#Python code: to calculate “key files” and “triplet file” for tryptophan 3D structures
#Input files: (i) drug_atom_lexical_txt.csv; (ii) sample_details_psi_ab_mix1.csv; (iii) pdb
#Output files: “key files” and “triplet files” will be generated for each tryptophan reside

```
# Program to calculate the amino acid triplets and key Frequency
# Author: Tarikul Islam Milon


import csv
import math
import Bio.PDB
from Bio.PDB import PDBParser
import pandas as pd
import os

dTheta = 29
dLen = 58
numOfLabels = 112

# AminoAcidName=input('Enter the Amino Acid Name:')
#aa_list = ['LYS', 'HIS', 'GLU', 'ARG', 'VAL', 'SER', 'PRO', 'PHE',
'MET', 'LEU', 'ILE', 'GLY', 'CYS', 'ASP', 'ALA',
          #'TRP', 'GLN', 'ASN', 'TYR', 'THR', 'TPO', 'SEP', 'PTR']
aa_list =['TRP']
Incomplete_residue = {'LYS': 84, 'HIS': 120, 'GLU': 84, 'ARG': 165,
'VAL': 35, 'SER': 20, 'PRO': 35, 'PHE': 165,
                      'MET': 56, 'LEU': 56, 'ILE': 56, 'GLY': 4, 'CYS':
20, 'ASP': 56, 'ALA': 10, 'TRP': 364, 'GLN': 84,
                      'ASN': 56, 'TYR': 220, 'THR': 35, 'TPO': 165,
'SEP': 120, 'PTR': 560}
# We can chnage amino acid
df = pd.read_csv('sample_details_psi_ab_mix1.csv')
PDB_list = df['protein'].to_list()
Chain = df['chain'].to_list()

atomSeq = {}
atomSeqNumber = open("drug_atom_lexical_txt.csv", 'r')
reader2 = csv.reader(atomSeqNumber)
next(reader2)
for row in reader2:
    atomSeq[row[2]] = row[1]
atomSeqNumber.close()


# Theta Bin for 3D

def thetaClass_(Theta):
    # classT=0
    if Theta >= 0 and Theta < 12.11:
        classT = 1
    elif Theta >= 12.11 and Theta < 17.32:
        classT = 2
    elif Theta >= 17.32 and Theta < 21.53:
```

```
    classT = 3
elif Theta >= 21.53 and Theta < 25.21:
    classT = 4
elif Theta >= 25.21 and Theta < 28.54:
    classT = 5
elif Theta >= 28.54 and Theta < 31.64:
    classT = 6
elif Theta >= 31.64 and Theta < 34.55:
    classT = 7
elif Theta >= 34.55 and Theta < 37.34:
    classT = 8
elif Theta >= 37.34 and Theta < 40.03:
    classT = 9
elif Theta >= 40.03 and Theta < 42.64:
    classT = 10
elif Theta >= 42.64 and Theta < 45.17:
    classT = 11
elif Theta >= 45.17 and Theta < 47.64:
    classT = 12
elif Theta >= 47.64 and Theta < 50.05:
    classT = 13
elif Theta >= 50.05 and Theta < 52.43:
    classT = 14
elif Theta >= 52.43 and Theta < 54.77:
    classT = 15
elif Theta >= 54.77 and Theta < 57.08:
    classT = 16
elif Theta >= 57.08 and Theta < 59.38:
    classT = 17
elif Theta >= 59.38 and Theta < 61.64:
    classT = 18
elif Theta >= 61.64 and Theta < 63.87:
    classT = 19
elif Theta >= 63.87 and Theta < 66.09:
    classT = 20
elif Theta >= 66.09 and Theta < 68.30:
    classT = 21
elif Theta >= 68.30 and Theta < 70.5:
    classT = 22
elif Theta >= 70.5 and Theta < 72.69:
    classT = 23
elif Theta >= 72.69 and Theta < 79.2:
    classT = 24
elif Theta >= 79.2 and Theta < 81.36:
    classT = 25
elif Theta >= 81.36 and Theta < 83.51:
    classT = 26
elif Theta >= 83.51 and Theta < 85.67:
    classT = 27
elif Theta >= 85.67 and Theta < 87.80:
    classT = 28
elif Theta >= 87.80 and Theta <= 90.00:
    classT = 29
return classT
```

```
# MaxDist bin for 3D
def dist12Class_(dist12):
    if dist12 >= 0 and dist12 < 1:
        classL = 1
    elif dist12 >= 1 and dist12 < 2:
        classL = 2
    elif dist12 >= 2 and dist12 < 3:
        classL = 3
    elif dist12 >= 3 and dist12 < 4:
        classL = 4
    elif dist12 >= 4 and dist12 < 5:
        classL = 5
    elif dist12 >= 5 and dist12 < 6:
        classL = 6
    elif dist12 >= 6 and dist12 < 7:
        classL = 7
    elif dist12 >= 7 and dist12 < 8:
        classL = 8
    elif dist12 >= 8 and dist12 < 9:
        classL = 9
    elif dist12 >= 9 and dist12 < 10:
        classL = 10
    elif dist12 >= 10 and dist12 < 11:
        classL = 11
    elif dist12 >= 11 and dist12 < 12:
        classL = 12
    elif dist12 >= 12 and dist12 < 13:
        classL = 13
    elif dist12 >= 13 and dist12 < 14:
        classL = 14
    elif dist12 >= 14 and dist12 < 15:
        classL = 15
    elif dist12 >= 15 and dist12 < 16:
        classL = 16
    elif dist12 >= 16 and dist12 < 17:
        classL = 17
    elif dist12 >= 17 and dist12 < 18:
        classL = 18
    elif dist12 >= 18 and dist12 < 19:
        classL = 19
    elif dist12 >= 19 and dist12 < 20:
        classL = 20
    elif dist12 >= 20 and dist12 < 21:
        classL = 21
    elif dist12 >= 21 and dist12 < 22:
        classL = 22
    elif dist12 >= 22 and dist12 < 23:
        classL = 23
    elif dist12 >= 23 and dist12 < 24:
        classL = 24
    elif dist12 >= 24 and dist12 < 25:
        classL = 25
```

```
elif dist12 >= 25 and dist12 < 26:
    classL = 26
elif dist12 >= 26 and dist12 < 27:
    classL = 27
elif dist12 >= 27 and dist12 < 28:
    classL = 28
elif dist12 >= 28 and dist12 < 29:
    classL = 29
elif dist12 >= 29 and dist12 < 30:
    classL = 30
elif dist12 >= 30 and dist12 < 31:
    classL = 31
elif dist12 >= 31 and dist12 < 32:
    classL = 32
elif dist12 >= 32 and dist12 < 33:
    classL = 33
elif dist12 >= 33 and dist12 < 34:
    classL = 34
elif dist12 >= 34 and dist12 < 35:
    classL = 35
elif dist12 >= 35 and dist12 < 36:
    classL = 36
elif dist12 >= 36 and dist12 < 37:
    classL = 37
elif dist12 >= 37 and dist12 < 38:
    classL = 38
elif dist12 >= 38 and dist12 < 39:
    classL = 39
elif dist12 >= 39 and dist12 < 40:
    classL = 40
elif dist12 >= 40 and dist12 < 41:
    classL = 41
elif dist12 >= 41 and dist12 < 42:
    classL = 42
elif dist12 >= 42 and dist12 < 43:
    classL = 43
elif dist12 >= 43 and dist12 < 44:
    classL = 44
elif dist12 >= 44 and dist12 < 45:
    classL = 45
elif dist12 >= 45 and dist12 < 46:
    classL = 46
elif dist12 >= 46 and dist12 < 47:
    classL = 47
elif dist12 >= 47 and dist12 < 48:
    classL = 48
elif dist12 >= 48 and dist12 < 49:
    classL = 49
elif dist12 >= 49 and dist12 < 50:
    classL = 50
elif dist12 >= 50 and dist12 < 51:
    classL = 51
elif dist12 >= 51 and dist12 < 52:
    classL = 52
```

```
    elif dist12 >= 52 and dist12 < 53:
        classL = 53
    elif dist12 >= 53 and dist12 < 54:
        classL = 54
    elif dist12 >= 54 and dist12 < 55:
        classL = 55
    elif dist12 >= 55 and dist12 < 56:
        classL = 56
    elif dist12 >= 56 and dist12 < 57:
        classL = 57
    elif dist12 >= 57 and dist12 < 1000:
        classL = 58

    return classL


def calDist(x1, y1, z1, x2, y2, z2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2 + (z2 - z1) ** 2)


for AminoAcidName in aa_list:
    duplicate = []
    Key_Dict_Total = ()
    DataFrame_Index = []
    Group_Information = []

    Output_Folder_Path =
f'/ddnB/work/wxx6941/TSR/code/code/psi_revision/9pdb/output_trp_tsr_a_b'
    outputFile3 =
open(f'{Output_Folder_Path}/Incomplete_Residue{AminoAcidName}.txt', 'w')
    #
outputFile4=open(f'{Output_Folder_Path}/sample_details_{AminoAcidName}.cs
v', 'w')
    for i in range(len(PDB_list)):
        PDB_ID = PDB_list[i]
        Chain_Name = Chain[i]
        rm_duplicate = f'{PDB_ID}_{Chain_Name}'
        if rm_duplicate not in duplicate:
            duplicate.append(rm_duplicate)

            PDB_File_Path =
"/ddnB/work/wxx6941/TSR/code/code/psi_revision/9pdb/PDB_datadir_HRemoved/
{}.pdb".format(
                PDB_ID)
            p = Bio.PDB.PDBParser()
            Structure = p.get_structure('PrimaryStructureChain',
PDB_File_Path)
            model = Structure[0]
            for chain in model:
                if chain.id == Chain_Name:
                    for residue in chain:
                        if str(residue)[9:12] == AminoAcidName.upper():
                            numeric_filter = filter(str.isdigit,
str(residue.id)[6:10])
```

```
                            Res_Id = "".join(numeric_filter)
                            for atom0 in residue:
                                Coord_end = atom0.get_vector()
                                X1_end = Coord_end[0]

                            X1_end = '{:.2f}'.format(X1_end)

                            outputFile1 = open(

f'{Output_Folder_Path}/{PDB_ID}_{Chain_Name.upper()}_{AminoAcidName}_{Res
_Id}_{X1_end}.triplets_theta29_dist17',
                                'w')
                            outputFile2 = open(

f'{Output_Folder_Path}/key_{PDB_ID}_{Chain_Name.upper()}_{AminoAcidName}_
{Res_Id}_{X1_end}.keys_theta29_dist17',
                                'w')
                            # header
                            outputFile1.writelines(
                                'Residue1   Residue2   Residue3   Edge1
Edge2  Edge3\t   Coor_R1\t          Coor_R2\t
CoorR3\tTheta\tmax_dist\td_3\tkey3D\n')
                            outputFile2.writelines('key\t\tfreq\n')

                            keyDict3D = {}
                            xCoord = {}
                            yCoord = {}
                            zCoord = {}
                            Atom = {}
                            counter2 = 0
                            for atom1 in residue:
                                atomCoord = atom1.get_vector()

                                Atom[counter2] = atom1.get_name()
                                xCoord[counter2] = atomCoord[0]
                                yCoord[counter2] = atomCoord[1]
                                zCoord[counter2] = atomCoord[2]

                                counter2 += 1
                            for i in range(len(xCoord)):
                                for j in range(i + 1, len(yCoord)):
                                    for k in range(j + 1, len(zCoord)):
                                        L1 = calDist(xCoord[i],
yCoord[i], zCoord[i], xCoord[j], yCoord[j], zCoord[j])
                                        L2 = calDist(xCoord[j],
yCoord[j], zCoord[j], xCoord[k], yCoord[k], zCoord[k])
                                        L3 = calDist(xCoord[i],
yCoord[i], zCoord[i], xCoord[k], yCoord[k], zCoord[k])

                                        l1 = atomSeq[Atom[j]]
                                        l2 = atomSeq[Atom[k]]
                                        l3 = atomSeq[Atom[i]]
```

```
(L1 ** 2) + 2 * (L2 ** 2) - L3 ** 2)

(L2 ** 2) + 2 * (L3 ** 2) - L1 ** 2)

(L3 ** 2) + 2 * (L1 ** 2) - L2 ** 2)
```

```
Median[Label.index(min(l1, l2, l3))]

index1[Label.index(min(l1, l2, l3))]

l3) == l2:

l2, l3) == l3:

l2, l3) == l1:

l2, l3) == l2:

l2, l3) == l1:

l2, l3) == l3:
```

```
Med1 = (1 / 2) * math.sqrt(2 *

Med2 = (1 / 2) * math.sqrt(2 *

Med3 = (1 / 2) * math.sqrt(2 *

Median = [Med1, Med2, Med3]
Label = [l1, l2, l3]
index1 = [L3, L1, L2]

# 1st Condition
if l1 != l2 != l3:
    X = [l1, l2, l3]
    b3 =

    d12 =

    if d12 == L3 and max(l1, l2,

        d13 = L2
    elif d12 == L3 and max(l1,

        d13 = L1

    elif d12 == L2 and max(l1,

        d13 = L1
    elif d12 == L2 and max(l1,

        d13 = L3
    elif d12 == L1 and max(l1,

        d13 = L2
    elif d12 == L1 and max(l1,

        d13 = L3
    X.remove(max(X))
    X.remove(min(X))
    Label1 = max(l1, l2, l3)
    Label2 = X[0]
    Label3 = min(l1, l2, l3)

# 2nd condition
elif l1 > l2 == l3:
    Label1 = l1
    if L2 > L1:
        b3 = Med3
        d13 = L1
        d12 = L2
        Label2 = l2
        Label3 = l3
    else:
        b3 = Med2
        d13 = L2
```

```
            d12 = L1
            Label2 = l3
            Label3 = l2

    elif l2 > l1 == l3:
        Label1 = l2
        if L3 > L2:
            b3 = Med1
            d13 = L2
            d12 = L3
            Label2 = l3
            Label3 = l1
        else:
            b3 = Med3
            d13 = L3
            d12 = L2
            Label2 = l1
            Label3 = l3

    elif l3 > l1 == l2:
        Label1 = l3
        if L1 > L3:
            b3 = Med2
            d13 = L3
            d12 = L1
            Label2 = l1
            Label3 = l2
        else:
            b3 = Med1
            d13 = L1
            d12 = L3
            Label2 = l2
            Label3 = l1
    # 3rd condition
    elif l1 == l2 > l3:
        b3 = Med3
        Label3 = l3
        if L1 > L3:
            d13 = L1
            d12 = L2
            Label1 = l1
            Label2 = l2
        else:
            d13 = L3
            d12 = L2
            Label1 = l2
            Label2 = l1

    elif l1 == l3 > l2:
        Label3 = l2
        b3 = Med2
        if L2 > L3:
            d13 = L2
            d12 = L1
```

```
            Label1 = l1
            Label2 = l3
        else:
            d13 = L3
            d12 = L1
            Label1 = l3
            Label2 = l1
    elif l2 == l3 > l1:
        Label3 = l1
        b3 = Med1
        if L2 > L1:
            d13 = L2
            d12 = L3
            Label1 = l2
            Label2 = l3
        else:
            d13 = L1
            d12 = L3
            Label1 = l3
            Label2 = l2

    # 4th condition
    if l1 == l2 == l3:
        if L2 >= max(L1, L2, L3):
            b3 = Med3
            d13 = L1
            d12 = L2
            Label1 = l1
            Label2 = l2
            Label3 = l3
        if L1 >= max(L1, L2, L3):
            b3 = Med2
            d13 = L2
            d12 = L1
            Label1 = l1
            Label2 = l3
            Label3 = l2

        if L3 >= max(L1, L2, L3):
            # b3=Med3
            # d13 =L1
            # d12 =L2
            # Corrected
            b3 = Med1
            d13 = L1
            d12 = L3
            Label1 = l3
            Label2 = l2
            Label3 = l1

    a = (d13 ** 2 - (d12 / 2) ** 2 -
b3 ** 2)
    b = (2 * (d12 / 2) * b3)
```

```
                                        Theta1 = (math.acos(a / b)) *
(180 / math.pi)

                                        if Theta1 <= 90:
                                            Theta = Theta1
                                        else:
                                            Theta = abs(180 - Theta1)
                                        maxDist = max(L1, L2, L3)
                                        ClassT1 = thetaClass_(Theta)
                                        ClassL1 = dist12Class_(maxDist)
                                        key3D = dLen * dTheta *
(numOfLabels ** 2) * (int(Label1) - 1) + \
                                                dLen * dTheta *
(numOfLabels) * (int(Label2) - 1) + \
                                                dLen * dTheta *
(int(Label3) - 1) + \
                                                dTheta * (ClassL1 - 1) +
\
                                                (ClassT1 - 1)

                                        if key3D in keyDict3D:
                                            keyDict3D[key3D] += 1
                                        else:
                                            keyDict3D[key3D] = 1

                                        outputFile1.write(
                                            "{}_{}_{}_{}
".format(AminoAcidName, Chain_Name, Res_Id, Atom[i]))
                                        outputFile1.write(
                                            "{}_{}_{}_{}
".format(AminoAcidName, Chain_Name, Res_Id, Atom[j]))
                                        outputFile1.write(
                                            "{}_{}_{}_{}
".format(AminoAcidName, Chain_Name, Res_Id, Atom[k]))
                                        outputFile1.write(" {:.2f}
{:.2f}  {:.2f} ".format(L1, L2, L3))
                                        outputFile1.write(
                                            "  {:.2f},{:.2f},{:.2f}
{:.2f},{:.2f},{:.2f}  ".format(xCoord[i],

yCoord[i],

zCoord[i],

xCoord[j],

yCoord[j],

zCoord[j]))

                                        outputFile1.write(
                                            " {:.2f},{:.2f},{:.2f}
".format(xCoord[k], yCoord[k], zCoord[k]))
                                        outputFile1.write(
```

```
                                                "{:.2f}   {:.2f}   {:.2f}
{:.0f}\n".format(Theta, maxDist, b3, key3D))

                            for value_ in keyDict3D:
                                outputFile2.writelines([str(value_),
'\t', str(keyDict3D[value_]), '\n'])
                            if sum(keyDict3D.values()) ==
Incomplete_residue[AminoAcidName]:
                                Key_Dict_Total += (keyDict3D,)

DataFrame_Index.append(f'{PDB_ID}_{Chain_Name}_{AminoAcidName}_{Res_Id}_{
X1_end}')
                                Group_Information.append(AminoAcidName)

                            else:

outputFile3.writelines(f'{PDB_ID}_{Chain_Name}_{AminoAcidName}_{Res_Id}_{
X1_end}\n')
                            outputFile1.close()
                            outputFile2.close()

    df = pd.DataFrame(Key_Dict_Total, index=Group_Information)
    df = df.rename_axis('group')
    df = df.fillna(0)
    df = df.astype('int')
    df.insert(0, 'protein', DataFrame_Index)
    df.to_csv(f"{Output_Folder_Path}/feature_map_with_header.csv",
header=True, index=True)

    df_Group = pd.DataFrame(columns=['group'], index=DataFrame_Index)
    df_Group = df_Group.rename_axis('protein')
    df_Group['group'] = Group_Information

df_Group.to_csv(f"{Output_Folder_Path}/sample_details_{AminoAcidName}.csv
", header=True, index=True)

    df_Group_2 = pd.DataFrame(columns=['group'], index=DataFrame_Index)
    df_Group_2 = df_Group_2.rename_axis('protein')
    df_Group_2['group'] = Group_Information

df_Group_2.to_csv(f"{Output_Folder_Path}/sample_details_2_{AminoAcidName}
.csv", header=True, index=True)

    df_Clustering = pd.DataFrame(Key_Dict_Total, index=DataFrame_Index)
    df_Clustering = df_Clustering.fillna(0)
    df_Clustering = df_Clustering.astype('int')

    first_column = list(df_Clustering.iloc[:, 0])
    DataFrame_Index_2 = []
    for i in range(len(df_Clustering)):
        DataFrame_Index_2.append(DataFrame_Index[i] + ";" +
str(int(first_column[i])))
    df_Clustering.iloc[:, 0] = DataFrame_Index_2
```

```
df_Clustering.to_csv(f"{Output_Folder_Path}/localFeatureVect_theta29_dist
35_NoFeatureSelection_keyCombine0.csv",
                         header=False, index=False)
    outputFile3.close()

print('completed Successfully')
```