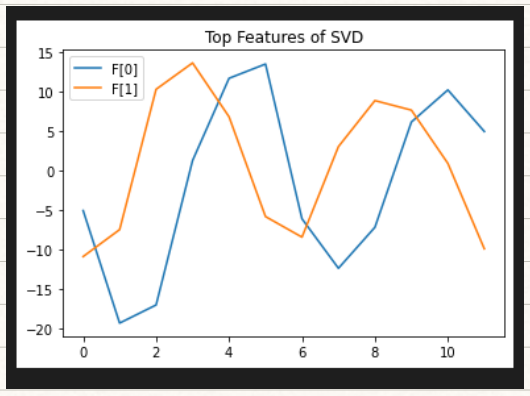
Hussam Djadi, HHD265, HW5 6SPDS

1. MEC score: 201

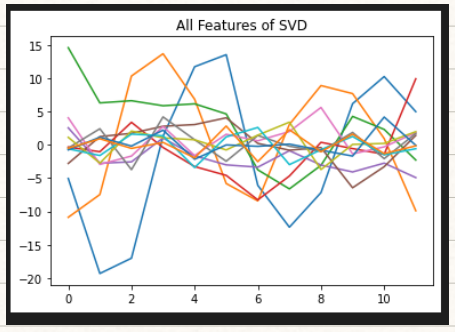
Code attached

2. a. These are the first two rows of $F = \Sigma V^T$

```
U: (678, 678)
S: (12, 12)
Vt: (12, 12)
F: (12, 12)
First 2 rows of F: [[ -5.04364799 -19.30436682 -17.00822924   1.30074591  11.72791997
   13.53743053  -6.07965326 -12.35554278  -7.15915187   6.18387842
   10.25206923   4.98817392]
 [-10.87058151  -7.45483727  10.33328024  13.67882688   6.85034916
   -5.80003526  -8.40851796   3.06883544   8.90183657   7.69342273
    0.94257836  -9.88651709]]
```

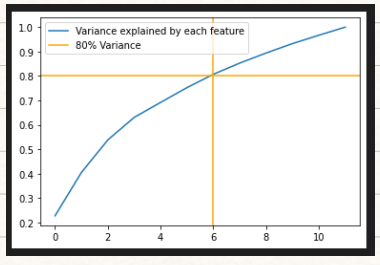


These are all the SVD features



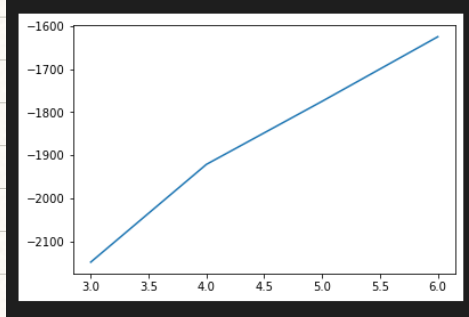
As we can see only the top two features have a noticeable pattern, and their pattern is also much more periodic than the others.

However they are not disproportionally important as 80% of the variance is explained by the top 6 features.

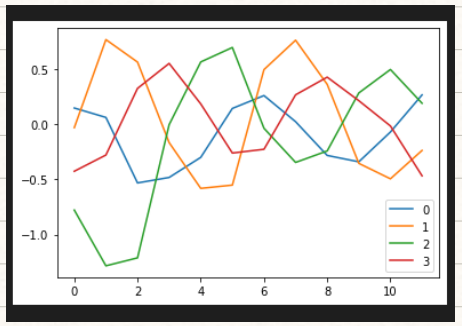


b. The evidence of one number of clusters is not immediately obvious from the score alone, however we see improvement start to diminish around 4 clusters. We also must take into account that more clusters usually improves accuracy regardless of the data.

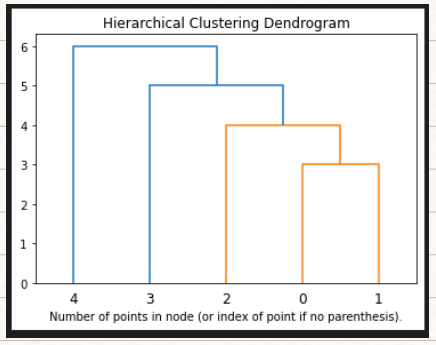
```
3: -2148.2997153988304
4: -1921.0651784420122
5: -1774.480851556624
6: -1624.4856017315446

[<matplotlib.lines.Line2D at 0x26fa8f00ee0>]
```

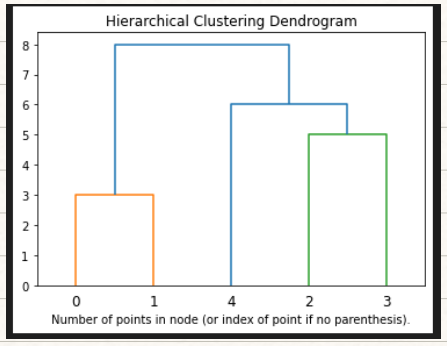


C. Again we see a strong periodicity trend that extends across all clustering methods. This suggests that the data itself is highly periodic.



3. Single Complete





4. My best interpretation of mapping gene regulation to a directed graph would be the following

a. Autoregulation → self-loop

b. Detect indirect autoregulation → find a cycle

c. Represent extent of regulation → weighted edges with positive and negative weights. Positive is promoting, negative is inhibiting

Open in Colab

In [ ]:
```
import numpy as np
```

In [ ]:
```
frag_mat = open("SNP_Fragment_Matrix.txt",'r')

hap_1 = {}
hap_2 = {}
R = np.zeros((80,194))
for line_num, line in enumerate(frag_mat):
  for char_index, char in enumerate(line.strip()):
    if char != "-":
      if char_index not in hap_1:
        hap_1[char_index] = char
        R[line_num,char_index] = 1
      if char_index in hap_1:
        if hap_1[char_index] == char:
          R[line_num,char_index] = 1
        else:
          R[line_num,char_index] = -1
          hap_2[char_index] = char
```

In [ ]:
```
haploid_mask = (R != 0).astype(np.int32)
(haploid_mask * R) == R
```

Out[ ]:
```
array([[ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       ...,
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True]])
```

In [ ]:
```
def f(haploid_mask,R,U,V):
  return np.linalg.norm(haploid_mask * (R - np.matmul(U, V.T)))

def argmin_F_U(haploid_mask, R, U, V):
  for row in range(U.shape[0]):
    U_copy = U.copy()
    U_copy[row] = U_copy[row] ^ 1
    f_orig = f(haploid_mask,R,U,V)
    f_mod = f(haploid_mask,R,U_copy,V)
    if f_mod < f_orig:
      U = U_copy
  return U
```

In [ ]:
```
alpha = 0.02
n = 80
m = 194
k = 2
#init U and V
U = np.random.randint(0,2,size = (n,1))
U = np.concatenate([U,np.bitwise_xor(U,1)],axis=1)

V = (np.random.randint(0,2,size = (m,1))*2-1).astype(np.int32)
V = np.concatenate([V,-V],axis=1)
```

```
for iteration in range(500):
  d_f = -2 * (haploid_mask * (R - U @ V.T)).T @ U
  V = V - alpha * d_f
  U = argmin_F_U(haploid_mask, R, U, V)
V = np.rint(V)
```

In [ ]:
```
# Initialize
V[0,1] = 1
```

In [ ]:
```
h1 = V.T[0]
h2 = V.T[1]
```

In [ ]:
```
s = ''
for i, j in enumerate(h2):
  if(j==1):
    s+=hap_1[i]
  if(j==-1):
    s+=hap_2[i]
```

In [ ]:
```
s = ''
for i in range(194):
  s += str(hap_1[i])
```

In [ ]:
```
def hd(row, h):
  d = 0
  for idx,i in enumerate(row):
    if(i != 0):
      if(i!=h[idx]):
        d+=1
  return d

def MEC(R,h1,h2):
  d = 0
  for row in R:
    d+=min(hd(row,h1),hd(row,h2))
  return d
```

In [ ]:
```
MEC(R,h1,h2)
```

Out[ ]:
```
204
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import seaborn as sns
```

Question #2

In [ ]:
```
# read csv
file_path = "2_YeastCycle.csv"
data = np.genfromtxt(file_path, delimiter=",")
```

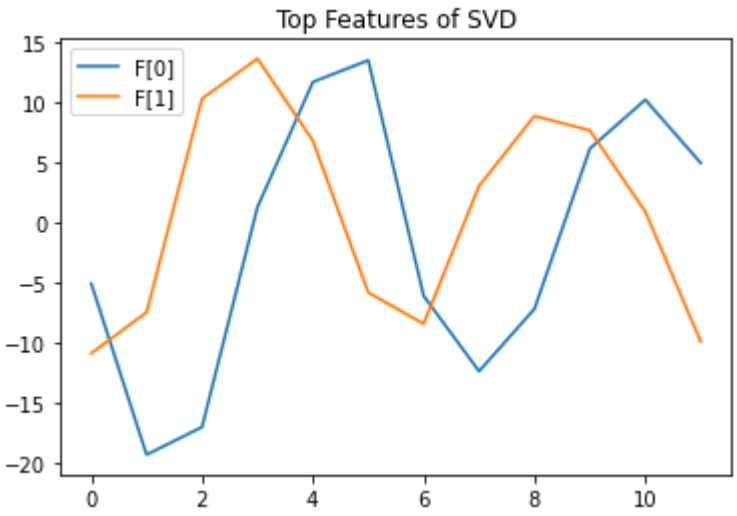
Part A

In [ ]:
```
# singular value decomp
svd = np.linalg.svd(data)

# obtain svd matrices
U = svd[0]
S_vec = svd[1]
S = S_vec * np.identity(S.shape[0])
Vt = svd[2]

# (i) plot first two rows of F = matmul(Σ, V_t)
F = np.matmul(S, Vt)

# print(f"Data: {data}")
print(f"U: {U.shape}")
print(f"S: {S.shape}")
# print(f"S: {S}")
print(f"Vt: {Vt.shape}")
# print(f"Vt: {Vt}")

print(f"F: {F.shape}")
print("First 2 rows of F:", F[:2])
```

```
U: (678, 678)
S: (12, 12)
Vt: (12, 12)
F: (12, 12)
First 2 rows of F: [[ -5.04364799 -19.30436682 -17.00822924   1.30074591  11.72791997
   13.53743053  -6.07965326 -12.35554278  -7.15915187   6.18387842
   10.25206923   4.98817392]
 [-10.87058151  -7.45483727  10.33328024  13.67882688   6.85034916
   -5.80003526  -8.40851796   3.06883544   8.90183657   7.69342273
    0.94257836  -9.88651709]]
```

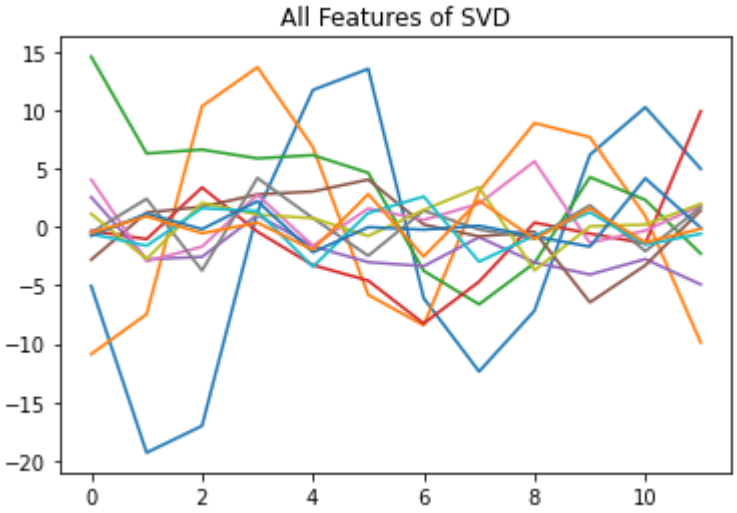
In [ ]:
```
plt.plot(F[0], label='F[0]')
plt.plot(F[1], label='F[1]')
# plt.plot(F[2], label='F[2]')
# plt.plot(F[3], label='F[3]')
plt.title("Top Features of SVD")
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x26faa6ba5e0>



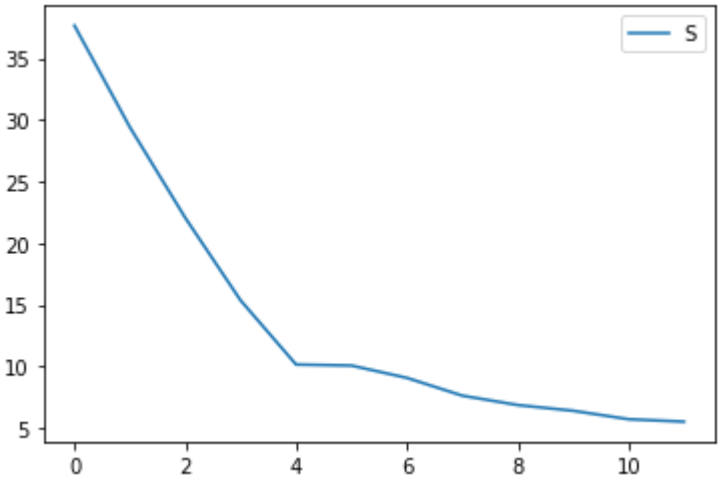
In [ ]:
```
for i in range(12):
        plt.plot(F[i])
plt.title("All Features of SVD")
```

Out[ ]: Text(0.5, 1.0, 'All Features of SVD')



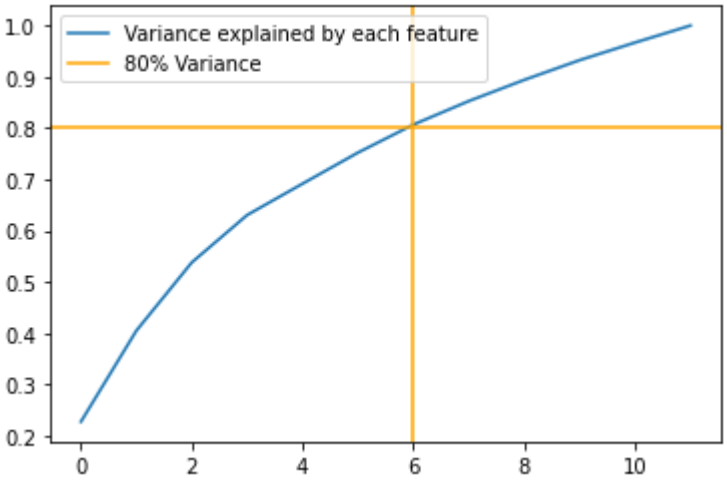
In [ ]:
```
plt.plot(S_vec, label = 'S')
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x26fa8b79ac0>



In [ ]:
```
plt.plot(np.cumsum(S_vec)/np.sum(S_vec), label='Variance explained by each feature')
plt.axhline(0.8, label='80% Variance', color='orange')
plt.axvline(6, color='orange')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x26fa8be2460>
```

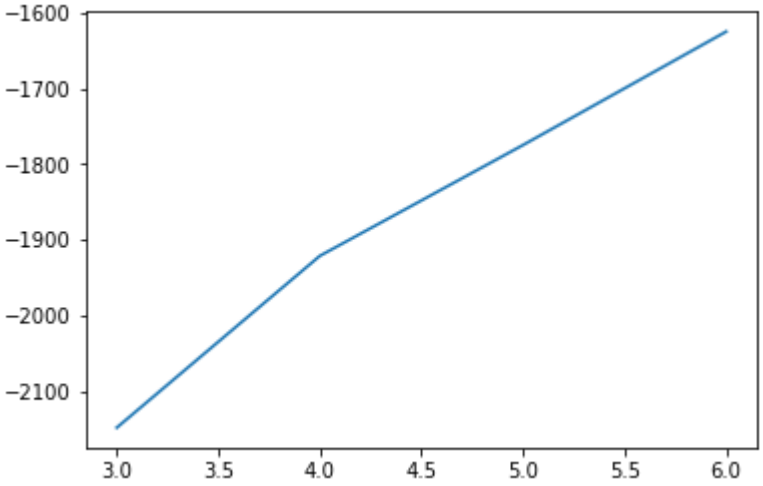


Part B

In [ ]:
```
k_score = []
for n_clusters in [3, 4, 5, 6]:
        kmeans = KMeans(n_clusters=n_clusters).fit(data)
        score = kmeans.score(data)
        k_score.append(score)
        print(f"{n_clusters}: {score}")

plt.plot(range(3,7), k_score)
```

```
3: -2148.2997153988304
4: -1921.0651784420122
5: -1774.480851556624
6: -1624.4856017315446
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x26fa8f00ee0>]
```

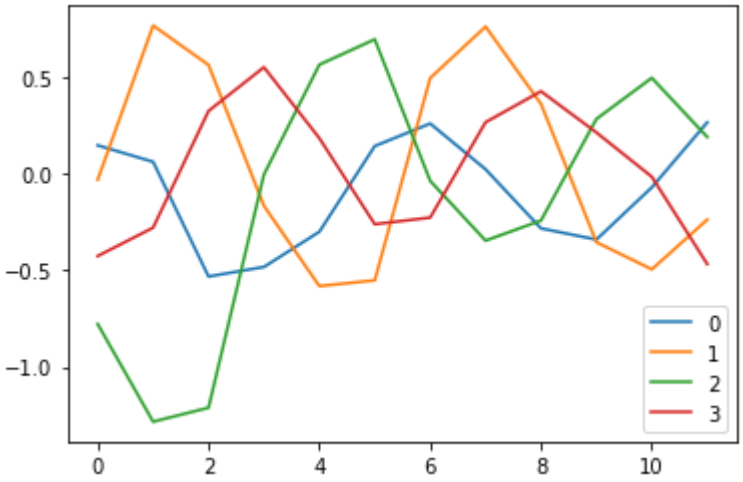


In [ ]:
```
k_score = []
n_clusters = 4

kmeans = KMeans(n_clusters=n_clusters)
cluster_indices = kmeans.fit_predict(data)

cluster_means = []

for cluster_name in range(n_clusters):
        cluster_means.append(np.mean(data[cluster_indices == cluster_name], axis=0))

print(cluster_means[0].shape)
```

```
(12,)
```

In [ ]:
```
for cluster_num, cluster in enumerate(cluster_means):
        plt.plot(cluster, label=cluster_num)
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x26faa4278b0>
```



In [ ]:

Heirarchical Clustering

https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/

In [ ]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

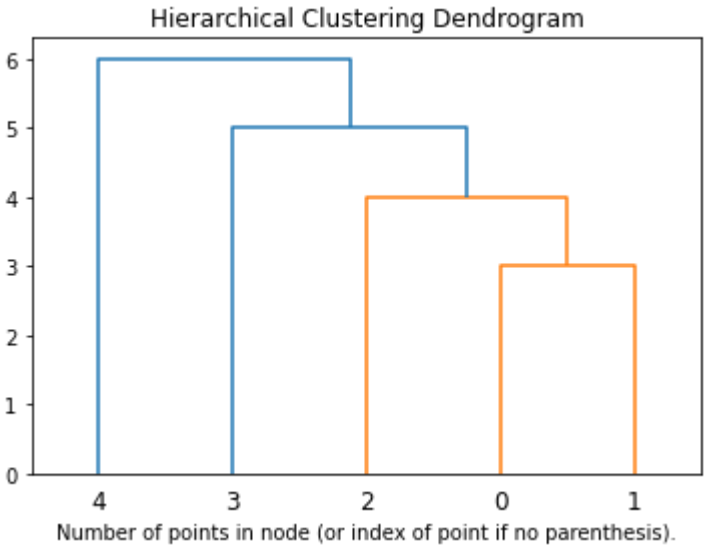
In [ ]:
```
dist_mat = [
        [0, 3, 8, 7, 8],
        [3, 0, 4, 8, 8],
        [8, 4, 0, 5, 6],
        [7, 8, 5, 0, 6],
        [8, 8, 6, 6, 0]
]
```

In [ ]:
```
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

Linkage = 'single'

In [ ]:
```
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None, affinity='precc
model = model.fit(dist_mat)
```

In [ ]:
```
plt.title("Hierarchical Clustering Dendrogram (Single Linkage)")
# plot the top three levels of the dendrogram
plot_dendrogram(model_single) # truncate_mode="level", p=3
plt.xlabel("Gene (0:a, 1:b, 2:c, 3:d, 4:e)")
plt.show()
```



Linkage = 'complete'

In [ ]:
```
model_complete = AgglomerativeClustering(distance_threshold=0, n_clusters=None, affini
model_complete = model_complete.fit(dist_mat)
```

In [ ]:
```
plt.title("Hierarchical Clustering Dendrogram (Complete Linkage)")
# plot the top three levels of the dendrogram
plot_dendrogram(model_complete) # truncate_mode="level", p=3
plt.xlabel("Gene (0:a, 1:b, 2:c, 3:d, 4:e)")
plt.show()
```