Answers

1.Explain the distance metric you utilized to calculate the similarity/dissimilarity between small molecules.

I use the the Euclidean distance. It can be calculated from the Cartesian coordinates of the points using the Pythagorean theorem, therefore occasionally being called the Pythagorean distance. As below:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}.$$

2.Use a dimensionality reduction algorithm (PCA, t-SNE, UMAP, etc) to generate a 2D visualization of the small molecule dataset. Each point should represent a single molecule. (Note: you may have to plot a subset of your data, depending on which dimensionality reduction algorithm you choose.)

I took a subset of 2000 ligands for the analysis, and I also bin the scores to set up colors for the gerneral TSNE plot

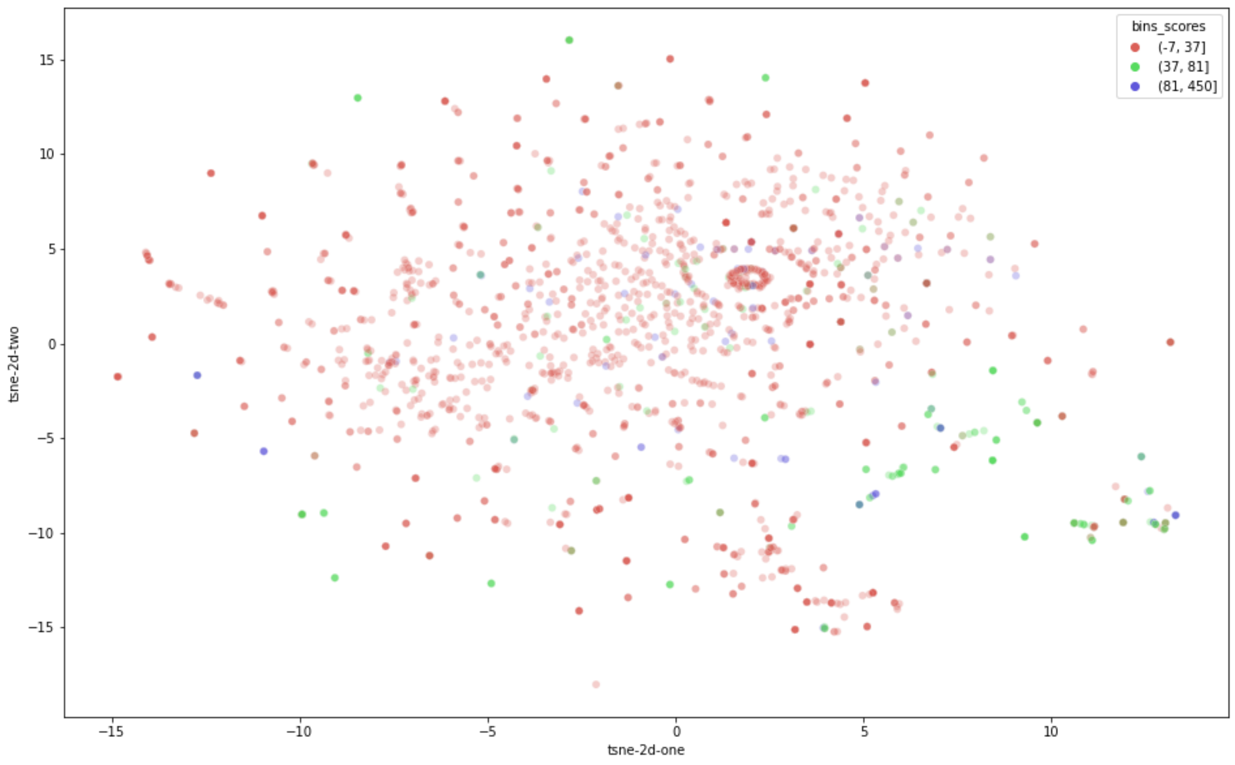
In [5]:

```
import glob
import pandas as pd
import numpy as np
import os
import sys
from algs import Ligand, HierarchicalClustering, PartitionClustering,Si
import matplotlib.pyplot as plt
from __future__ import print_function
import time
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
import json

csv = pd.read_csv('../ligand_information.csv', index_col= False)
Data = list(np.random.randint(1,csv.shape[0],size=2000))

Ligands = []

for i in Data:

    a_ligand = Ligand(csv[csv.index==i])
    Ligands.append(a_ligand)

IDs = []
scores = []
dataset = []
for ligand in Ligands:
    IDs.append(ligand.get_ID())
    scores.append(ligand.get_scores())
    dataset.append(ligand.get_OnBits())
dataset = np.stack((dataset), axis=0)
onBits = [ 'pixel'+str(i) for i in range(dataset.shape[1]) ]
df = pd.DataFrame(dataset,columns=onBits)
_df = df[onBits].values


time_start = time.time()
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(_df)
print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_st

df['scores'] = scores
#score_ir = pd.interval_range(start =-6.2, end = 534.3, freq = 100)

df['bins_scores'] = pd.cut(df['scores'], bins = [-7,37,81,450])
del df['scores']
df['tsne-2d-one'] = tsne_results[:,0]
df['tsne-2d-two'] = tsne_results[:,1]
plt.figure(figsize=(16,10))
```

```
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="bins_scores",
    palette=sns.color_palette("hls", len(set(df['bins_scores']))),
    data=df,
    legend="full",
    alpha=0.3
)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 2000 samples in 0.172s...
[t-SNE] Computed neighbors for 2000 samples in 6.352s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2000
[t-SNE] Computed conditional probabilities for sample 2000 / 2000
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.61
1977
[t-SNE] KL divergence after 300 iterations: 1.613455
t-SNE done! Time elapsed: 42.167349100112915 seconds
```

Out[5]: `<AxesSubplot:xlabel='tsne-2d-one', ylabel='tsne-2d-two'>`

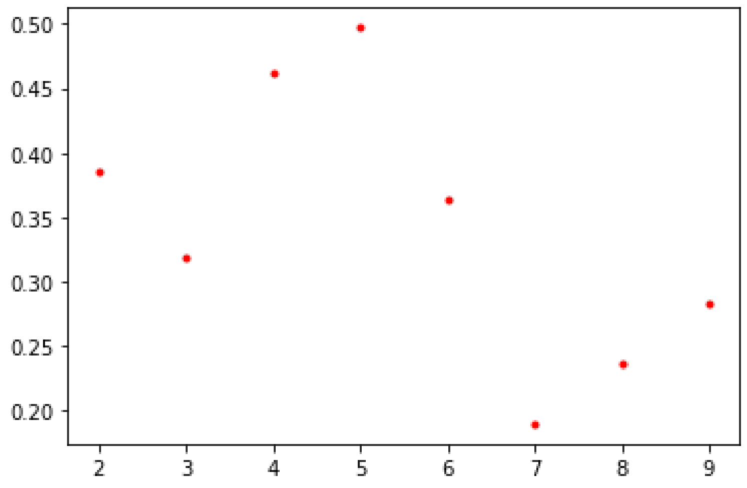


3.Cluster the small molecules using your implementation of a partitioning clustering algorithm. Visualize this clustering by coloring clusters on the 2D visualization generated in question 2.

In [9]:
```
# Trying to find the most suitable cluster numbers
num_Cs = []
Silhouette_Coefficients = []

for num in range(2,10):
    print(num)
    kmeans = PartitionClustering(Ligands, num, 10)
    kmeans.implement()
    Sil_co = Silhouette_Coefficient(Ligands, kmeans.labels)
    num_Cs.append(num)
    Silhouette_Coefficients.append(Sil_co)

```

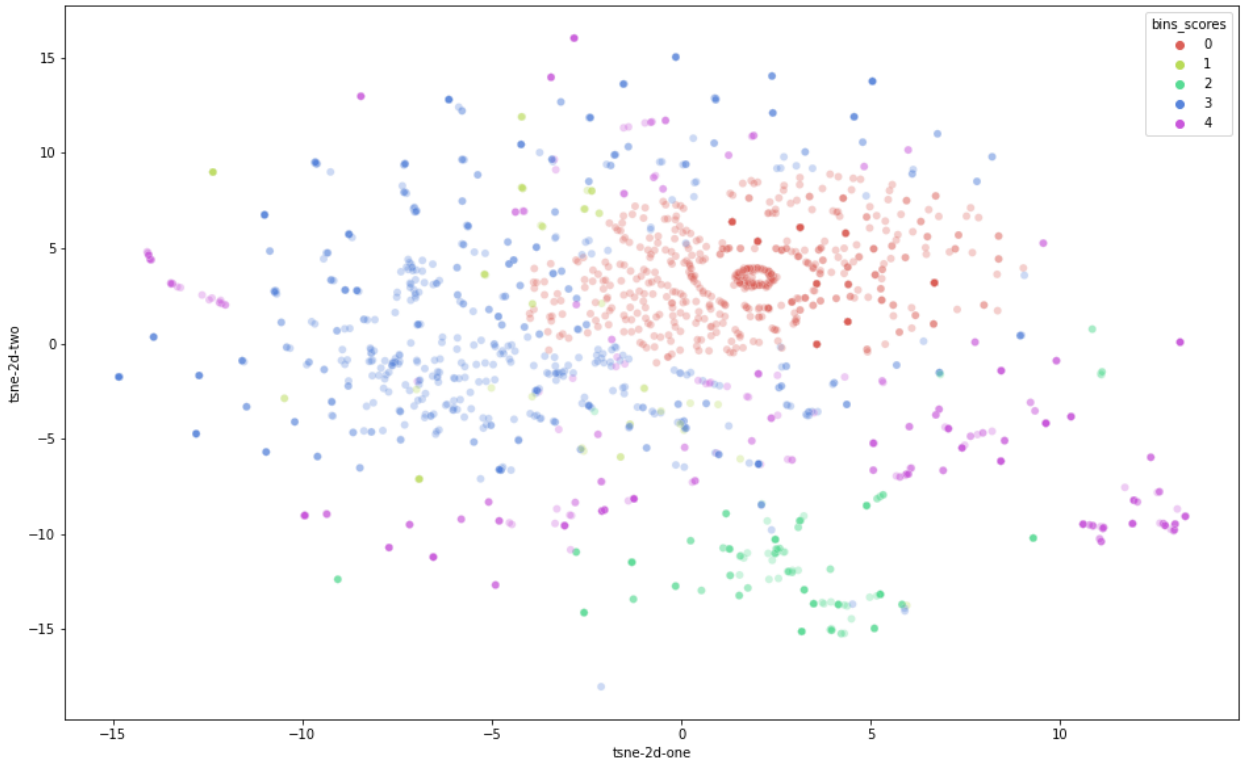
In [7]:
```
plt.plot(num_Cs, Silhouette_Coefficients,'.r')
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x7f55e91da0d0>]
```



In [10]:
```
kmeans = PartitionClustering(Ligands, 5, 10)
kmeans.implement()
```

In [ ]:
```
df['bins_scores'] = kmeans.labels
df['tsne-2d-one'] = tsne_results[:,0]
df['tsne-2d-two'] = tsne_results[:,1]
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="bins_scores",
    palette=sns.color_palette("hls", len(set(df['bins_scores']))),
    data=df,
    legend="full",
    alpha=0.3
)

```



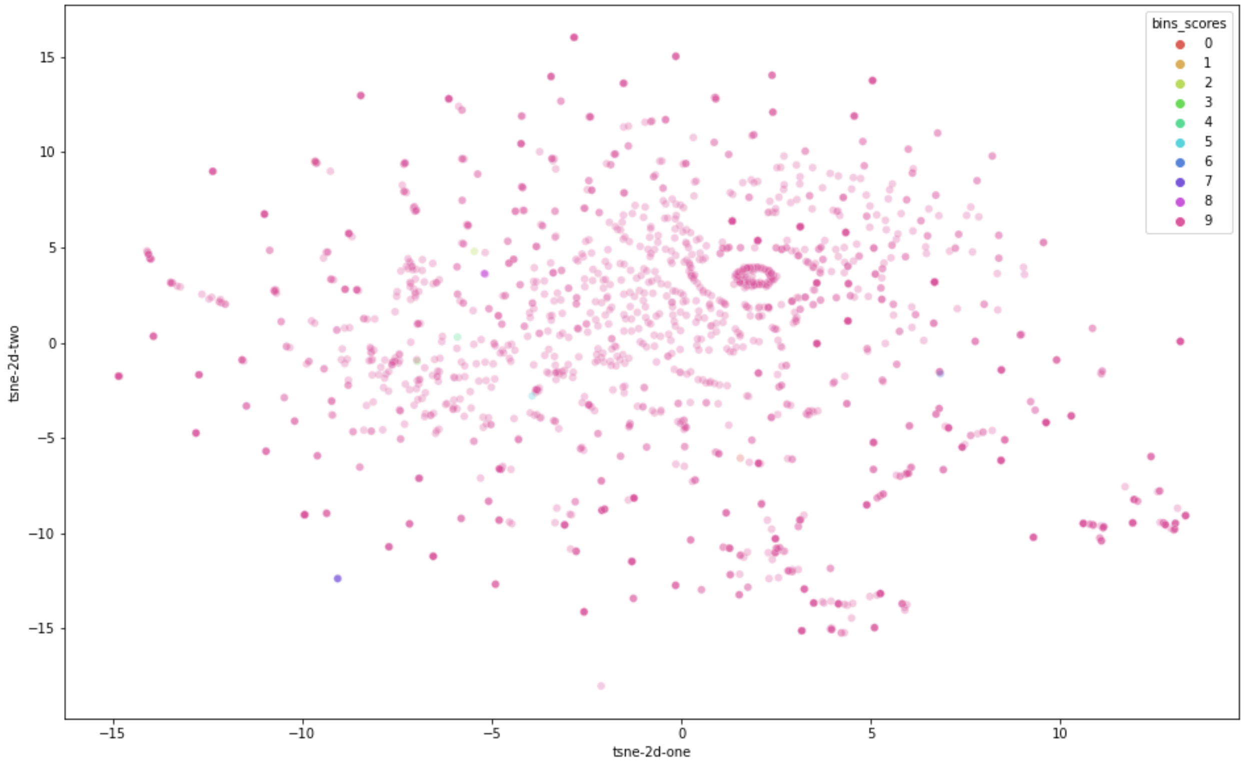
4.Explain your choice of partitioning clustering algorithm. Is it sensitive to initialization conditions? How do you select the number of clusters?

It is sensitive to the initialization conditions. It will be better after several times of iterations. For selecting the number of clusters. I plot the clustering quality scores with the cluster number and I choose 5 clusters based on this results.

5.Cluster the small molecules using your implementation of a hierarchical clustering algorithm. Visualize this clustering in the same way as question

```
Hiera_C = HierarchicalClustering(Ligands, 10)
Hiera_C.implement()
```

```
df['bins_scores'] = Hiera_C.labels
df['tsne-2d-one'] = tsne_results[:,0]
df['tsne-2d-two'] = tsne_results[:,1]
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="bins_scores",
    palette=sns.color_palette("hls", len(set(df['bins_scores']))),
    data=df,
    legend="full",
    alpha=0.3
)
```



6.Explain your choice of hierarchical clustering algorithm. Is it sensitive to initialization conditions? How do you select the number of clusters?

It is a determinate algorthims so it is not sensitive to the initialization conditions. For deciding the number of clusters, I do a plots of the number of clusters verse the clustering quality socres. And I realize that after a certain number, there will just be more single dots adding to the big cluster. So in the end I decide to do 10, in which case it would be ok to seprate the different ligands but also is not a huge cluster number.

7.Evaluate the quality of both clusterings using your implementation of a clustering quality metric. Explain your choice of quality metric. Which clustering performed ‘best’ according to your metric?

From the results (Silhouette_Coefficient) I got, the HierarchicalClustering work a little better than the k means, however, another thing to keep in mind is that the way that HierarchicalClustering is computed is very similar to the way that Silhouette_Coefficient is computed. Given that tsne and kmeans shows a more clear clustering result: although HierarchicalClustering has a higher Silhouette_Coefficient. I still believe that k means sever is a better model for this case.

Silhouette_Coefficient(Ligands, Hiera_C.labels)

Silhouette_Coefficient(Ligands, kmeans.labels)

8. Compare the two clusterings using your implementation of clustering similarity. How similar are the two clusterings using this function?

In [27]:
```
Rand_Index(kmeans.labels, Hiera_C.labels)
```

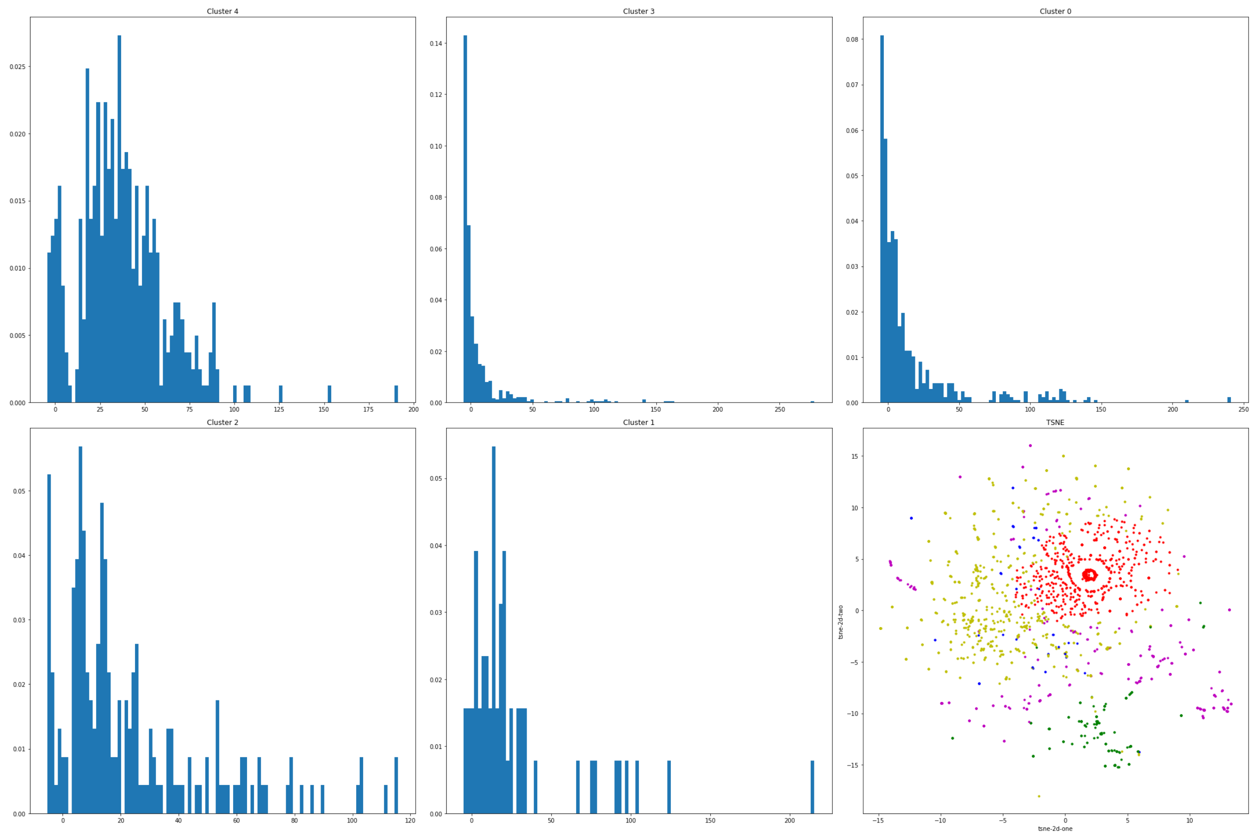
Out[27]:
```
0.2877113556778389
```

Based on the rand index also the tsne results, I think this two cluster methods give very different results.

9. For the “best” clustering, as determined by your quality metric, visualize the distribution of Autodock Vina scores in each cluster. Do members of the same cluster have similar docking scores? Why or why not?

In [29]:
```
# Kmeans
index = 2000
Cluster_scores = {}
for i in range(index):
    c = kmeans.labels[i]
    if c not in Cluster_scores:
        Cluster_scores[c] = [kmeans.scores[i]]
    else:
        Cluster_scores[c].append(kmeans.scores[i])
num = 0
keys = []
for index, key in enumerate(Cluster_scores):
    if len(Cluster_scores[key]) == 1:
        pass
    else:
        keys.append(key)
        num = num +1
```

In [ ]:
```python
plt.rcParams["figure.figsize"]=30,20

fig, axs = plt.subplots(nrows = 2, ncols = 3)
p = 0
for ax in axs.flatten():
    if p <5:
        values = Cluster_scores[keys[p]]
        ax.hist(values, 100, density = True)
        ax.set_title('Cluster '+str(keys[p]))
        p = p +1
    else:
        df['bins_scores'] = kmeans.labels
        cols = ['r','b','g','y','m']
        for i  in range(2000):

            ax.plot(df["tsne-2d-one"][i], df["tsne-2d-two"][i], cols[km
        ax.set_title('TSNE')
        ax.set_xlabel('tsne-2d-one')
        ax.set_ylabel('tsne-2d-two')


fig.tight_layout()
plt.show()
```



Based on the distribution, all of these clusters are having varied scores. However, cluster 2 which was colored in red 1) seems to form a good cluster 2) seems to be more enriched in higher scores. It is not surprised to me that the scores are varied becasue in the k means for calculating the distance, the onBites regrading different location are weighted uniformly, which might not be powerful enough to reflecting Autodock Vina scores.

10. Select the top scoring molecule from each cluster. This is your list of cluster heads. Visualize the top 5 by score in PyMOL and pick your favorite. Are they structurally diverse?

In [46]:
```
num = 0
for index, key in enumerate(Cluster_scores):
    if len(Cluster_scores[key]) == 1:
        continue
    else:
        num = num +1

```

In [47]:
```

top_scores = []
for i in range(p):
    values = Cluster_scores[keys[i]]
    score = max(values)
    top_scores.append(score)
    top_scores.sort()
```

In [48]:
```
top_scores
```

Out[48]:
```
[115.7, 191.3, 215.5, 241.0, 278.2]
```

In [49]:
```
Leads = []
for Ligand in Ligands:
    if Ligand.get_scores() in top_scores:
        Leads.append(Ligand.get_ID())
```

In [50]:
```
Leads
```

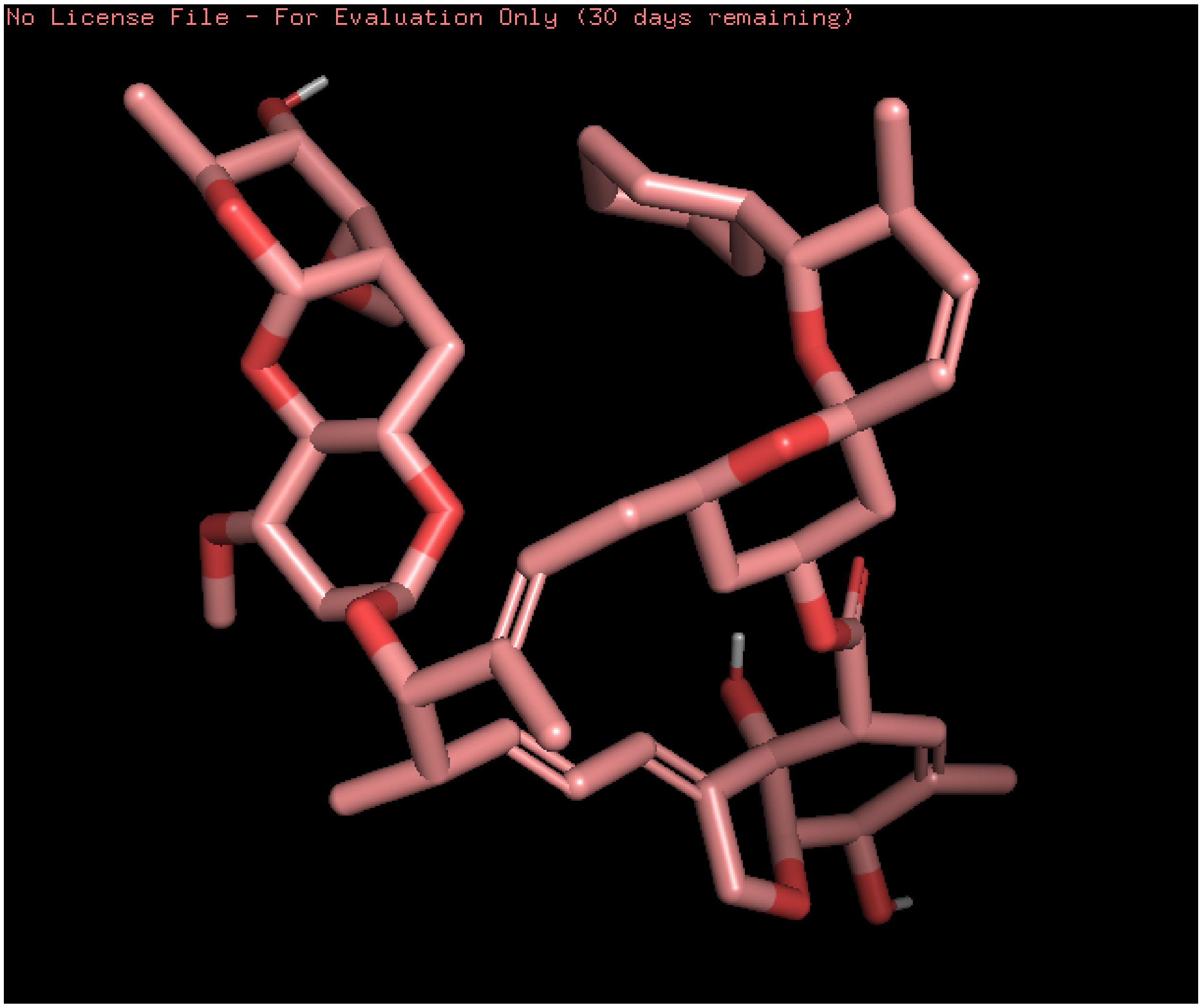
Out[50]:
```
[8418, 8569, 7701, 8901, 6843]
```

Answers: yes, they are all structurally diverse and have a complex structure.

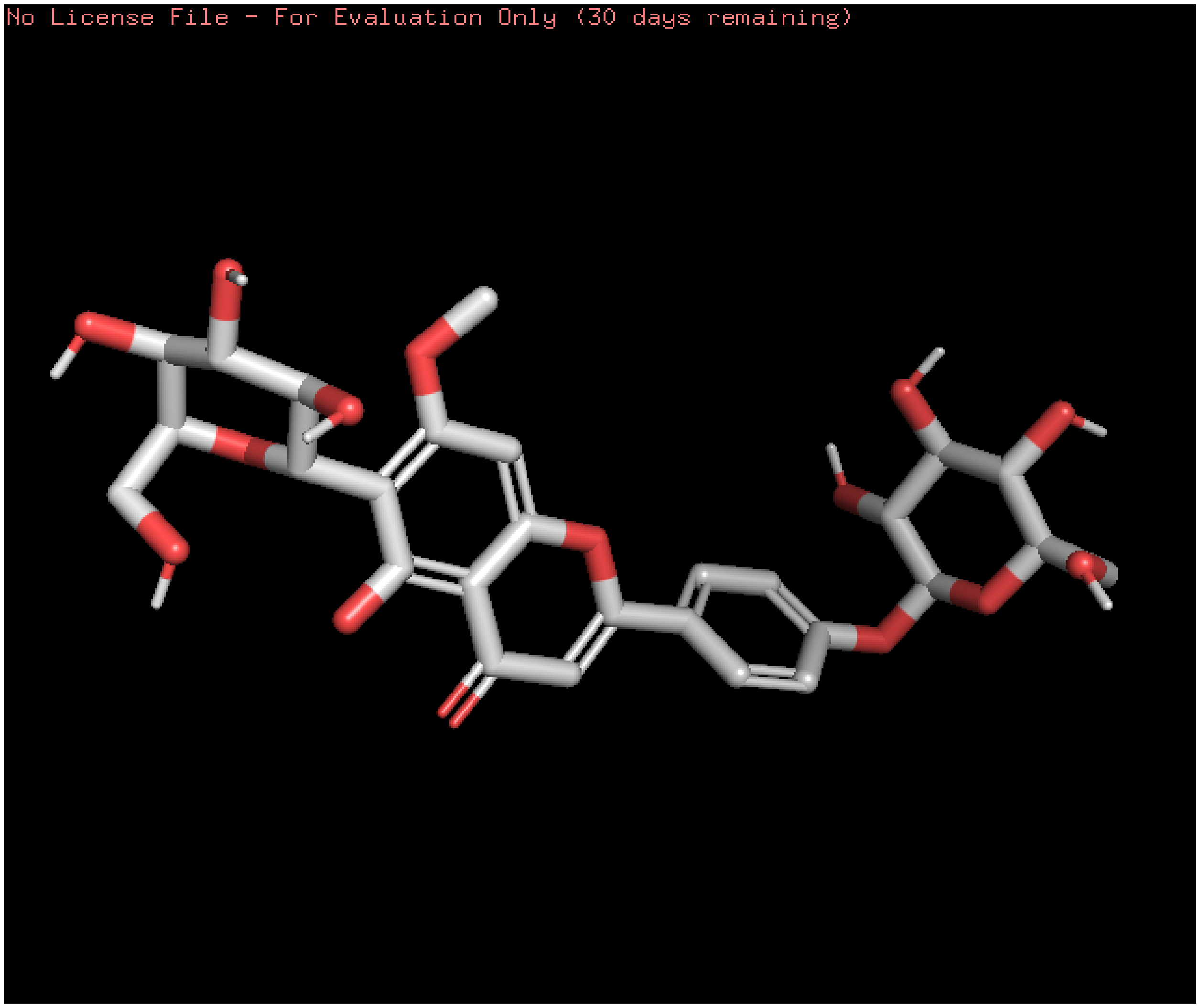
8418

No License File - For Evaluation Only (30 days remaining)

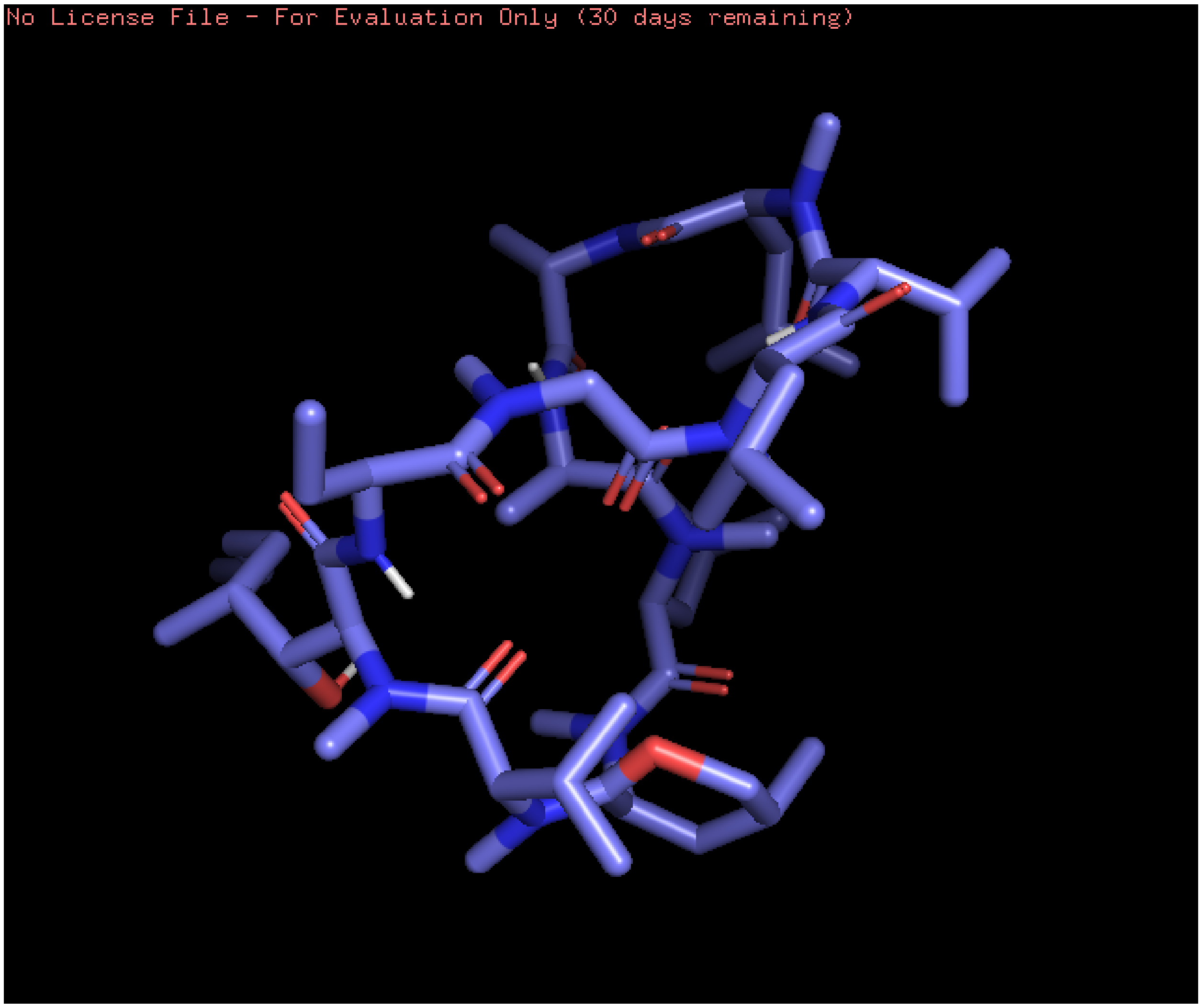
8569


No License File - For Evaluation Only (30 days remaining)

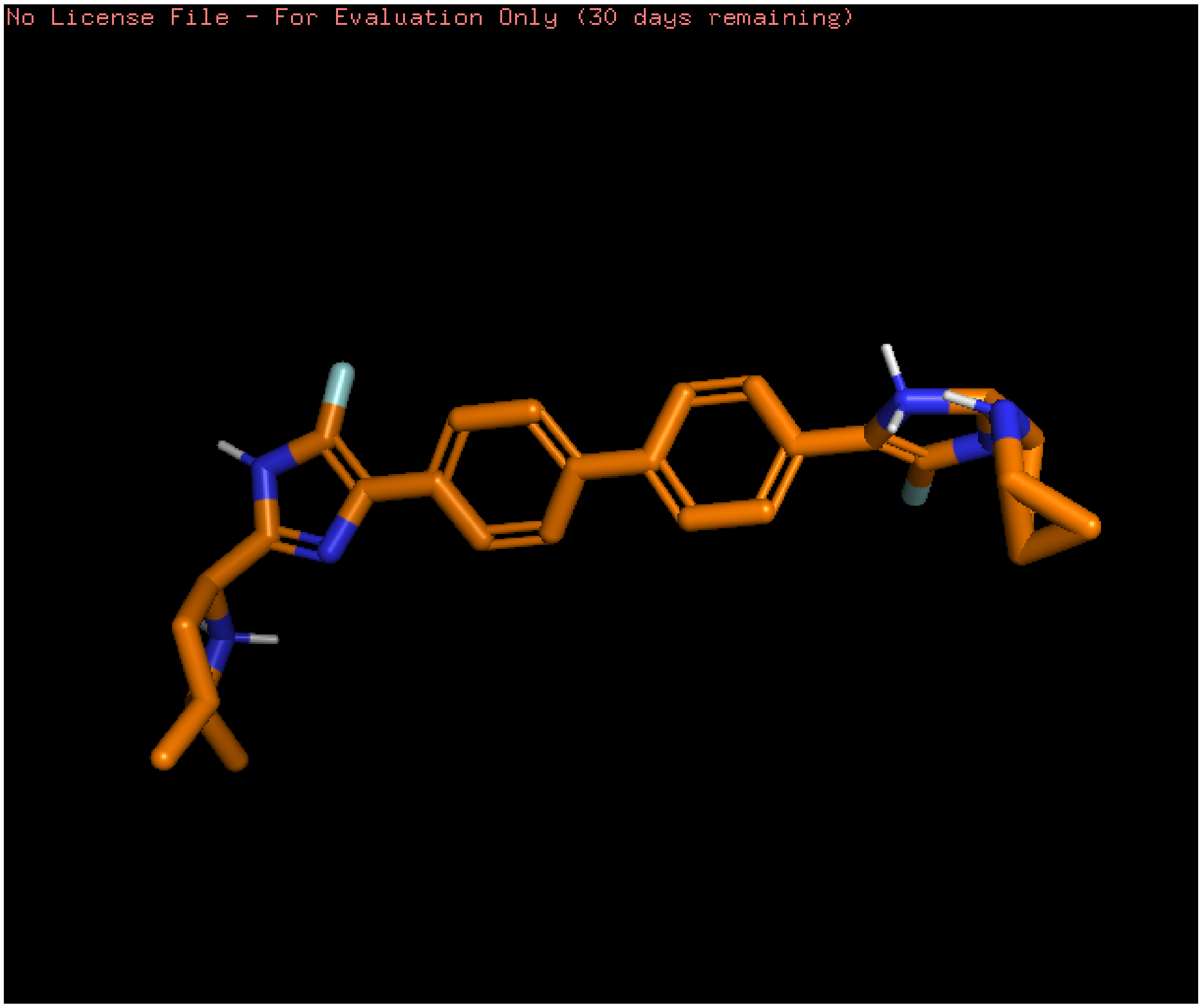
7701



8901



6843


No License File - For Evaluation Only (30 days remaining)