In [ ]:
```
### unsupervised learning
### Hierarchical Clustering - Agglomerative Clustering
```

In [2]:
```
import pandas as pd
```

In [3]:
```
import matplotlib.pyplot as plt
```

In [4]:
```
df = pd.read_excel(r"C:\Users\shubham lokare\OneDrive\Desktop\Hierarchical Clustering_Ha
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   UnivID    25 non-null     int64  
 1   Univ      25 non-null     object 
 2   State     25 non-null     object 
 3   SAT       24 non-null     float64
 4   Top10     25 non-null     int64  
 5   Accept    25 non-null     int64  
 6   SFRatio   24 non-null     float64
 7   Expenses  25 non-null     int64  
 8   GradRate  24 non-null     float64
dtypes: float64(3), int64(4), object(2)
memory usage: 1.9+ KB
```

In [7]:
```
df.head()
```

Out[7]:

	UnivID	Univ	State	SAT	Top10	Accept	SFRatio	Expenses	GradRate
0	1	Brown	RI	1310.0	89	22	13.0	22704	94.0
1	2	CalTech	CA	1415.0	100	25	6.0	63575	81.0
2	3	CMU	PA	1260.0	62	59	9.0	25026	72.0
3	4	Columbia	NY	1310.0	76	24	12.0	31510	NaN
4	5	Cornell	NY	1280.0	83	33	13.0	21864	90.0

In [8]:
```
df.describe()
```

Out[8]:

	UnivID	SAT	Top10	Accept	SFRatio	Expenses	GradRate
count	25.000000	24.000000	25.000000	25.000000	24.000000	25.000000	24.000000
mean	13.000000	1266.916667	76.480000	39.200000	12.708333	27388.000000	86.666667
std	7.359801	110.663578	19.433905	19.727308	4.154402	14424.883165	9.248580
min	1.000000	1005.000000	28.000000	14.000000	6.000000	8704.000000	67.000000
25%	7.000000	1236.250000	74.000000	24.000000	10.750000	15140.000000	80.750000
50%	13.000000	1287.500000	81.000000	36.000000	12.000000	27553.000000	90.000000
75%	19.000000	1345.000000	90.000000	50.000000	14.250000	34870.000000	94.000000
max	25.000000	1415.000000	100.000000	90.000000	25.000000	63575.000000	97.000000

In [9]:
```python
# Data Preprocessing
# **Cleaning Unwanted columns**
# UnivID is the identity to each university.
# Analytically it does not have any value (Nominal data).
# We can safely ignore the ID column by dropping the column.

df.drop(['UnivID'], axis = 1, inplace = True)
```

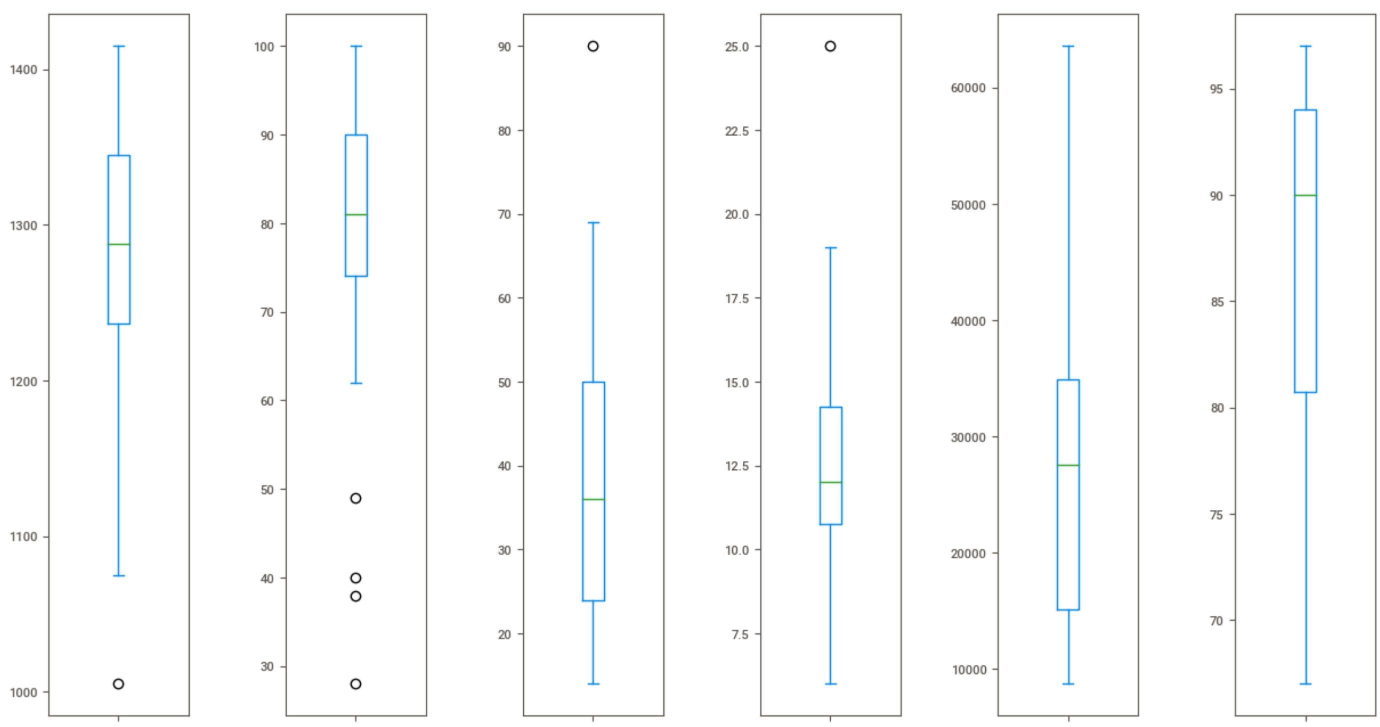
In [13]:
```python
## Auto EDA
import sweetviz as sw

sv = sw.analyze(df)
sv.show_html()
```

```
|
…
Report SWEETVIZ_REPORT.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not
pop up, regardless, the report IS saved in your notebook/colab files.
```

In [14]:
```python
# we check missinh values
df.isna().sum()
```

Out[14]:
```
Univ        0
State       0
SAT         1
Top10       0
Accept      0
SFRatio     1
Expenses    0
GradRate    1
dtype: int64
```

In [15]:
```python
## check outliers
# Boxplot
df.plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))
plt.subplots_adjust(wspace = 0.70) # ws is the width of the padding between subplots, as
plt.show()
```



In [16]:
```python
## use AUtO EDA Library to clean the data
from AutoClean import AutoClean
```

```python
clean_pipeline = AutoClean(df.iloc[:, 1:], mode = 'manual', missing_num = 'auto',
                           outliers = 'winz', encode_categ = ['auto'])
```

```
AutoClean process completed in 0.188648 seconds
Logfile saved to: C:\Users\shubham lokare\autoclean.log
```

In [17]:
```python
df_clean = clean_pipeline.output
```

In [19]:
```python
df_clean.head()
```

Out[19]:

	State	SAT	Top10	Accept	SFRatio	Expenses	GradRate	State_lab
0	RI	1310	89	22	13	22704	94	13
1	CA	1415	100	25	6	63575	81	0
2	PA	1260	62	59	9	25026	72	12
3	NY	1310	76	24	12	31510	92	11
4	NY	1280	83	33	13	21864	90	11

In [21]:
```python
## remove State column
df_clean.drop(['State'] ,axis = 1 , inplace = True)
```

In [25]:
```python
df_clean.head()
```

Out[25]:

	SAT	Top10	Accept	SFRatio	Expenses	GradRate	State_lab
0	1310	89	22	13	22704	94	13
1	1415	100	25	6	63575	81	0
2	1260	62	59	9	25026	72	12
3	1310	76	24	12	31510	92	11
4	1280	83	33	13	21864	90	11

In [26]:
```python
# ## Normalization/MinMax Scaler - To address the scale differences

# ### Python Pipelines
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler
```

In [27]:
```python
pipeline = make_pipeline(MinMaxScaler())
```

In [30]:
```python
# Train the data preprocessing pipeline on data

df_pipelined = pd.DataFrame(pipeline.fit_transform(df_clean), index = df_clean.index)
```

In [31]:
```python
df_pipelined.head()
```

Out[31]:

	0	1	2	3	4	5	6
0	0.676923	0.78	0.106667	0.500000	0.255144	0.900000	0.8125
1	1.000000	1.00	0.146667	0.000000	1.000000	0.466667	0.0000
2	0.523077	0.24	0.600000	0.214286	0.297461	0.166667	0.7500
3	0.676923	0.52	0.133333	0.428571	0.415629	0.833333	0.6875
4	0.584615	0.66	0.253333	0.500000	0.239835	0.766667	0.6875

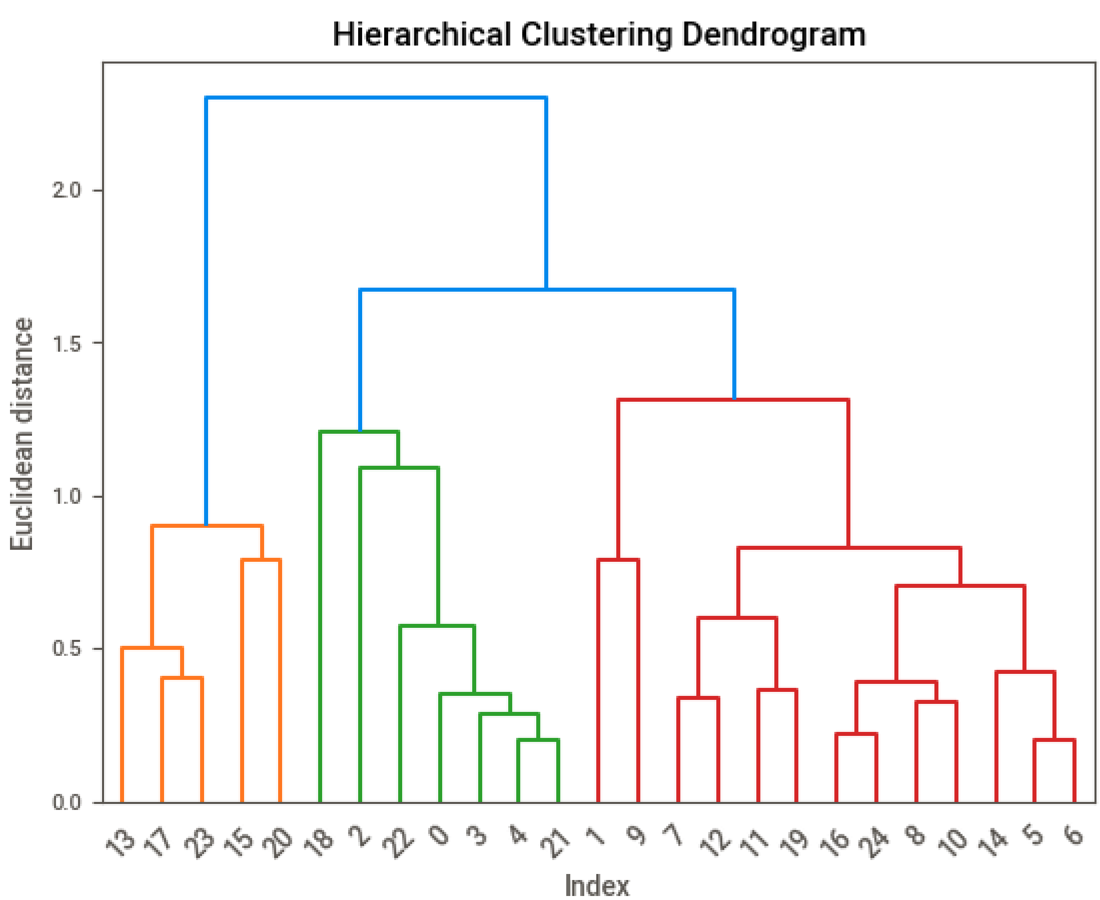
In [32]:
```
df_pipelined.describe()
```

Out[32]:

	0	1	2	3	4	5	6
count	25.000000	25.000000	25.000000	25.000000	25.000000	25.000000	25.000000
mean	0.559631	0.565600	0.335467	0.454286	0.340508	0.662667	0.457500
std	0.303341	0.312811	0.261610	0.256613	0.262887	0.303882	0.316495
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.461538	0.480000	0.133333	0.285714	0.117293	0.466667	0.187500
50%	0.600000	0.620000	0.293333	0.428571	0.343515	0.766667	0.437500
75%	0.769231	0.800000	0.480000	0.571429	0.476864	0.900000	0.750000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [ ]:
```
## Model Building

# CLUSTERING MODEL BUILDING
# Hierarchical Clustering - Agglomerative Clustering
```

In [33]:
```
## plot Dendrogram

from scipy.cluster.hierarchy import linkage, dendrogram
tree_plot = dendrogram(linkage(df_pipelined, method  = "complete"))

plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Index')
plt.ylabel('Euclidean distance')
plt.show()
```



In [34]:
```
## After seen dendrogram we decide to choice 3 cluster
from sklearn.cluster import AgglomerativeClustering
```

In [35]:
```
hcluster = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'co

hcluster1= hcluster.fit_predict(df_pipelined)
hcluster1
```

Out[35]:
```
array([2, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 2, 0, 1, 2,
       2, 1, 0], dtype=int64)
```

In [36]:
```
hcluster.labels_
```

Out[36]:
```
array([2, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 2, 0, 1, 2,
       2, 1, 0], dtype=int64)
```

In [39]:
```
## change labels into series
cluster_labels = pd.Series(hcluster.labels_)
```

In [40]:
```
## # Combine the labels obtained with the data

df_clust = pd.concat([cluster_labels, df_clean], axis = 1)
```

In [41]:
```
df_clust.head()
```

Out[41]:

	0	SAT	Top10	Accept	SFRatio	Expenses	GradRate	State_lab
0	2	1310	89	22	13	22704	94	13
1	0	1415	100	25	6	63575	81	0
2	2	1260	62	59	9	25026	72	12
3	2	1310	76	24	12	31510	92	11
4	2	1280	83	33	13	21864	90	11

In [42]:
```
df_clust.columns
df_clust = df_clust.rename(columns = {0: 'cluster'})
df_clust.head()
```

Out[42]:

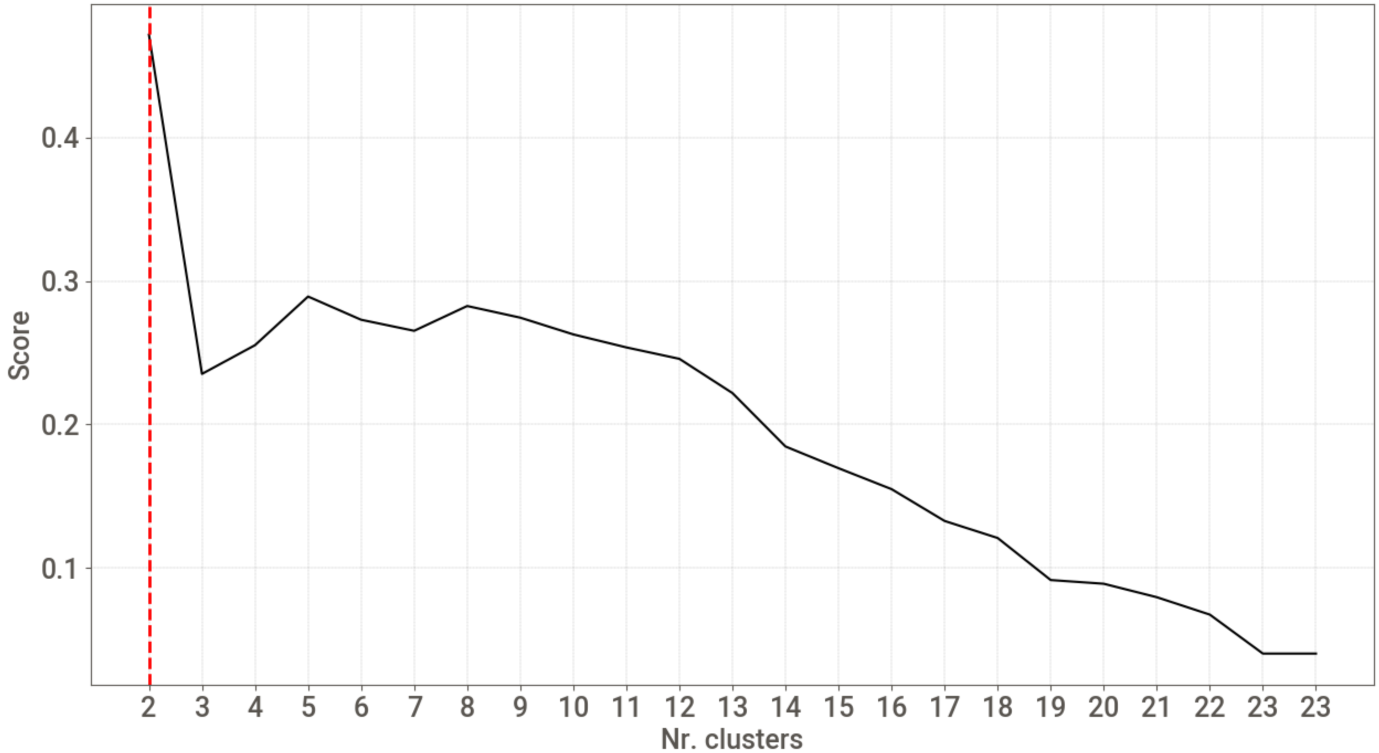
	cluster	SAT	Top10	Accept	SFRatio	Expenses	GradRate	State_lab
0	2	1310	89	22	13	22704	94	13
1	0	1415	100	25	6	63575	81	0
2	2	1260	62	59	9	25026	72	12
3	2	1310	76	24	12	31510	92	11
4	2	1280	83	33	13	21864	90	11

In [ ]:
```
#  Clusters Evaluation

# Silhouette coefficient:
# Silhouette coefficient is a Metric, which is used for calculating
# goodness of the clustering technique, and the value ranges between (-1 to +1).
# It tells how similar an object is to its own cluster (cohesion) compared to
# other clusters (separation).
# A score of 1 denotes the best meaning that the data point is very compact
# within the cluster to which it belongs and far away from the other clusters.
# Values near 0 denote overlapping clusters.
```

In [43]:
```python
from sklearn import metrics
metrics.silhouette_score(df_pipelined, cluster_labels)
```
Out[43]:
```
0.24924028962347694
```
In [45]:
```python
# **Calinski Harabasz:**
# Higher value of the CH index means clusters are well separated.
# There is no thumb rule which is an acceptable cut-off value.

metrics.calinski_harabasz_score(df_pipelined, cluster_labels)
```
Out[45]:
```
16.978378266346144
```
In [46]:
```python
# **Davies-Bouldin Index:**
# Unlike the previous two metrics, this score measures the similarity of clusters.
# The lower the score the better the separation between your clusters.
# Vales can range from zero and infinity

metrics.davies_bouldin_score(df_pipelined, cluster_labels)
```
Out[46]:
```
1.2808172915766176
```
In [47]:
```python
##  Experiment to obtain the best clusters by altering the parameters

## help of clusteval library

from clusteval import clusteval
import numpy as np
```
In [50]:
```python
# Silhouette cluster evaluation.
best_cluster = clusteval(evaluate = 'silhouette')

df_array = np.array(df_pipelined)
```
In [51]:
```python
# Fit
best_cluster.fit(df_array)
```
```
[clusteval] >INFO> Saving data in memory.
[clusteval] >INFO> Fit with method=[agglomerative], metric=[euclidean], linkage=[ward]
[clusteval] >INFO> Evaluate using silhouette.
[clusteval] >INFO: 100%|████████████████████████████████████████████████████████████████
██████████████████████████████████| 23/23 [00:00<00:00, 508.91it/s]
[clusteval] >INFO> Compute dendrogram threshold.
[clusteval] >INFO> Optimal number clusters detected: [2].
[clusteval] >INFO> Fin.
```

Out[51]:
```
{'evaluate': 'silhouette',
 'score':     cluster_threshold  clusters     score
 0                   2         2  0.471638
 1                   3         3  0.235200
 2                   4         4  0.255234
 3                   5         5  0.289120
 4                   6         6  0.272932
 5                   7         7  0.265237
 6                   8         8  0.282524
 7                   9         9  0.274408
 8                  10        10  0.262729
 9                  11        11  0.253672
 10                 12        12  0.245703
 11                 13        13  0.221844
 12                 14        14  0.184605
 13                 15        15  0.169466
 14                 16        16  0.154874
 15                 17        17  0.132693
 16                 18        18  0.120831
 17                 19        19  0.091511
 18                 20        20  0.088930
 19                 21        21  0.079554
 20                 22        22  0.067415
 21                 23        23  0.040262
 22                 24        23  0.040262,
 'labx': array([1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
        1, 0, 1]),
 'fig': {'silscores': array([0.47163809, 0.23519988, 0.25523379, 0.28912046, 0.27293166,
         0.26523697, 0.28252416, 0.27440766, 0.26272852, 0.25367218,
         0.24570312, 0.22184431, 0.18460494, 0.16946642, 0.15487358,
         0.13269309, 0.12083052, 0.09151137, 0.08893041, 0.07955423,
         0.06741495, 0.04026216, 0.04026216]),
  'sillclust': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 1
8,
         19, 20, 21, 22, 23, 23]),
  'clustcutt': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 1
8,
         19, 20, 21, 22, 23, 24])},
 'max_d': 2.771903720145042,
 'max_d_lower': 1.7241946557650798,
 'max_d_upper': 3.819612784525004}
```

In [52]:
```
# plot
best_cluster.plot()
```



Out[52]:
```
(<Figure size 1500x800 with 1 Axes>,
 <Axes: xlabel='Nr. clusters', ylabel='Score'>)
```

In [55]:
```
## with help of that we decide 2 cluster is best because of its silhouette score maximum
## Using the report from clusteval library building 2 clusters
# Fit using agglomerativeClustering with metrics: euclidean, and linkage: ward

hcluster2 = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage = 'w

hicluster= hcluster2.fit_predict(df_pipelined)
```

In [56]:
```
hcluster2.labels_
```

Out[56]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0], dtype=int64)
```

In [57]:
```
cluster_labels2 = pd.Series(hcluster2.labels_)
```

In [58]:
```
# Concate the Results with data
df_cluster2 = pd.concat([cluster_labels2, df_clean], axis = 1)

df_cluster2 = df_cluster2.rename(columns = {0:'cluster'})
df_cluster2.head()
```

Out[58]:

	cluster	SAT	Top10	Accept	SFRatio	Expenses	GradRate	State_lab
0	0	1310	89	22	13	22704	94	13
1	0	1415	100	25	6	63575	81	0
2	1	1260	62	59	9	25026	72	12
3	0	1310	76	24	12	31510	92	11
4	0	1280	83	33	13	21864	90	11

In [ ]: