代谢反应分子网络的整体工作流程

1. **数据预处理（输入raw数据，输出mgf文件和Feature List）**

* 核心目的

利用成熟的数据处理软件MS-DIAL，输入raw数据，输出mgf文件和Feature List。

* 样本数据要求（空白、给药、药物（可选）最少各一个，n=3是推荐的），空白避免污染
* MS-DIAL处理参数
* 输出mgf文件时不能选择扣除空白，否则会报错、（F1-MSDIAL-MS2.mgf）
* Feature List文件格式统一化（Demo文件1）

FALSE MS/MS assigned项下，删除FALSE的行、将各组平均值前移，调整列名称

1. **Targeted Ions List生成**

* 将Feature List过滤生成Targeted Ions List，过滤方式选择：分布差异；



* 分布差异成分的定义：

1. 含药生物样本中大于响应阈值（依据仪器的类型而定50W）；
2. 含药样本平均响应/空白样本平均响应≥30；

由Demo文件1手动处理，获得Targeted Ions List的文件F2-Targeted Ions.csv

1. **In-house代谢反应库及用户自定义输入功能**

KEGG常见代谢反应，各个反应类型、常见加和离子

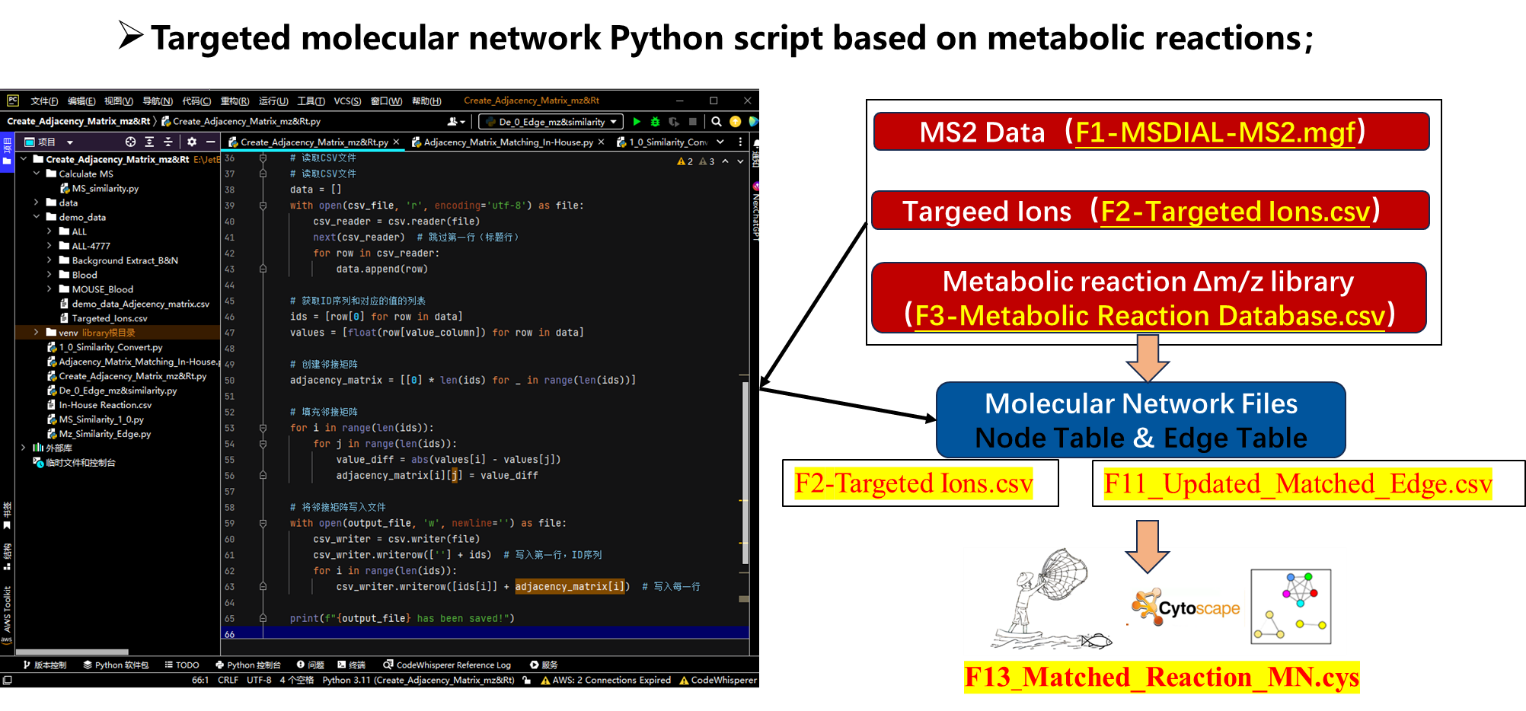
* 文件格式：ID, Δm/z, 分子式变化, 反应类型（几步反应）, 反应描述（内置文件1：F3-Metabolic Reaction Database.csv）
* 自定义输入功能：输入分子式变化, 自定义反应描述
* 更新F3-Metabolic Reaction Database.csv

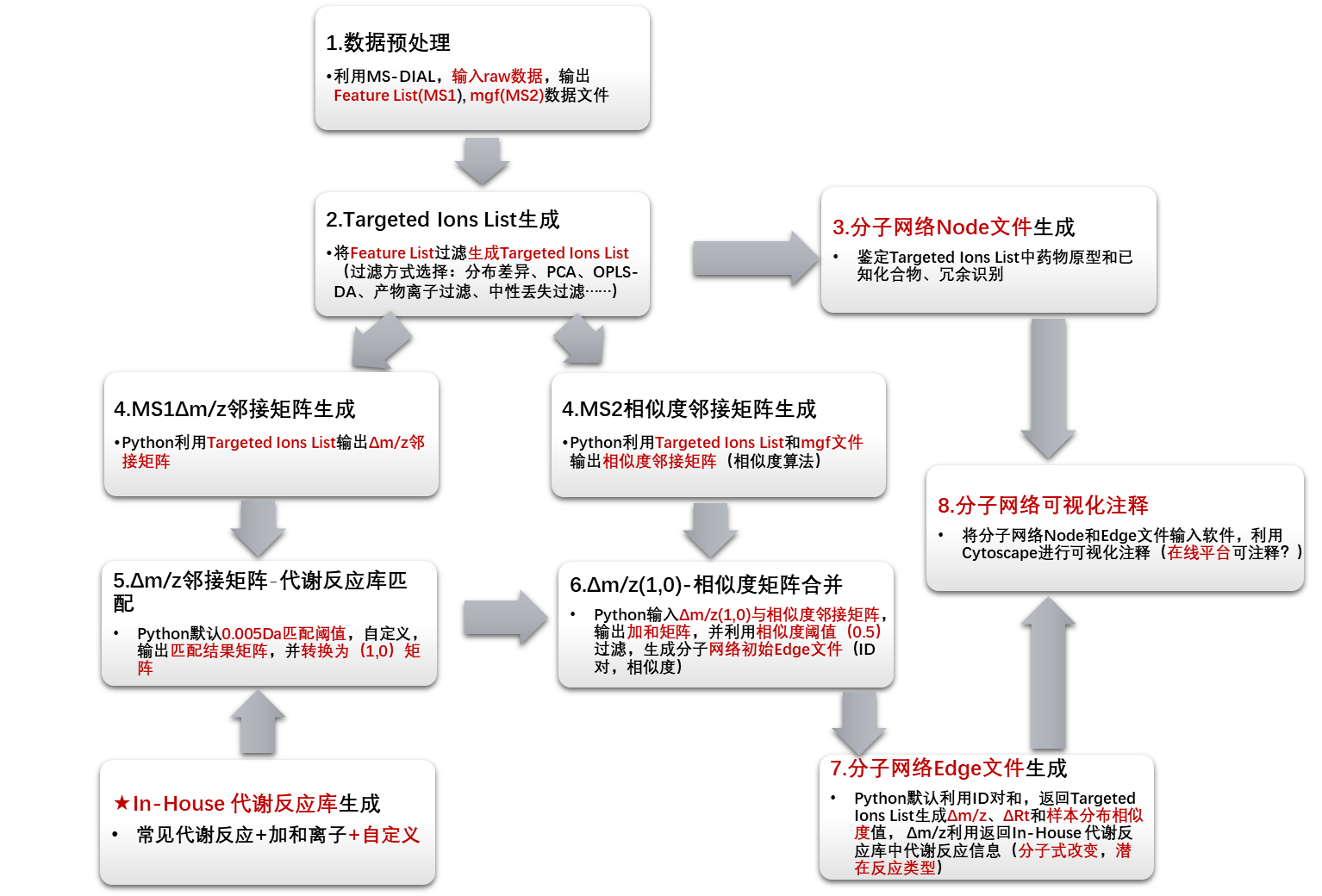
1. 提供一个输入窗口，供用户输入Formula Change的值（为分子式变化的分子式），和输入Formula Change值对应的Reaction Description（文本描述，限定30字符）；
2. 依据用户输入的Formula Change的分子式，利用pyteomics包计算对应的精确分子量（Mass Difference(Da)值）；
3. 将上述步骤中的输入的Formula Change、Reaction Description值和计算的Mass Difference(Da)值写入F3-Metabolic Reaction Database.csv，并保存。

Python脚本 《User\_Defined\_Reactions.py》

import pandas as pd  
import pyteomics.mass  
  
# 读取现有的Metabolic Reaction Database.csv文件  
def read\_database(filename):  
 try:  
 database = pd.read\_csv(filename)  
 return database  
 except FileNotFoundError:  
 print("Metabolic Reaction Database.csv not found. Creating a new database.")  
 return pd.DataFrame(columns=['ID'**,** 'Reaction Description'**,** 'Mass Difference(Da)'**,** 'Formula Change'])  
  
# 获取下一个可用的ID  
def get\_next\_id(database):  
 if database.empty:  
 return **1** else:  
 return database['ID'].max() + **1**# 提示用户输入新的Reaction信息  
def input\_reaction\_info():  
 formula\_change = input("Enter Formula Change (e.g., H2O): ")  
 reaction\_description = input("Enter Reaction Description (max 30 characters): ")[:**30**]  
 reaction\_description = "User\_" + reaction\_description  
 return formula\_change**,** reaction\_description  
  
  
# 计算Mass Difference(Da)值  
def calculate\_mass\_difference(formula\_change):  
 mass\_diff = pyteomics.mass.calculate\_mass(formula\_change)  
 return mass\_diff  
  
# 将新的Reaction写入数据库  
def add\_reaction\_to\_database(database**,** formula\_change**,** reaction\_description**,** mass\_difference):  
 next\_id = get\_next\_id(database)  
 new\_row = {'ID': next\_id**,** 'Reaction Description': reaction\_description**,** 'Mass Difference(Da)': mass\_difference**,** 'Formula Change': formula\_change}  
 database = pd.concat([database**,** pd.DataFrame([new\_row])]**,** ignore\_index=True)  
 return database  
  
# 保存更新后的数据库到CSV文件  
def save\_database(database**,** filename):  
 database.to\_csv(filename**,** index=False)  
 print("Database updated and saved.")  
  
def main():  
 database\_filename = "F3-Metabolic Reaction Database.csv"  
 database = read\_database(database\_filename)  
  
 formula\_change**,** reaction\_description = input\_reaction\_info()  
  
 mass\_difference = calculate\_mass\_difference(formula\_change)  
  
 database = add\_reaction\_to\_database(database**,** formula\_change**,** reaction\_description**,** mass\_difference)  
  
 save\_database(database**,** database\_filename)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **工作**示意图





1. **Edge生成**
   1. **Δm/z邻接矩阵生成**

* 输入：Targeted Ions.csv文件（F2-Targeted Ions.csv）（UTF-8编码）第一行为ID（升序排列），第2行为m/z
* 输出：Δm/z邻接矩阵文件（F4\_Adjacency\_Matrix\_mz.csv）

Python脚本 《Adjacency\_Matrix\_mz&Rt.py》

* import csv  
    
  def generate\_adjacency\_matrix(csv\_file**,** output\_file**,** value\_column):  
   # 读取CSV文件  
   # 读取CSV文件  
   data = []  
   with open(csv\_file**,** 'r'**,** encoding='utf-8') as file:  
   csv\_reader = csv.reader(file)  
   next(csv\_reader) # 跳过第一行（标题行）  
   for row in csv\_reader:  
   data.append(row)  
    
   # 获取ID序列和对应的值的列表  
   ids = [row[**0**] for row in data]  
   values = [float(row[value\_column]) for row in data]  
    
   # 创建邻接矩阵  
   adjacency\_matrix = [[**0**] \* len(ids) for \_ in range(len(ids))]  
    
   # 填充邻接矩阵  
   for i in range(len(ids)):  
   for j in range(len(ids)):  
   value\_diff = abs(values[i] - values[j])  
   adjacency\_matrix[i][j] = value\_diff  
    
   # 将邻接矩阵写入文件  
   with open(output\_file**,** 'w'**,** newline='') as file:  
   csv\_writer = csv.writer(file)  
   csv\_writer.writerow([''] + ids) # 写入第一行，ID序列  
   for i in range(len(ids)):  
   csv\_writer.writerow([ids[i]] + adjacency\_matrix[i]) # 写入每一行  
    
   print(f"{output\_file} has been saved!")  
    
  # 生成邻接矩阵并保存为Adjacency\_Matrix\_mz.csv  
  generate\_adjacency\_matrix('F2-Targeted Ions.csv'**,** 'F4\_Adjacency\_Matrix\_mz.csv'**, 1**)  
  1. Δm/z邻接矩阵与In-house代谢反应库匹配&1-0矩阵生成
* 输入：Δm/z邻接矩阵文件（F4\_Adjacency\_Matrix\_mz.csv）、内置代谢反应库（F3-Metabolic Reaction Database.csv）
* 输出： Δm/z邻接矩阵与F3-Metabolic Reaction Database.csv匹配结果：

获得匹配,填充匹配的Δm/z值，否则为0（F5\_Re\_Adjacency\_Matrix\_mz.csv），进而获得Δm/z邻接矩阵与In-house-Reaction.csv匹配的1-0矩阵：获得匹配填充1否则为0（F6\_ 1\_0\_Adjacency\_Matrix\_mz.csv）

Python脚本 《Diff\_mz\_Matching.py》

import csv  
  
def read\_csv\_file(filename):  
 data = []  
 with open(filename**,** 'r') as file:  
 reader = csv.reader(file)  
 for row in reader:  
 data.append(row)  
 return data  
  
def write\_csv\_file(filename**,** data):  
 with open(filename**,** 'w'**,** newline='') as file:  
 writer = csv.writer(file)  
 writer.writerows(data)  
  
  
def replace\_matching\_values(adj\_matrix**,** inhouse\_data**,** threshold):  
 # 获取列名所在的行  
 header\_row = inhouse\_data[**0**]  
 mass\_difference\_index = header\_row.index("Mass Difference(Da)")  
  
 for i in range(**1,** len(adj\_matrix)):  
 for j in range(**1,** len(adj\_matrix[i])):  
 if adj\_matrix[i][j] != '0':  
 mz\_diff = float(adj\_matrix[i][j])  
 for row in inhouse\_data[**1**:]:  
 theoretical\_diff = float(row[mass\_difference\_index])  
 if abs(mz\_diff - theoretical\_diff) < threshold:  
 adj\_matrix[i][j] = theoretical\_diff  
 break  
 else:  
 adj\_matrix[i][j] = '0'  
 return adj\_matrix  
  
  
def replace\_with\_ones(adj\_matrix):  
 for i in range(**1,** len(adj\_matrix)):  
 for j in range(**1,** len(adj\_matrix[i])):  
 if adj\_matrix[i][j] != '0':  
 adj\_matrix[i][j] = '1'  
 return adj\_matrix  
  
# 读取Adjacency\_Matrix\_mz.csv文件  
adj\_matrix = read\_csv\_file('F4\_Adjacency\_Matrix\_mz.csv')  
  
# 读取In-House Reaction.csv文件  
inhouse\_data = read\_csv\_file('F3-Metabolic Reaction Database.csv')  
  
# 第一步：将邻接矩阵中的m/z差值与In-House Reaction.csv中的理论值进行匹配  
threshold = **0.005**replaced\_adj\_matrix = replace\_matching\_values(adj\_matrix**,** inhouse\_data**,** threshold)  
  
# 将结果写入Re\_Adjacency\_Matrix\_mz.csv文件  
write\_csv\_file('F5\_Re\_Adjacency\_Matrix\_mz.csv'**,** replaced\_adj\_matrix)  
print("Re\_Adjacency\_Matrix\_mz.csv has been saved !")  
  
# 第二步：将Re\_Adjacency\_Matrix\_mz.csv中的非零m/z差值替换为1  
one\_zero\_adj\_matrix = replace\_with\_ones(replaced\_adj\_matrix)  
  
# 将结果写入1\_0\_Adjacency\_Matrix\_mz.csv文件  
write\_csv\_file('F6\_1\_0\_Adjacency\_Matrix\_mz.csv'**,** one\_zero\_adj\_matrix)  
print("1\_0\_Adjacency\_Matrix\_mz.csv has been saved !")

* 1. 相似度邻接矩阵生成
* 输入： mgf文件（F1-MSDIAL-MS2.mgf）、Targeted Ions.xlsx文件（F2-Targeted Ions.csv）
* 输出： 相似度邻接矩阵文件"F7\_ModCos\_Adjecency\_matrix.csv" ，ID升序排列

Python脚本 《ModCos\_Adjacency\_Matrix.py》

#Calculate MS/MS cosine similarity  
*"""  
function detail:  
Calculate MS/MS cosine similarity by Python mathms package!  
Read and filter mgf files!  
reference: https://github.com/matchms/matchms  
"""*# \_\*\_coding:utf-8\_\*\_  
#!usr/bin/env python3  
  
## Importing python packages  
import os  
import time  
import pandas as pd  
from matchms.importing import load\_from\_mgf  
from matchms import calculate\_scores  
from matchms.similarity import ModifiedCosine  
  
## Set up similarity calculation parameters  
similarity\_measure = ModifiedCosine(tolerance=**0.005**)  
  
## Initialization program execution start time  
start = time.time()  
  
print(os.getcwd())  
#E:\YPF\YPF-Metabolic profile-20211230\Online-Data\Dog\TMN  
  
## Read mgf file  
filename\_mgf1 = "F1-MSDIAL-MS2.mgf"  
spectrums1=list(load\_from\_mgf(filename\_mgf1))  
  
## Read lists for filtering  
target\_ions\_csv\_file = "F2-Targeted Ions.csv"  
df\_fliter\_all = pd.read\_csv(target\_ions\_csv\_file)  
print("The {} has been read!".format(target\_ions\_csv\_file))  
print("The target ions count is {}".format(str(df\_fliter\_all.shape[**0**])))  
## Define the function for filtering by ID  
def my\_fliter\_spec\_by\_scan\_id(spec\_f**,**filter\_scan\_id):  
 spec\_f2 = []  
 spec\_f1 = spec\_f  
 for i in range(len(spec\_f1)):  
 #print(i)  
 #print(spec\_f1[i].metadata["mslevel"]=="2")  
 if int(spec\_f1[i].metadata["scans"]) in filter\_scan\_id :  
 #print(i)  
 spec\_f2.append(spec\_f1[i])  
 return spec\_f2  
  
## Define the function that converts the cosine score into a matrix  
def conbine\_score\_to\_matrix\_f(my\_filter\_score\_p**,**specf1):  
 my\_filter\_id = []  
 for i in range(len(specf1)):  
 my\_filter\_id.append(specf1[i].metadata["scans"])  
 df\_score1 = pd.DataFrame(my\_filter\_score\_p.scores[**0**])["score"]  
 for i in range(**1,**len(specf1)):  
 df\_score2 = pd.DataFrame(my\_filter\_score\_p.scores[i])["score"]  
 df\_score1 = pd.concat([df\_score1**,**df\_score2]**,**axis=**1**)  
 df\_score1.index = my\_filter\_id  
 df\_score1.columns = my\_filter\_id  
 return df\_score1  
  
## Define the function that calculates the cosine score of the fraction  
def cal\_filter\_ms2\_cosine():  
 # cal\_f\_name = df\_fliter\_all.sheet\_names[sheet\_f]  
 # df\_fliter = df\_fliter\_all.parse(sheet\_name=sheet\_f)  
 cal\_f\_name = target\_ions\_csv\_file.split(".")[**0**]  
 df\_fliter = df\_fliter\_all  
 filter\_scan\_ID = df\_fliter["ID"].to\_list()  
 specf1 = my\_fliter\_spec\_by\_scan\_id(spectrums1**,**filter\_scan\_ID)  
 print(len(specf1))  
 my\_filter\_scores1 = calculate\_scores(specf1**,** specf1**,** similarity\_measure**,** is\_symmetric=True)  
 #conbine\_score\_to\_matrix\_f(my\_filter\_scores1,specf1).to\_csv("{}\_Adjecency\_matrix.csv".format(cal\_f\_name))  
 #conbine\_score\_to\_matrix\_f(my\_filter\_scores1, specf1).to\_csv("{}\_Adjecency\_matrix.csv".format("all-pos"))  
 conbine\_score\_to\_matrix\_f(my\_filter\_scores1**,** specf1).to\_csv("{}\_Adjecency\_matrix.csv".format("F7\_ModCos"))  
 print("{} has been saved!".format(cal\_f\_name))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 ## Calling the main function  
 #for i in len(df\_fliter\_all.sheet\_names):  
 cal\_filter\_ms2\_cosine()  
  
 ## Calculate program run time  
 end = time.time()  
 print("The program takes a total of {:.2f} seconds to execute!".format(end - start))

* 1. 矩阵加和&初始Edge输出
* 输入：相似度邻接矩阵文件（F7\_ModCos\_Adjecency\_matrix.csv）、Δm/z匹配1-0矩阵（F6\_1\_0\_Adjacency\_Matrix\_mz.csv）
* 输出：

step1:输出合并的邻接矩阵文件（F8\_Mz&Similarity\_Adjecency\_matrix.csv）

step2：将邻接矩阵文件F8\_Mz&Similarity\_Adjecency\_matrix.csv）转化为初始Edge文件

step3：Edge的过滤（**MS2相似度>阈值&代谢反应匹配 例如Value > 1.5**）

* 输入：输出可用于网络构建的初始Edge文件Mz&Similarity\_Edge\_1.5.csv（F9\_Mz&Similarity\_RawEdge.csv，(删除重复组合的Edge））

Python脚本 《Mz\_Similarity\_RawEdge.py》

import pandas as pd  
  
# 读取CSV文件并使用pandas  
similarity\_matrix = pd.read\_csv('F7\_ModCos\_Adjecency\_matrix.csv'**,** index\_col=**0**)  
mz\_matrix = pd.read\_csv('F6\_1\_0\_Adjacency\_Matrix\_mz.csv'**,** index\_col=**0**)  
  
# 直接执行数据相加操作，保持第一行和第一列的数值不变  
result\_matrix = similarity\_matrix.add(mz\_matrix**,** fill\_value=**0**)  
  
# 写入CSV文件，保持第一行和第一列的数值不变  
result\_matrix.to\_csv('F8\_Mz&Similarity\_Adjecency\_matrix.csv')  
print("Mz&Similarity\_Adjecency\_matrix.csv has been saved!")  
  
# 提取邻接矩阵数据并去重  
edge\_data = result\_matrix.stack().reset\_index()  
edge\_data.columns = ['ID1'**,** 'ID2'**,** 'Value']  
edge\_data = edge\_data[edge\_data['Value'] > **1.5**]  
edge\_data = edge\_data[edge\_data['ID1'].astype(int) - edge\_data['ID2'].astype(int) <= **0**]  
  
# 写入CSV文件  
edge\_data.to\_csv('F9\_Mz&Similarity\_RawEdge.csv'**,** index=False)

* 1. Edge的赋值的匹配和输出

**Function1:**基于F9\_Mz&Similarity\_RawEdge.csv文件的ID对，在F2-Targeted Ions.csv中匹配ID，计算Δm/z, ΔRt, 多样本分布相似度，并增加对应赋值，**此脚本更新F9\_Mz&Similarity\_RawEdge.csv**

Python脚本 《Corr\_Rt\_mz\_Calculater.py》

import pandas as pd  
import numpy as np  
  
# 定义一个函数来计算相关性  
def calculate\_correlation(detected\_ions\_data**,** edge\_data):  
 correlations = []  
  
 # 遍历Edge.csv中的每一行  
 for index**,** row in edge\_data.iterrows():  
 ID1 = row['ID1']  
 ID2 = row['ID2']  
  
 # 检查ID1和ID2是否在Detected\_Ions.csv数据中  
 if ID1 not in detected\_ions\_data['ID'].values or ID2 not in detected\_ions\_data['ID'].values:  
 correlations.append(None)  
 else:  
 # 提取ID1和ID2对应的响应强度值列，此处表示自Targeted Ions.csv的第17列开始为离子的样本分布信息  
 ID1\_data = detected\_ions\_data.loc[detected\_ions\_data['ID'] == ID1**,** detected\_ions\_data.columns[**17**:]]  
 ID2\_data = detected\_ions\_data.loc[detected\_ions\_data['ID'] == ID2**,** detected\_ions\_data.columns[**17**:]]  
  
 # 计算线性相关系数  
 correlation = np.corrcoef(ID1\_data.values[**0**]**,** ID2\_data.values[**0**])[**0, 1**]  
 correlations.append(correlation)  
  
 # 将结果添加到Edge.csv文件的新列"Correlation"  
 edge\_data['Correlation'] = correlations  
  
# 定义一个函数来计算保留时间差异  
def calculate\_retention\_time\_difference(detected\_ions\_data**,** edge\_data):  
 rt\_differences = []  
  
 # 遍历Edge.csv中的每一行  
 for index**,** row in edge\_data.iterrows():  
 ID1 = row['ID1']  
 ID2 = row['ID2']  
  
 # 检查ID1和ID2是否在Detected\_Ions.csv数据中  
 if ID1 not in detected\_ions\_data['ID'].values or ID2 not in detected\_ions\_data['ID'].values:  
 rt\_differences.append(None)  
 else:  
 # 提取ID1和ID2对应的保留时间列  
 ID1\_rt = detected\_ions\_data.loc[detected\_ions\_data['ID'] == ID1**,** 'Rt(min)'].values[**0**]  
 ID2\_rt = detected\_ions\_data.loc[detected\_ions\_data['ID'] == ID2**,** 'Rt(min)'].values[**0**]  
  
 # 计算保留时间差值的绝对值  
 rt\_difference = abs(ID1\_rt - ID2\_rt)  
 rt\_differences.append(rt\_difference)  
  
 # 将结果添加到Edge.csv文件的新列"Retention Time Difference"  
 edge\_data['Retention Time Difference'] = rt\_differences  
  
# 定义一个函数来计算m/z差异  
def calculate\_MZ\_difference(detected\_ions\_data**,** edge\_data):  
 mz\_differences = []  
  
 # 遍历Edge.csv中的每一行  
 for index**,** row in edge\_data.iterrows():  
 ID1 = row['ID1']  
 ID2 = row['ID2']  
  
 # 检查ID1和ID2是否在Detected\_Ions.csv数据中  
 if ID1 not in detected\_ions\_data['ID'].values or ID2 not in detected\_ions\_data['ID'].values:  
 mz\_differences.append(None)  
 else:  
 # 提取ID1和ID2对应的m/z列  
 ID1\_mz = detected\_ions\_data.loc[detected\_ions\_data['ID'] == ID1**,** 'm/z'].values[**0**]  
 ID2\_mz = detected\_ions\_data.loc[detected\_ions\_data['ID'] == ID2**,** 'm/z'].values[**0**]  
  
 # 计算m/z差值的绝对值  
 mz\_difference = abs(ID1\_mz - ID2\_mz)  
 mz\_differences.append(mz\_difference)  
  
 # 将结果添加到Edge.csv文件的新列"MZ Difference"  
 edge\_data['MZ Difference'] = mz\_differences  
  
# 定义Detected\_Ions.csv文件的路径  
detected\_ions\_file = 'F2-Targeted Ions.csv'  
  
# 读取Detected\_Ions.csv文件  
detected\_ions\_data = pd.read\_csv(detected\_ions\_file)  
  
# 定义Edge.csv文件的路径  
edge\_file = ' F9\_Mz&Similarity\_RawEdge.csv '  
  
# 读取Edge.csv文件  
edge\_data = pd.read\_csv(edge\_file)  
  
# 调用三个函数来执行计算  
calculate\_correlation(detected\_ions\_data**,** edge\_data)  
calculate\_retention\_time\_difference(detected\_ions\_data**,** edge\_data)  
calculate\_MZ\_difference(detected\_ions\_data**,** edge\_data)  
  
# 保存更新后的Edge.csv文件  
edge\_data.to\_csv(edge\_file**,** index=False)

**Function2:Edge值的匹配填充（代谢反应相关）及冗余判断**

基于更新F9\_Mz&Similarity\_RawEdge.csv中的Δm/z与F3-Metabolic Reaction Database.csv匹配，返回分子式变化、反应描述，及分子量变化；同时依据计算值填充MS2的ModCos相似度值，执行冗余判断。

**输出最终分子网络Edge文件：F11\_Updated\_Matched\_Edge.csv 可用于分子网络构建**

**需要识别的冗余数据定义为：**

1.相同的保留时间窗口内ΔRt ≤ 0.015 min内潜在的不同加和离子和源内碎片离子

2.不同加和离子和源内碎片离子均源于同一个分子，其具有较高样本分布相关性和MS2相似性，且均在同一个时间窗口流出；

3.样本分布相关性为计算为：在同一个保留时间窗口内，任意两个离子在多个样本间的响应值，进行线性相关回归，r>0.9则认为高相关；

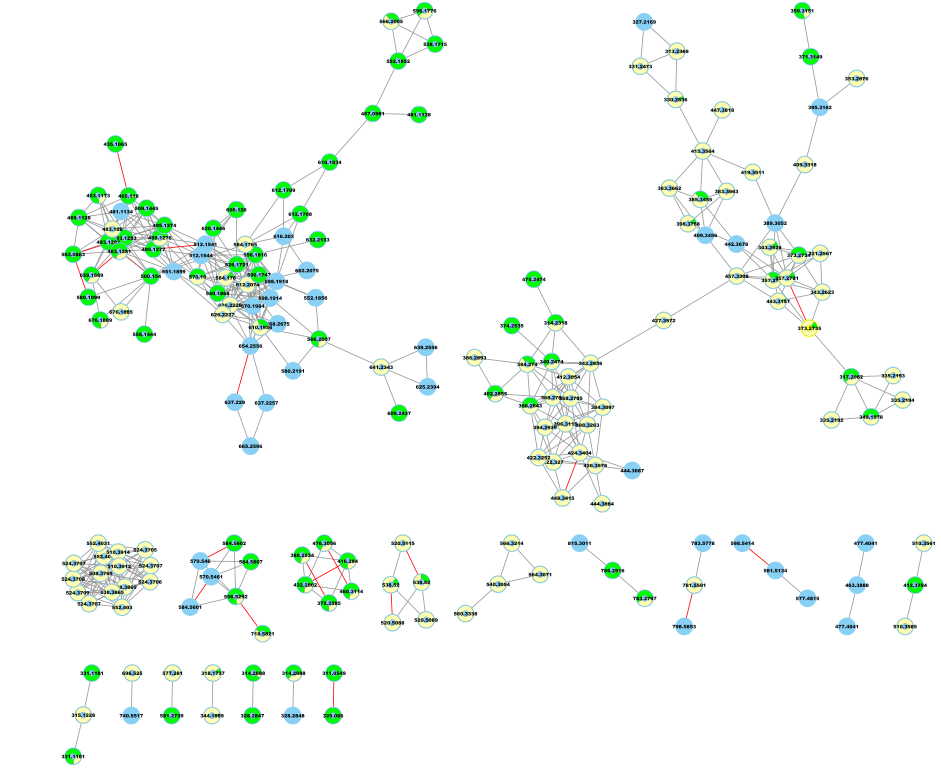
**统计各个反应数量输出：F12\_Matched FormulaChange Count.csv**

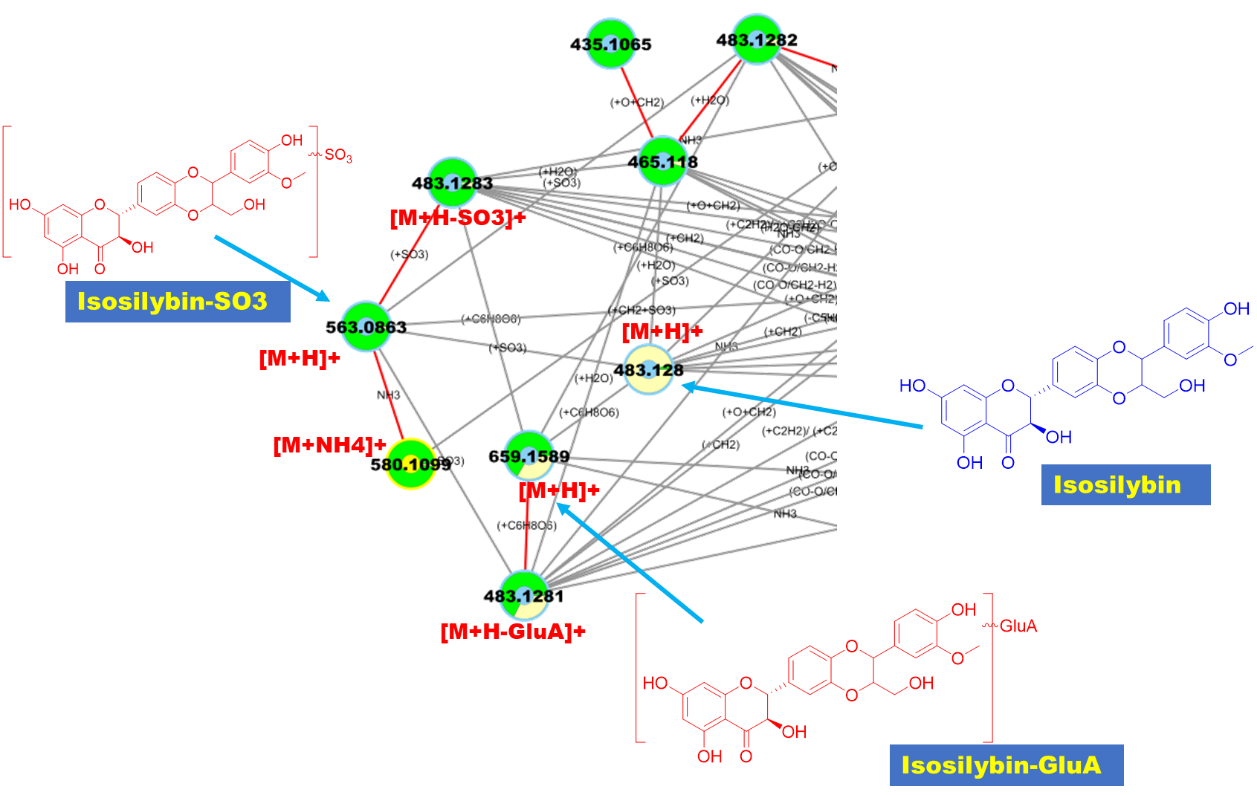
Python脚本 《EdgeValue\_Matching.py》

# 导入pandas库  
import pandas as pd  
  
# 读取测定值和理论值的CSV文件  
measured = pd.read\_csv("F9\_Mz&Similarity\_RawEdge.csv ")  
theoretical = pd.read\_csv("F3-Metabolic Reaction Database.csv")  
  
# 设定MZ Difference匹配阈值  
threshold = **0.01**# 创建一个空的DataFrame来存储匹配结果  
matched = pd.DataFrame()  
  
# 遍历测定值的每一行  
for i**,** row in measured.iterrows():  
 # 获取当前行的MZ Difference值  
 mz = row["MZ Difference"]  
 # 在理论值中找到与当前MZ Difference值最接近的一行  
 closest = theoretical.iloc[(theoretical["Mass Difference(Da)"] - mz).abs().idxmin()]  
 # 如果最接近的差值小于阈值，则认为匹配成功  
 if abs(closest["Mass Difference(Da)"] - mz) < threshold:  
 # 将匹配的理论MZ Difference，FormulaChange和Class添加到当前行的后面  
 row["Matched MZ Difference"] = closest["Mass Difference(Da)"]  
 row["Matched FormulaChange"] = closest["Formula Change"]  
 row["Matched Reaction Description"] = closest["Reaction Description"]  
 # 将当前行拼接到匹配结果中，忽略索引  
 matched = pd.concat([matched**,** pd.DataFrame([row])]**,** ignore\_index=True)  
  
# 将匹配结果保存为新的CSV文件  
matched.to\_csv("F10\_Matched\_Edge.csv"**,** index=False)  
  
# 读取匹配结果的CSV文件  
matched = pd.read\_csv("F10\_Matched\_Edge.csv ")  
  
  
# 添加Redundant Data列  
matched["Redundant Data"] = (matched["Correlation"] >= **0.9**) & (matched["Retention Time Difference"] <= **0.015**)  
  
# 添加ModCos列  
matched["ModCos"] = matched["Value"] - **1**# 保存更新后的CSV文件  
matched.to\_csv("F11\_Updated\_Matched\_Edge.csv"**,** index=False)  
  
  
# 对FormulaChange列进行计数，并保存为一个新的Series  
counts = matched["Matched FormulaChange"].value\_counts()  
  
# 将counts结果保存为文件导出，文件名为counts.csv，分隔符为逗号，编码为utf-8  
counts.to\_csv("F12\_Matched FormulaChange Count.csv"**,** sep=","**,** encoding="utf-8")

1. 分子网络可视化

将所生成的 F11\_Updated\_Matched\_Edge.csv文件（Edge文件）和F2-Targeted Ions.csv导入Cytoscape网络可视化，分子网络文件为**F13\_Matched\_Reaction\_MN.cys**：

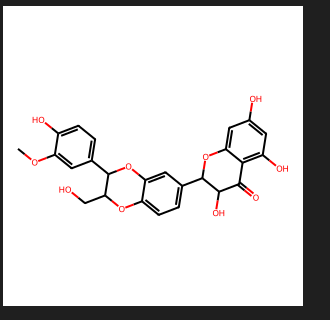




1. 化学结构可视化（未实现）

Node表F2-Targeted Ions.csv中的MS/MS matched值为TURE的离子可以依据SMILES的SMILES格式，通过下列代码实现分子结构可视化：

输入：O=C1C=2C(O)=CC(O)=CC2OC(C3=CC=C4OC(CO)C(OC4=C3)C5=CC=C(O)C(OC)=C5)C1O

输出：

Python脚本 《SMILES\_to\_Structure.py》

from rdkit import Chem  
from rdkit.Chem import Draw  
  
# 读取SMILES字符串  
smiles = "O=C1C=2C(O)=CC(O)=CC2OC(C3=CC=C4OC(CO)C(OC4=C3)C5=CC=C(O)C(OC)=C5)C1O"  
  
# 将SMILES字符串转换为RDKit分子对象  
mol = Chem.MolFromSmiles(smiles)  
  
# 生成分子结构图  
img = Draw.MolToImage(mol**,** size=(**300, 300**))  
  
# 显示图片  
img.show()