2][CH2][CH2][CH2]1")
        Six_mem_sat_ring2=pybel.Smarts("[!a;R;X4]1@[!a;R;X4]@[!a;R;X4]@[!a;R;X4]@[!a;R;X4]@[!
a;R;X4]1")
        Six_mem_sat_ring3=pybel.Smarts("[!a;#6;r6]=[!a;#6;r6]")
#
        Six_mem_sat_ring=pybel.Smarts("[A;!a;R1]1!=[A;!a;R1][A;!a;R1][A;!a;R1][A;!a;R1][A;!a;R1]1"
)
#               Six_mem_sat_ring2=pybel.Smarts("[#6;r6]!=[#6;r6]")
        macro_mem_sat_ring=pybel.Smarts("[r;!r3;!r4;!r5;!r6][#6]!=[#6]")
        Three_mem_unsat_ring=pybel.Smarts("[r3][A;r3]=,#[A;r3]")
        Four_mem_unsat_ring=pybel.Smarts("[r4][A;r4]=,#[A;r4]")
        Five_mem_unsat_ring=pybel.Smarts("[r5][A;r5]=,#[A;r5]")
        Six_mem_unsat_ring=pybel.Smarts("[r6][A;r6]=,#[A;r6]")
        macro_mem_unsat_ring=pybel.Smarts("[r;!r3;!r4;!r5;!r6][A;R]=,#[A;R]")
        Five_mem_arom_ring1=pybel.Smarts("a:1:a:a:a:a:1")
        Six_mem_arom_ring1=pybel.Smarts("a:1:a:a:a:a:a:1")
        Four_mem_arom_ring=pybel.Smarts("[r4&a]")
        Five_mem_arom_ring2=pybel.Smarts("[r5&a]")
        Six_mem_arom_ring2=pybel.Smarts("[r6&a]")
        macro_mem_arom_ring=pybel.Smarts("[r;!r3;!r4;!r5;!r6;a]")
        three_sat_res=Three_mem_sat_ring.findall(mymol)
        four_sat_res=Four_mem_sat_ring.findall(mymol)
#               five_sat_res=Five_mem_sat_ring.findall(mymol)
        six_sat_res=Six_mem_sat_ring.findall(mymol)
        macro_sat_res=macro_mem_sat_ring.findall(mymol)
```

```
        three_sat_res2=Three_mem_sat_ring2.findall(mymol)
        three_sat_res3=Three_mem_sat_ring3.findall(mymol)
        four_sat_res2=Four_mem_sat_ring2.findall(mymol)
        four_sat_res3=Four_mem_sat_ring3.findall(mymol)
        five_sat_res=Five_mem_sat_ring.findall(mymol)
        five_sat_res2=Five_mem_sat_ring2.findall(mymol)
        five_sat_res3=Five_mem_sat_ring3.findall(mymol)
        six_sat_res2=Six_mem_sat_ring2.findall(mymol)
        six_sat_res3=Six_mem_sat_ring3.findall(mymol)
        three_unsat_res=Three_mem_unsat_ring.findall(mymol)
        four_unsat_res=Four_mem_unsat_ring.findall(mymol)
        five_unsat_res=Five_mem_unsat_ring.findall(mymol)
        six_unsat_res=Six_mem_unsat_ring.findall(mymol)
        macro_unsat_res=macro_mem_unsat_ring.findall(mymol)
        four_arom_res=Four_mem_arom_ring.findall(mymol)
        five_arom_res1=Five_mem_arom_ring1.findall(mymol)
        six_arom_res1=Six_mem_arom_ring1.findall(mymol)
        macro_arom_res=macro_mem_arom_ring.findall(mymol)
        five_arom_res2=Five_mem_arom_ring2.findall(mymol)
        six_arom_res2=Six_mem_arom_ring2.findall(mymol)
#Class3
        if (len(three_sat_res)>=1):
                fp_list[2]='1'
        if (len(three_sat_res2)>=1):
                fp_list[2]='1'
#Class 4
        if (len(three_unsat_res)>=1):
                fp_list[3]='1'
#Class 5
        if (len(four_sat_res)>=1):
                fp_list[4]='1'
        if (len(four_sat_res2)>=1):
                fp_list[4]='1'
#Class 6
        if (len(four_unsat_res)>=1):
                fp_list[5]='1'
#Class 7
        if (len(five_sat_res)>=1):
                fp_list[6]='1'
        if len(five_sat_res2)>=1:
                fp_list[6]='1'
#Class 8
        if (len(five_unsat_res)>=1):
                fp_list[7]='1'
#Class 9
        if len(five_arom_res1)>=1:
                fp_list[8]='1'
        if len(five_arom_res2)==5:
                fp_list[8]='1'
#Class 13
        if (len(six_sat_res)>=1):
```

```
                fp_list[12]='1'
        if (len(six_sat_res2)>=1):
                fp_list[12]='1'
#Class 14
        if (len(six_unsat_res)>=1):
                fp_list[13]='1'
#Class 15
        if len(six_arom_res1)>=1:
                fp_list[14]='1'
        if len(six_arom_res2)==6:
                fp_list[14]='1'
#Class 18
        if (len(macro_sat_res)>=1):
                fp_list[17]='1'
#Class 19
        if (len(macro_unsat_res)>=1):
                fp_list[18]='1'
#For macrocyclic-aromatic rings whose bit is class 41
        if len(macro_arom_res)>=1:
                fp_list[18]='1'
#Patterns for special classes
        benzene=pybel.Smarts("c1ccccc1")
        benzene_isolated=pybel.Smarts("[cR1]1[cR1][cR1][cR1][cR1][cR1]1")
        pyrrole=pybel.Smarts("c1ccnc1")
        pyrrole_isolated=pybel.Smarts("[cR1]1[cR1][cR1][nR1][cR1]1")
        pyridine=pybel.Smarts("c1ccncc1")
        pyridine_isolated=pybel.Smarts("[cR1]1[cR1][cR1][nR1][cR1][cR1]1")
        furan=pybel.Smarts("c1ccoc1")
        furan_isolated=pybel.Smarts("[cR1]1[cR1][cR1][oR1][cR1]1")
        thiophene=pybel.Smarts("c1ccsc1")
        thiophene_isolated=pybel.Smarts("[cR1]1[cR1][cR1][sR1][cR1]1")
        fused=pybel.Smarts("[R2]")
        conpat=pybel.Smarts("[R]!@[R]")
        ringcount=rc.rc(smiles)
        benzene_res=benzene.findall(mymol)
        pyrrole_res=pyrrole.findall(mymol)
        pyridine_res=pyridine.findall(mymol)
        furan_res=furan.findall(mymol)
        thiophene_res=thiophene.findall(mymol)
        fused_res=fused.findall(mymol)
        res_benzene_neighbour=benzene_isolated.findall(mymol)
        res_pyrrole_neighbour=pyrrole_isolated.findall(mymol)
        res_furan_neighbour=furan_isolated.findall(mymol)
        res_thiophene_neighbour=thiophene_isolated.findall(mymol)
        res_pyridine_neighbour=pyridine_isolated.findall(mymol)
        if len(fused_res)==0:
                if len(benzene_res)==1:
                        fp_list[15]='1'
                if len(pyrrole_res)==1:
                        fp_list[9]='1'
                if len(pyridine_res)==1:
```

```
                fp_list[16]='1'
        if len(furan_res)==1:
                fp_list[10]='1'
        if len(thiophene_res)==1:
                fp_list[11]='1'
if len(fused_res)!=0 and len(conpat_res)>=1:
        if len(res_benzene_neighbour)>=1:
                fp_list[15]='1'
        if len(res_pyrrole_neighbour)==1:
                fp_list[9]='1'
        if len(res_pyridine_neighbour)==1:
                fp_list[16]='1'
        if len(res_furan_neighbour)==1:
                fp_list[10]='1'
        if len(res_thiophene_neighbour)==1:
                fp_list[11]='1'
if (len(bis2)>0):
        if len(fused_res)==0:
                if len(benzene_res)==1:
                        fp_list[15]='1'
                if len(pyrrole_res)==1:
                        fp_list[9]='1'
                if len(pyridine_res)==1:
                        fp_list[16]='1'
                if len(furan_res)==1:
                        fp_list[10]='1'
                if len(thiophene_res)==1:
                        fp_list[11]='1'
if len(fused_res)!=0 and len(conpat_res)>=1:
        if len(res_benzene_neighbour)==1:
                fp_list[15]='1'
        if len(res_pyrrole_neighbour)==1:
                fp_list[9]='1'
        if len(res_pyridine_neighbour)==1:
                fp_list[16]='1'
        if len(res_furan_neighbour)==1:
                fp_list[10]='1'
        if len(res_thiophene_neighbour)==1:
                fp_list[11]='1'
if (len(bis)>0):
        print("entered bis loop")
        bs=[]
        labels=[]
        for bi in bis:
                b = m.GetBondBetweenAtoms(bi[0],bi[1])
                if b.GetBeginAtomIdx()==bi[0]:
                        labels.append((10,1))
                else:
                        labels.append((1,10))
                bs.append(b.GetIdx())
        nm = Chem.FragmentOnBonds(m,bs,dummyLabels=labels)
```

```
                frag = Chem.MolToSmiles(nm,True)
        #Fragmented frags
                fment=frag.split(".")
                nRings=[]
                for fragment in fment:
                        corrected_smiles=cs.remove_non_alphabetic(fragment)
                        largest=0
                        fragmentcount=rc.rc(corrected_smiles)
                        if fragmentcount > largest:
                        #       print("This is the fragmented frag",fragment)
#                               new_mol=Chem.MolFromSmiles(fragment)
                                fragment = re.sub(r'\[al.*?\]', '[Al]', fragment, re.IGNORECASE)
                                new_frag=pybel.readstring("smi",fragment)
#Compute for openbabel patterns
                                benzene_res=benzene.findall(new_frag)
                                pyrrole_res=pyrrole.findall(new_frag)
                                pyridine_res=pyridine.findall(new_frag)
                                furan_res=furan.findall(new_frag)
                                conpat_res=conpat.findall(new_frag)
                                thiophene_res=thiophene.findall(new_frag)
                                fused_res=fused.findall(new_frag)
                                res_benzene_neighbour=benzene_isolated.findall(new_frag)
                                res_pyrrole_neighbour=pyrrole_isolated.findall(new_frag)
                                res_furan_neighbour=furan_isolated.findall(new_frag)
                                res_thiophene_neighbour=thiophene_isolated.findall(new_frag)
                                res_pyridine_neighbour=pyridine_isolated.findall(new_frag)
                                if len(fused_res)==0 and len(conpat_res)>=0:
                                        if len(benzene_res)>=1:
                                                fp_list[15]='1'
                                        if len(pyrrole_res)>=1:
                                                fp_list[9]='1'
                                        if len(pyridine_res)>=1:
                                                fp_list[16]='1'
                                        if len(furan_res)>=1:
                                                fp_list[10]='1'
                                        if len(thiophene_res)>=1:
                                                fp_list[11]='1'
                                if len(fused_res)!=0 and len(conpat_res)>=1:
                                        if len(res_benzene_neighbour)>=1:
                                                fp_list[15]='1'
                                        if len(res_pyrrole_neighbour)==1:
                                                fp_list[9]='1'
                                        if len(res_pyridine_neighbour)==1:
                                                fp_list[16]='1'
                                        if len(res_furan_neighbour)==1:
                                                fp_list[10]='1'
                                        if len(res_thiophene_neighbour)==1:
                                                fp_list[11]='1'
        if len(bis2)==0:
                if len(fused_res)==0:
                        if len(benzene_res)==1:
```

```
                    fp_list[15]='1'
                if len(pyrrole_res)==1:
                    fp_list[9]='1'
                if len(pyridine_res)==1:
                    fp_list[16]='1'
                if len(furan_res)==1:
                    fp_list[10]='1'
                if len(thiophene_res)==1:
                    fp_list[11]='1'
            if len(fused_res)!=0 and len(conpat_res)>=1:
                if len(res_benzene_neighbour)==1:
                    fp_list[15]='1'
                if len(res_pyrrole_neighbour)==1:
                    fp_list[9]='1'
                if len(res_pyridine_neighbour)==1:
                    fp_list[16]='1'
                if len(res_furan_neighbour)==1:
                    fp_list[10]='1'
                if len(res_thiophene_neighbour)==1:
                    fp_list[11]='1'
    return fp_list
```

4. bi_cl21_to37.py: To check the molecules belongs to which class from class 21 to class 37 (If it is bicyclic group) based on ring count and number of carbon atoms in each ring.

##

MPDS-CL: 4. bi_cl21_to37.py

DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA

##

```
import re,sys,os,getopt,openbabel
from openbabel import pybel
import rdkit as rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
import checknature as ck
import ringcount as rc
import cleansmiles as cs
fused=pybel.Smarts("[R2]")
conpat=pybel.Smarts("[R]!@[R]")
three_and_others=pybel.Smarts("[r3][R2][r]")
four_and_others=pybel.Smarts("[r4][R2][r]")
Five_mem_nonarom_ring1=pybel.Smarts("[A;!a;R1]1[A;!a;R1][A;!a;R1][A;!a;R1][A;!a;R1]1")
Five_mem_nonarom_ring2=pybel.Smarts("[r5&!a]")
Six_mem_nonarom_ring1=pybel.Smarts("[A;!a;R1]1[A;!a;R1][A;!a;R1][A;!a;R1][A;!a;R1][A;!a;R1]1")
Six_mem_nonarom_ring2=pybel.Smarts("[r6&!a]")
Five_mem_arom_ring1=pybel.Smarts("a:1:a:a:a:a:1")
Five_mem_arom_ring2=pybel.Smarts("[r5&a]")
Six_mem_arom_ring=pybel.Smarts("a:1:a:a:a:a:a:1")
Six_mem_arom_ring2=pybel.Smarts("[r6&a]")
Six_mem_ring2=pybel.Smarts("[r6]")
macro_mem_ring=pybel.Smarts("[r;!r3;!r4;!r5;!r6]")
macro_mem_nonarom_ring=pybel.Smarts("[r;!r3;!r4;!r5;!r6;!a]")
macro_mem_arom_ring=pybel.Smarts("[r;!r3;!r4;!r5;!r6;a]")
geseven_and_geseven=pybel.Smarts("[r;!r3!r4!r5!r6][R2][r;!r3!r4!r5!r6]")
indole=pybel.Smarts("c1cc2ccccc2n1")
def checkbi(bis,bis2,o1,smiles,fp_list,mymol,m):
        ringcount=rc.rc(smiles)
        mymol=pybel.readstring("smi",smiles)
        res_fused=fused.findall(mymol)
        res_conpat=conpat.findall(mymol)
        res_three_and_others=three_and_others.findall(mymol)
        res_four_and_others=four_and_others.findall(mymol)
        res_five_mem_nonarom_ring1=Five_mem_nonarom_ring1.findall(mymol)
        res_five_mem_nonarom_ring2=Five_mem_nonarom_ring2.findall(mymol)
        res_six_mem_nonarom_ring1=Six_mem_nonarom_ring1.findall(mymol)
        res_six_mem_nonarom_ring2=Six_mem_nonarom_ring2.findall(mymol)
        res_five_mem_arom_ring1=Five_mem_arom_ring1.findall(mymol)
```

```
        res_five_mem_arom_ring2=Five_mem_arom_ring2.findall(mymol)
        res_six_mem_arom_ring1=Six_mem_arom_ring.findall(mymol)
        res_six_mem_arom_ring2=Six_mem_arom_ring2.findall(mymol)
        res_six_mem_ring2=Six_mem_ring2.findall(mymol)
        res_macro_mem_ring=macro_mem_ring.findall(mymol)
        res_macro_mem_nonarom_ring=macro_mem_nonarom_ring.findall(mymol)
        res_macro_mem_arom_ring=macro_mem_arom_ring.findall(mymol)
        res_geseven_and_geseven=geseven_and_geseven.findall(mymol)
        res_indole=indole.findall(mymol)
        three_and_others_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFrom
Smarts("[r3][R2][r;!r3]"))
        four_and_others_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromS
marts("[r4][R2][r;!r3!r4]"))
        five_mem_nonarom_ring2_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.
MolFromSmarts("[r5&!a]"))
        six_mem_nonarom_ring2_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.M
olFromSmarts("[r6&!a]"))
        five_mem_arom_ring2_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolF
romSmarts("[r5&a]"))
        six_mem_arom_ring2_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFr
omSmarts("[r6&a]"))
        six_mem_ring2_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSm
arts("[r6]"))
        macro_mem_ring_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromS
marts("[r;!r3;!r4;!r5;!r6]"))
        macro_mem_nonarom_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.Mol
FromSmarts("[r;!r3;!r4;!r5;!r6;!a]"))
        macro_mem_arom_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFro
mSmarts("[r;!r3;!r4;!r5;!r6;a]"))
        geseven_and_geseven_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolF
romSmarts("[r;!r3;!r4;!r5;!r6][R2][r;!r3;!r4;!r5;!r6]"))
        fused_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSmarts("[R2]
"))
        conpat_rdk=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSmarts("[R]
!@[R]"))
        if ringcount==2:
        #Class21
                if len(res_conpat)==1:
                        fp_list[20]='1'
        #Class22
                if len(res_three_and_others)>=1 and len(res_fused)>=1:
                        fp_list[21]='1'
        #Class23
                if len(res_four_and_others)>=1 and len(res_fused)>=2:
                        fp_list[22]='1'
        # #class24
                if len(res_five_mem_arom_ring2)==8 and len(res_fused)>=2:
                        fp_list[23]='1'
        #class25
                if len(res_five_mem_arom_ring2)==5 and len(res_five_mem_nonarom_ring2)==3
and len(res_fused)>=2:
```

```
                        fp_list[24]='1'
        #class26
                if len(res_five_mem_arom_ring2)==5 and len(res_six_mem_arom_ring2)==6 and
len(res_fused)>=2:
                        fp_list[25]='1'
        #class27
                if len(res_indole)>=1:
                        fp_list[26]='1'
        #class28
                if len(res_five_mem_arom_ring2)==5 and len(res_six_mem_nonarom_ring2)==4 and
len(res_fused)>=2:
                        fp_list[27]='1'
        #class29
                if len(res_five_mem_nonarom_ring2)==3 and len(res_six_mem_arom_ring2)==6 and
len(res_fused)>=2:
                        fp_list[28]='1'
        #class30
                if len(res_five_mem_nonarom_ring2)==8 and len(res_fused)>=2:
                        fp_list[29]='1'
                if len(res_five_mem_nonarom_ring2)==7 and len(res_fused)>=2:
                        fp_list[29]='1'
                if len(res_five_mem_nonarom_ring2)==5 and len(res_six_mem_nonarom_ring2)==6
and len(res_fused)>=2:
                        fp_list[29]='1'
                if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=7 and len(res_fused)>=2:
                        fp_list[29]='1'
                if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=5 and len(res_fused)>=2:
                        fp_list[29]='1'
        #Class31
                if len(res_five_mem_arom_ring2)==5 and len(res_macro_mem_arom_ring)>=5 and
len(res_fused)>=2:
                        fp_list[30]='1'
                if len(res_five_mem_arom_ring2)==5 and len(res_macro_mem_nonarom_ring)>=5
and len(res_fused)>=2:
                        fp_list[30]='1'
        #class32
                if len(res_six_mem_arom_ring2)==10 and len(res_fused)>=2:
                        fp_list[31]='1'
        #class33
                if len(res_six_mem_arom_ring2)==6 and len(res_six_mem_nonarom_ring2)==4 and
len(res_fused)>=2:
                        fp_list[32]='1'
        #class34
                if len(res_six_mem_nonarom_ring2)==10 and len(res_fused)>=2:
                        fp_list[33]='1'
                if len(res_six_mem_nonarom_ring2)==9 and len(res_fused)>=2:
                        fp_list[33]='1'
                if len(res_six_mem_nonarom_ring2)==8 and len(res_fused)>=2:
                        fp_list[33]='1'
```

```
        #class35
                if len(res_six_mem_ring2)>=1 and len(res_macro_mem_ring)>=1 and
len(res_fused)>=2:
                        fp_list[34]='1'
        #class36
                if len(res_geseven_and_geseven)>=2 and len(res_fused)>=2:
                        fp_list[35]='1'
        #class37
                if len(res_fused)==1:
                        fp_list[36]='1'

        elif (len(bis2)>0) and len(bis)==0:
                        if ringcount==2:
                        #Class21
                                if len(res_conpat)==1:
                                        fp_list[20]='1'
                        #Class22
                                if len(res_three_and_others)>=1 and len(res_fused)>=1:
                                        fp_list[21]='1'
                        #Class23
                                if len(res_four_and_others)>=1 and len(res_fused)>=2:
                                        fp_list[22]='1'
                        #Class24
                                if len(res_five_mem_arom_ring2)==8 and len(res_fused)>=2:
                                        fp_list[23]='1'
                        #class25
                                if len(res_five_mem_arom_ring2)==5 and
len(res_five_mem_nonarom_ring2)==3 and len(res_fused)>=2:
                                        fp_list[24]='1'
                        #class26
                                if len(res_five_mem_arom_ring2)==5 and
len(res_six_mem_arom_ring2)==6 and len(res_fused)>=2:
                                        fp_list[25]='1'
                        #class27
                                if len(res_indole)>=1:
                                        fp_list[26]='1'
                        #class28
                                if len(res_five_mem_arom_ring2)==5 and
len(res_six_mem_nonarom_ring2)==4 and len(res_fused)>=2:
                                        fp_list[27]='1'
                        #class29
                                if len(res_five_mem_nonarom_ring2)==3 and
len(res_six_mem_arom_ring2)==6 and len(res_fused)>=2:
                                        fp_list[28]='1'
                        #class30
                                if len(res_five_mem_nonarom_ring2)==8 and len(res_fused)>=2:
                                        fp_list[29]='1'
                                if len(res_five_mem_nonarom_ring2)==7 and len(res_fused)>=2:
                                        fp_list[29]='1'
                                if len(res_five_mem_nonarom_ring2)==5 and
len(res_six_mem_nonarom_ring2)==6 and len(res_fused)>=2:
```

```
                                        fp_list[29]='1'
                                if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=7 and len(res_fused)>=2:
                                        fp_list[29]='1'
                                if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=5 and len(res_fused)>=2:
                                        fp_list[29]='1'
                        #Class31
                                if len(res_five_mem_arom_ring2)==5 and
len(res_macro_mem_arom_ring)>=5 and len(res_fused)>=2:
                                        fp_list[30]='1'
                                if len(res_five_mem_arom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=5 and len(res_fused)>=2:
                                        fp_list[30]='1'
                        #class32
                                if len(res_six_mem_arom_ring2)==10 and len(res_fused)>=2:
                                        fp_list[31]='1'
                        #class33
                                if len(res_six_mem_arom_ring2)==6 and
len(res_six_mem_nonarom_ring2)==4 and len(res_fused)>=2:
                                        fp_list[32]='1'
                        #class34
                                if len(res_six_mem_nonarom_ring2)==10 and len(res_fused)>=2:
                                        fp_list[33]='1'
                                if len(res_six_mem_nonarom_ring2)==9 and len(res_fused)>=2:
                                        fp_list[33]='1'
                                if len(res_six_mem_nonarom_ring2)==8 and len(res_fused)>=2:
                                        fp_list[33]='1'
                        #class35
                                if len(res_six_mem_ring2)>=1 and len(res_macro_mem_ring)>=1
and len(res_fused)>=2:
                                        fp_list[34]='1'
                        #class36
                                if len(res_geseven_and_geseven)>=2 and len(res_fused)>=2:
                                        fp_list[35]='1'
                        #class37
                                if len(res_fused)==1:
                                        fp_list[36]='1'

        elif (len(bis)>0):
                bs=[]
                labels=[]
                for bi in bis:
                        b = m.GetBondBetweenAtoms(bi[0],bi[1])
                        if b.GetBeginAtomIdx()==bi[0]:
                                labels.append((10,1))
                        else:
                                labels.append((1,10))
                        bs.append(b.GetIdx())
                nm = Chem.FragmentOnBonds(m,bs,dummyLabels=labels)
                frag = Chem.MolToSmiles(nm,True)
```

```
        #Fragmented frags
                fment=frag.split(".")
                nRings=[]
                for fragment in fment:
                #       noise.write("This is fragmented frag:"+str(fragment+"\n"))
                        corrected_smiles=cs.remove_non_alphabetic(fragment)
                #       print(corrected_smiles)
                #       noise.write("This is corrected frag:"+str(corrected_smiles+"\n"))
                        largest=0
                        fragmentcount=rc.rc(corrected_smiles)
                        if fragmentcount > largest:
#                               if "[al]" in fragment:
                                fragment = re.sub(r'\[al.*?\]', '[Al]', fragment, re.IGNORECASE)
                                new_frag = pybel.readstring("smi",fragment)
#                               new_mol=Chem.MolFromSmiles(fragment)
                                res_fused=fused.findall(new_frag)
                                res_conpat=conpat.findall(new_frag)
                                res_three_and_others=three_and_others.findall(new_frag)
                                res_four_and_others=four_and_others.findall(new_frag)

        res_five_mem_nonarom_ring1=Five_mem_nonarom_ring1.findall(new_frag)

        res_five_mem_nonarom_ring2=Five_mem_nonarom_ring2.findall(new_frag)

        res_six_mem_nonarom_ring1=Six_mem_nonarom_ring1.findall(new_frag)

        res_six_mem_nonarom_ring2=Six_mem_nonarom_ring2.findall(new_frag)

        res_five_mem_arom_ring1=Five_mem_arom_ring1.findall(new_frag)

        res_five_mem_arom_ring2=Five_mem_arom_ring2.findall(new_frag)
                                res_six_mem_arom_ring1=Six_mem_arom_ring.findall(new_frag)
                                res_six_mem_arom_ring2=Six_mem_arom_ring2.findall(new_frag)
                                res_six_mem_ring2=Six_mem_ring2.findall(new_frag)
                                res_macro_mem_ring=macro_mem_ring.findall(new_frag)

        res_macro_mem_nonarom_ring=macro_mem_nonarom_ring.findall(new_frag)

        res_macro_mem_arom_ring=macro_mem_arom_ring.findall(new_frag)
                                res_geseven_and_geseven=geseven_and_geseven.findall(new_frag)
                                res_indole=indole.findall(new_frag)
                                if fragmentcount==2:
                                        if len(res_conpat)==1:
                                                fp_list[20]='1'
                                #Class22
                                        if len(res_three_and_others)>=1 and len(res_fused)>=1:
                                                fp_list[21]='1'
                                #Class23
                                        if len(res_four_and_others)>=1 and len(res_fused)>=2:
                                                fp_list[22]='1'
                                #Class24
```

```
                                        if len(res_five_mem_arom_ring2)==8 and
len(res_fused)>=2:
                                                fp_list[23]='1'
                                #class25
                                        if len(res_five_mem_arom_ring2)==5 and
len(res_five_mem_nonarom_ring2)==3 and len(res_fused)>=2:
                                                fp_list[24]='1'
                                #class26
                                        if len(res_five_mem_arom_ring2)==5 and
len(res_six_mem_arom_ring2)==6 and len(res_fused)>=2:
                                                fp_list[25]='1'
                                #class27
                                        if len(res_indole)>=1:
                                                fp_list[26]='1'
                                #class28
                                        if len(res_five_mem_arom_ring2)==5 and
len(res_six_mem_nonarom_ring2)==4 and len(res_fused)>=2:
                                                fp_list[27]='1'
                                #class29
                                        if len(res_five_mem_nonarom_ring2)==3 and
len(res_six_mem_arom_ring2)==6 and len(res_fused)>=2:
                                                fp_list[28]='1'
                                #class30
                                        if len(res_five_mem_nonarom_ring2)==8 and
len(res_fused)>=2:
                                                fp_list[29]='1'
                                        if len(res_five_mem_nonarom_ring2)==7 and
len(res_fused)>=2:
                                                fp_list[29]='1'
                                        if len(res_five_mem_nonarom_ring2)==5 and
len(res_six_mem_nonarom_ring2)==6 and len(res_fused)>=2:
                                                fp_list[29]='1'
                                        if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=7 and len(res_fused)>=2:
                                                fp_list[29]='1'
                                        if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=5 and len(res_fused)>=2:
                                                fp_list[29]='1'
                                #Class31
                                        if len(res_five_mem_arom_ring2)==5 and
len(res_macro_mem_arom_ring)>=5 and len(res_fused)>=2:
                                                fp_list[30]='1'
                                        if len(res_five_mem_arom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=5 and len(res_fused)>=2:
                                                fp_list[30]='1'
                                #class32
                                        if len(res_six_mem_arom_ring2)==10 and
len(res_fused)>=2:
                                                fp_list[31]='1'
                                #class33
```

```
                                        if len(res_six_mem_arom_ring2)==6 and
len(res_six_mem_nonarom_ring2)==4 and len(res_fused)>=2:
                                                fp_list[32]='1'
                                #class34
                                        if len(res_six_mem_nonarom_ring2)==10 and
len(res_fused)>=2:
                                                fp_list[33]='1'
                                        if len(res_six_mem_nonarom_ring2)==9 and
len(res_fused)>=2:
                                                fp_list[33]='1'
                                        if len(res_six_mem_nonarom_ring2)==8 and
len(res_fused)>=2:
                                                fp_list[33]='1'
                                #class35
                                        if len(res_six_mem_ring2)>=1 and
len(res_macro_mem_ring)>=1 and len(res_fused)>=2:
                                                fp_list[34]='1'
                                #class36
                                        if len(res_geseven_and_geseven)>=2 and len(res_fused)>=2:
                                                fp_list[35]='1'
                                #class37
                                        if len(res_fused)==1:
                                                fp_list[36]='1'

        elif len(bis2)==0:
                if ringcount==2:
                        if len(res_conpat)==1:
                                fp_list[20]='1'
                #Class22
                        if len(res_three_and_others)>=1 and len(res_fused)>=1:
                                fp_list[21]='1'
                #Class23
                        if len(res_four_and_others)>=1 and len(res_fused)>=2:
                                fp_list[22]='1'
                #Class24
                        if len(res_five_mem_arom_ring2)==8 and len(res_fused)>=2:
                                fp_list[23]='1'
                #class25
                        if len(res_five_mem_arom_ring2)==5 and
len(res_five_mem_nonarom_ring2)==3 and len(res_fused)>=2:
                                fp_list[24]='1'
                #class26
                        if len(res_five_mem_arom_ring2)==5 and len(res_six_mem_arom_ring2)==6
and len(res_fused)>=2:
                                fp_list[25]='1'
                #class27
                        if len(res_indole)>=1:
                                fp_list[26]='1'
                #class28
                        if len(res_five_mem_arom_ring2)==5 and
len(res_six_mem_nonarom_ring2)==4 and len(res_fused)>=2:
```

```
                    fp_list[27]='1'
            #class29
                    if len(res_five_mem_nonarom_ring2)==3 and
len(res_six_mem_arom_ring2)==6 and len(res_fused)>=2:
                    fp_list[28]='1'
            #class30
                    if len(res_five_mem_nonarom_ring2)==8 and len(res_fused)>=2:
                            fp_list[29]='1'
                    if len(res_five_mem_nonarom_ring2)==7 and len(res_fused)>=2:
                            fp_list[29]='1'
                    if len(res_five_mem_nonarom_ring2)==5 and
len(res_six_mem_nonarom_ring2)==6 and len(res_fused)>=2:
                            fp_list[29]='1'
                    if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=7 and len(res_fused)>=2:
                            fp_list[29]='1'
                    if len(res_five_mem_nonarom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=5 and len(res_fused)>=2:
                            fp_list[29]='1'
            #Class31
                    if len(res_five_mem_arom_ring2)==5 and
len(res_macro_mem_arom_ring)>=5 and len(res_fused)>=2:
                            fp_list[30]='1'
                    if len(res_five_mem_arom_ring2)==5 and
len(res_macro_mem_nonarom_ring)>=5 and len(res_fused)>=2:
                            fp_list[30]='1'
            #class32
                    if len(res_six_mem_arom_ring2)==10 and len(res_fused)>=2:
                            fp_list[31]='1'
            #class33
                    if len(res_six_mem_arom_ring2)==6 and
len(res_six_mem_nonarom_ring2)==4 and len(res_fused)>=2:
                            fp_list[32]='1'
            #class34
                    if len(res_six_mem_nonarom_ring2)==10 and len(res_fused)>=2:
                            fp_list[33]='1'
                    if len(res_six_mem_nonarom_ring2)==9 and len(res_fused)>=2:
                            fp_list[33]='1'
                    if len(res_six_mem_nonarom_ring2)==8 and len(res_fused)>=2:
                            fp_list[33]='1'
            #class35
                    if len(res_six_mem_ring2)>=1 and len(res_macro_mem_ring)>=1 and
len(res_fused)>=2:
                            fp_list[34]='1'
            #class36
                    if len(res_geseven_and_geseven)>=2 and len(res_fused)>=2:
                            fp_list[35]='1'
            #class37
                    if len(res_fused)==1:
                            fp_list[36]='1'
        return fp_list
```

5. tr_cl38_to46.py: To check the molecules belongs to which class from class 28 to class 46(if it is tricyclic group) based on ring count and number of carbon atoms in each ring. Along with this, checks the fused and connected rings

```
################################################################################
```

MPDS-CL: 5. tr_cl38_to46.py

DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA

```
################################################################################
```

```
import re,sys,os,getopt,openbabel
from openbabel import pybel
import rdkit as rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
import checknature as ck
import ringcount as rc
import cleansmiles as cs
import check_for_frag as ckf
import check_for_frag_rdk as ckfr
patt3=pybel.Smarts("[a;R]")
patt4=pybel.Smarts("[n,o,s,p;R]")
patt5=pybel.Smarts("[c]:[!c]")
patt6=pybel.Smarts("[R]=[R]")
patt7=pybel.Smarts("[R]#[R]")
patt8=pybel.Smarts("[A;R]")
fivear=pybel.Smarts("[r5&a]")
sixar=pybel.Smarts("[r6&a]")
def checktr(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,conpat,conpat_res):
        ringcount=rc.rc(smiles)
        mymol=pybel.readstring("smi",smiles)
        result3=patt3.findall(mymol)
        result4=patt4.findall(mymol)
        result5=patt5.findall(mymol)
        result6=patt6.findall(mymol)
        result7=patt7.findall(mymol)
        result8=patt8.findall(mymol)
        res_fivear=fivear.findall(mymol)
        res_sixar=sixar.findall(mymol)
        conpat=pybel.Smarts("[R]!@[R]")
        fusedpat=pybel.Smarts("[R2]")
#For tricyclic with connected rings
        if ringcount==3 and len(conpat_res)==2 and len(fusedpat_res)==0:
                pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
```

```
                if mixed_arom_one==1:
                        fp_list[37]='1'
                if mixed_arom_two==1:
                        fp_list[38]='1'
                if pure_arom==1:
                        fp_list[39]='1'
                if pure_alip==1:
                        fp_list[40]='1'
#For tricyclic fused and connected
        if ringcount==3 and len(fusedpat_res)==2 and len(conpat_res)==1:
                fp_list[45]='1'
        if (len(bis2)>0) and len(bis)==0:
                if ringcount==3 and len(conpat_res)==2 and len(fusedpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[37]='1'
                        if mixed_arom_two==1:
                                fp_list[38]='1'
                        if pure_arom==1:
                                fp_list[39]='1'
                        if pure_alip==1:
                                fp_list[40]='1'
                if ringcount==3 and len(fusedpat_res)==3 and len(conpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
                        if pure_arom==1:
                                fp_list[43]='1'
                        if pure_alip==1:
                                fp_list[44]='1'
                if ringcount==3 and len(fusedpat_res)==4 and len(conpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
                        if pure_arom==1:
                                fp_list[43]='1'
                        if pure_alip==1:
                                fp_list[44]='1'
                if ringcount==3 and len(fusedpat_res)==5 and len(conpat_res)==0:
```

```
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
                        if pure_arom==1:
                                fp_list[43]='1'
                        if pure_alip==1:
                                fp_list[44]='1'
        elif (len(bis)>0):
                bs=[]
                labels=[]
                for bi in bis:
                        b = m.GetBondBetweenAtoms(bi[0],bi[1])
                        if b.GetBeginAtomIdx()==bi[0]:
                                labels.append((10,1))
                        else:
                                labels.append((1,10))
                        bs.append(b.GetIdx())
                nm = Chem.FragmentOnBonds(m,bs,dummyLabels=labels)
                frag = Chem.MolToSmiles(nm,True)
#Fragmented frags
                fment=frag.split(".")
                nRings=[]
                for fragment in fment:
                #       noise.write("This is fragmented frag:"+str(fragment+"\n"))
                        corrected_smiles=cs.remove_non_alphabetic(fragment)
                #       noise.write("This is corrected frag:"+str(corrected_smiles+"\n"))
                        largest=0
                        fragmentcount=rc.rc(corrected_smiles)
#                       print("Fragmented smiles from tr_class", corrected_smiles)
                        if fragmentcount > largest:
                                new_mol=Chem.MolFromSmiles(corrected_smiles)
#                               if new_mol==None:
#                                       print("Error with the molecule")
#                               else:
#                               if "[al]" in fragment:
                                fragment = re.sub(r'\[al.*?\]', '[Al]', fragment, re.IGNORECASE)
                                new_frag=pybel.readstring("smi",fragment)
                                result3=patt3.findall(new_frag)
                                result4=patt4.findall(new_frag)
                                result5=patt5.findall(new_frag)
                                result6=patt6.findall(new_frag)
                                result7=patt7.findall(new_frag)
                                result8=patt8.findall(new_frag)
                                res_fivear=fivear.findall(new_frag)
                                res_sixar=sixar.findall(new_frag)
                                conpat_res=conpat.findall(new_frag)
                                fusedpat_res=fusedpat.findall(new_frag)
```

```
                    if fragmentcount==3 and len(conpat_res)==2 and
len(fusedpat_res)==0:

        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two=ckf.checknatureforfrag(m,correcte
d_smiles,result3,result4,result5,result6,result7,result8,
                        res_fivear,res_sixar,fusedpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[37]='1'
                        if mixed_arom_two==1:
                                fp_list[38]='1'
                        if pure_arom==1:
                                fp_list[39]='1'
                        if pure_alip==1:
                                fp_list[40]='1'
                    if fragmentcount==3 and len(fusedpat_res)==4 and
len(conpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,corrected_smiles,result3,result4,result5,result6,result7,result8,res_fivear,
                        res_sixar,fusedpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
                        if pure_arom==1:
                                fp_list[43]='1'
                        if pure_alip==1:
                                fp_list[44]='1'
                    if fragmentcount==3 and len(fusedpat_res)==3 and
len(conpat_res)==0:
                        print("Identified that the mol has 3 fusedatoms")
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,corrected_smiles,result3,result4,result5,result6,result7,result8,res_fivear,
                        res_sixar,fusedpat_res,conpat_res)
                        if mixed_arom_one==1:
                                print("One ring is aromatic")
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
                        if pure_arom==1:
                                fp_list[43]='1'
                        if pure_alip==1:
                                fp_list[44]='1'
                    if fragmentcount==3 and len(fusedpat_res)==5 and
len(conpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
```

```
                                        if pure_arom==1:
                                                fp_list[43]='1'
                                        if pure_alip==1:
                                                fp_list[44]='1'
                                if fragmentcount==3 and len(fusedpat_res)>=1 and
len(conpat_res)>=1:
                                        fp_list[45]='1'
        elif len(bis2)==0:
                result3=patt3.findall(mymol)
                result4=patt4.findall(mymol)
                result5=patt5.findall(mymol)
                result6=patt6.findall(mymol)
                result7=patt7.findall(mymol)
                result8=patt8.findall(mymol)
                res_fivear=fivear.findall(mymol)
                res_sixar=sixar.findall(mymol)
                conpat_res=conpat.findall(mymol)
                fusedpat_res=fusedpat.findall(mymol)
                if ringcount==3 and len(conpat_res)==2 and len(fusedpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[37]='1'
                        if mixed_arom_two==1:
                                fp_list[38]='1'
                        if pure_arom==1:
                                fp_list[39]='1'
                        if pure_alip==1:
                                fp_list[40]='1'
                if ringcount==3 and len(fusedpat_res)==3 and len(conpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
                        if pure_arom==1:
                                fp_list[43]='1'
                        if pure_alip==1:
                                fp_list[44]='1'
                if ringcount==3 and len(fusedpat_res)==5 and len(conpat_res)==0:
                        pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                        if mixed_arom_one==1:
                                fp_list[41]='1'
                        if mixed_arom_two==1:
                                fp_list[42]='1'
                        if pure_arom==1:
```

```
                                        fp_list[43]='1'
                                if pure_alip==1:
                                        fp_list[44]='1'
                        if ringcount==3 and len(fusedpat_res)==4:
                                pure_alip,pure_arom,mixed_arom_one,mixed_arom_two =
ckf.checknatureforfrag(m,smiles,result3,result4,result5,result6,result7,result8,res_fivear,res_sixar,fus
edpat_res,conpat_res)
                                if mixed_arom_one==1:
                                        fp_list[41]='1'
                                if mixed_arom_two==1:
                                        fp_list[42]='1'
                                if pure_arom==1:
                                        fp_list[43]='1'
                                if pure_alip==1:
                                        fp_list[44]='1'
                        if ringcount==3 and len(fusedpat_res)>=1 and len(conpat_res)>=1:
                                fp_list[45]='1'
#Adamantane to be classified in class 50
        if ringcount==3 and len(fusedpat_res)>=6:
                                fp_list[49]='1'
        else:
                pass
        return fp_list
```

6. tt_cl47_to50.py: To check the molecules belongs to which class from class 28 to class 46(if it is tetracyclic group) based on ring count and number of carbon atoms in each ring

```
################################################################################
```

MPDS-CL: 6. tt_cl47_to50.py

DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA

```
################################################################################
import re,sys,os,getopt,openbabel
from openbabel import pybel
import rdkit as rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
import checknature as ck
import ringcount as rc
import cleansmiles as cs
import check_for_frag as ckf
def checktt(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,conpat,conpat_res):
        ringcount=rc.rc(smiles)
        mymol=pybel.readstring("smi",smiles)
        conpat=pybel.Smarts("[R]!@[R]")
        fusedpat=pybel.Smarts("[R2]")
        rdk_conpat=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSmarts("[R]
!@[R]"))
        rdk_fusedpat=Chem.MolFromSmiles(smiles).GetSubstructMatches(Chem.MolFromSmarts("[
R2]"))

        if (len(bis)>0):
                bs=[]
                labels=[]
                for bi in bis:
                        b = m.GetBondBetweenAtoms(bi[0],bi[1])
                        if b.GetBeginAtomIdx()==bi[0]:
                                labels.append((10,1))
                        else:
                                labels.append((1,10))
                        bs.append(b.GetIdx())
                nm = Chem.FragmentOnBonds(m,bs,dummyLabels=labels)
                frag = Chem.MolToSmiles(nm,True)
#Fragmented frags
                fment=frag.split(".")
                nRings=[]
                for fragment in fment:
#                       noise.write("This is fragmented frag:"+str(fragment+"\n"))
                        corrected_smiles=cs.remove_non_alphabetic(fragment)
#                       noise.write("This is corrected frag:"+str(corrected_smiles+"\n"))
```

```
                        largest=0
                        fragmentcount=rc.rc(corrected_smiles)
#                       print("This is fragment count through second path of mono", fragmentcount)
                        if fragmentcount > largest:
                                if fragmentcount==4:
                                        new_mol=Chem.MolFromSmiles(corrected_smiles)
                        #               if new_mol==None:
                        #                       print("Error with the molecule")
                        #               else:
#                                       if "[al]" in fragment:
                                        fragment = re.sub(r'\[al.*?\]', '[Al]', fragment,
re.IGNORECASE)
                                        new_frag=pybel.readstring("smi",fragment)
                                        conpat_res=conpat.findall(new_frag)
                                        fusedpat_res=fusedpat.findall(new_frag)
                                #Class47 For all four connected rings
                                        if len(conpat_res)==3 and len(fusedpat_res)==0:
                                                fp_list[46]='1'
                                #Class48
                                        if len(fusedpat_res)<=6 and len(conpat_res)==0:
                                                fp_list[47]='1'
                                #Class49
                                        if len(fusedpat_res)>=5 and len(conpat_res)==1:
                                                fp_list[48]='1'
                                        if len(fusedpat_res)==4 and len(conpat_res)==1:
                                                fp_list[48]='1'
                                        if len(fusedpat_res)==4 and len(conpat_res)==2:
                                                fp_list[48]='1'
                                        if len(fusedpat_res)==3 and len(conpat_res)==1:
                                                fp_list[48]='1'
                                        if len(fusedpat_res)==2 and len(conpat_res)==2:
                                                fp_list[48]='1'
                                        if len(fusedpat_res)==3 and len(conpat_res)==2:
                                                fp_list[48]='1'
                                        if len(fusedpat_res)==1 and len(conpat_res)==2:
                                                fp_list[48]='1'
                                        if len(fusedpat_res)==2 and len(conpat_res)==1:
                                                fp_list[48]='1'
                                #Class50
                                        if len(fusedpat_res)>6:
                                                fp_list[49]='1'
                        #Class50 special for adamantane
                        if fragmentcount==3 and len(fusedpat_res)>=6:
                                        fp_list[49]='1'
        elif len(bis2)==0:
                conpat_res=conpat.findall(mymol)
                fusedpat_res=fusedpat.findall(mymol)
                if ringcount==4:
                #Class47 For all four connected rings
                        if len(conpat_res)==3 and len(fusedpat_res)==0:
                                fp_list[46]='1'
```

```
        #Class48
                if len(fusedpat_res)==4 and len(conpat_res)==0:
                        fp_list[47]='1'
                if len(fusedpat_res)==6 and len(conpat_res)==0:
                        fp_list[47]='1'
        #Class49
                if len(fusedpat_res)==4 and len(conpat_res)==1:
                        fp_list[48]='1'
                if len(fusedpat_res)==2 and len(conpat_res)==2:
                        fp_list[48]='1'
                if len(fusedpat_res)>=1 and len(conpat_res)>=1:
                        fp_list[48]='1'
        #Class50
                if len(fusedpat_res)>6:
                        fp_list[49]='1'
else:
        pass

return fp_list
```

7. multiple_cl20.py: To check the molecules belongs to class 20(If it is mono cyclic with multiple element group in a ring) based on ring count and number of carbon atoms in each ring

##

MPDS-CL: 7. multiple_cl20.py

DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA

##

```
import re,sys,os,getopt
from openbabel import pybel
import rdkit as rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import rdmolops
import checknature as ck
import ringcount as rc
import cleansmiles as cs
def checkspecial(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,conpat,conpat_res):
# Find all the rings in the molecule
	ring_atoms_list = rdmolops.GetSymmSSSR(m)
	# Count the occurrences of non-carbon atoms in each ring
	atom_count = {}
	for i, ring_atoms in enumerate(ring_atoms_list):
		for atom in ring_atoms:
			symbol = m.GetAtomWithIdx(atom).GetSymbol()
			if symbol != 'C':
				if symbol == 'Li' or 'Na' or 'K' or 'Rb' or 'Cs' or 'Fr' or 'Be' or 'Mg' or 'Ca' or 'Sr' or 'Ba' or 'Ra' or 'He' or 'B' or 'N' or 'O' or 'F' or 'Ne' or 'Al' or 'Si' or 'P' or 'S' or 'Cl' or 'Ar' or 'Ga' or 'Ge' or 'As' or 'Se' or 'Br' or 'Kr' or 'In' or 'Sn' or 'Sb' or 'Te' or 'I' or 'Xe' or 'Tl' or 'Pb' or 'Bi' or 'Po' or 'At' or 'Rn' or 'Nh' or 'Fl' or 'Mc' or 'Lv' or 'Ts' or 'Og':
#			if symbol != 'C' and symbol == 'Li' or 'Na' or 'K' or 'Rb' or 'Cs' or 'Fr' or 'Be' or 'Mg' or 'Ca' or 'Sr' or 'Ba' or 'Ra' or 'He' or 'B' or 'N' or 'O' or 'F' or 'Ne' or 'Al' or 'Si' or 'P' or 'S' or 'Cl' or 'Ar' or 'Ga' or 'Ge' or 'As' or 'Se' or 'Br' or 'Kr' or 'In' or 'Sn' or 'Sb' or 'Te' or 'I' or 'Xe' or 'Tl' or 'Pb' or 'Bi' or 'Po' or 'At' or 'Rn' or 'Nh' or 'Fl' or 'Mc' or 'Lv' or 'Ts' or 'Og':
					if symbol not in atom_count:
						atom_count[symbol] = [0] * len(ring_atoms_list)
					atom_count[symbol][i] += 1
	for i, ring_atoms in enumerate(ring_atoms_list):
		total_count = sum(atom_count[symbol][i] for symbol in atom_count)
		for symbol, counts in atom_count.items():
			count = counts[i]
			if total_count > 1:
				fp_list[19]='1'
	else:
		pass
	return fp_list
```

8. po_cl51.py: To check the molecules belongs to which class from class 51(if it is pentacyclic group) based on ring count and number of carbon atoms in each ring.

```
################################################################################
```

MPDS-CL: 8. po_cl51.py

DEVELOPED @ ACDSD, CSIR-NEIST, JORHAT-06, ASSAM, INDIA

```
################################################################################
import re,sys,os,getopt,openbabel
from openbabel import pybel
import rdkit as rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
import checknature as ck
import ringcount as rc
import cleansmiles as cs
import check_for_frag as ckf
r_linker_r=pybel.Smarts("[R][!R][R]")
#Patterns for mono classes
def checkpo(bis,bis2,o1,smiles,fp_list,mymol,m,fusedpat,fusedpat_res,conpat,conpat_res):
	ringcount=rc.rc(smiles)
	if (len(bis)>0):
		bs=[]
		labels=[]
		for bi in bis:
			b = m.GetBondBetweenAtoms(bi[0],bi[1])
			if b.GetBeginAtomIdx()==bi[0]:
				labels.append((10,1))
			else:
				labels.append((1,10))
			bs.append(b.GetIdx())
		nm = Chem.FragmentOnBonds(m,bs,dummyLabels=labels)
		frag = Chem.MolToSmiles(nm,True)
#Fragmented frags
		fment=frag.split(".")
		nRings=[]
		for fragment in fment:
			corrected_smiles=cs.remove_non_alphabetic(fragment)
			largest=0
			fragmentcount=rc.rc(corrected_smiles)
			if fragmentcount > largest:
				if fragmentcount>=5:
					fp_list[50]='1'
	elif len(bis2)==0:
		if ringcount>=5:
			fp_list[50]='1'
	elif ringcount>=5:
```

```
            fp_list[50]='1'
else:
            pass
return fp_list
```

Disclaimer

Copyright

The present web portal is a copyright of Molecular Property Diagnostic Suite (MPDS) hosted at Advanced Computation and Data Sciences (ACDS) Division, CSIR-NEIST, Jorhat, Assam, India. You are not permitted to copy all or parts of the portal's contents without the prior consent.

Unauthorised reproduction, storage, or dissemination of copyrighted documents, data or other associated content that falls under the umbrella of MPDS, whether in physical form such as CDROM or any other data carriers, or in virtual spaces like newsgroups or online con services, is strictly prohibited without explicit consent from the rightful copyright holder. The specific set of criteria encompasses all MPDS logos. The act of linking to this website and appropriately attributing any of its documents is unequivocally authorised.

In the event that you seek authorization from MPDS to replicate either the entirely or a segment of this website, kindly establish contact with the copyright proprietor.

Liability

The web portal will consistently provide up-to-date and precise information. We are committed to addressing any issues that are brought to our attention with due approval from the competent authority. We do not provide coverage for any loss or damage, including indirect or consequential loss or damage, as well as any loss or damage resulting from lost data or profits, that may arise from or be related to the use of this website, regardless of the circumstances. The MPDS shall not be held liable for any claims arising from the use or non-use of the website, the utilisation of published information, misuse of the connection, or any technical failures.

MPDS reserves the right to modify, delete, or temporarily suspend the release, distribution or release content at any moment with no prior notice.

Grant of Source Code

The content available on this portal is licenced under the Open Software Licence v. 3.0 (OSL-3.0), which includes both derived and original works. The term "Source Code" refers to the favoured form of the Original Work that enables modifications to be made, along with all accompanying documentation that provides instructions on how to modify the Original Work. The Licensor hereby commits to furnishing a machine-readable version of the Source Code for the Original Work, which shall be included with every distribution of the said Original Work by the Licensor. The Licensor retains the right to fulfil this obligation by storing a machine-readable version of the Source Code in an information repository that is designed to allow affordable and convenient access by a user, as long as the Licensor keeps releasing the Original Work.

August 2023