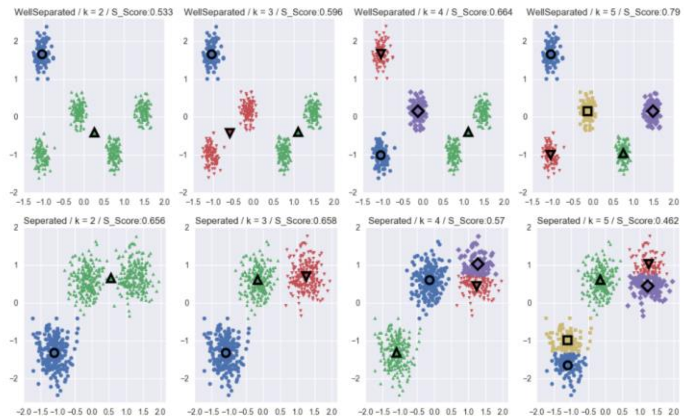
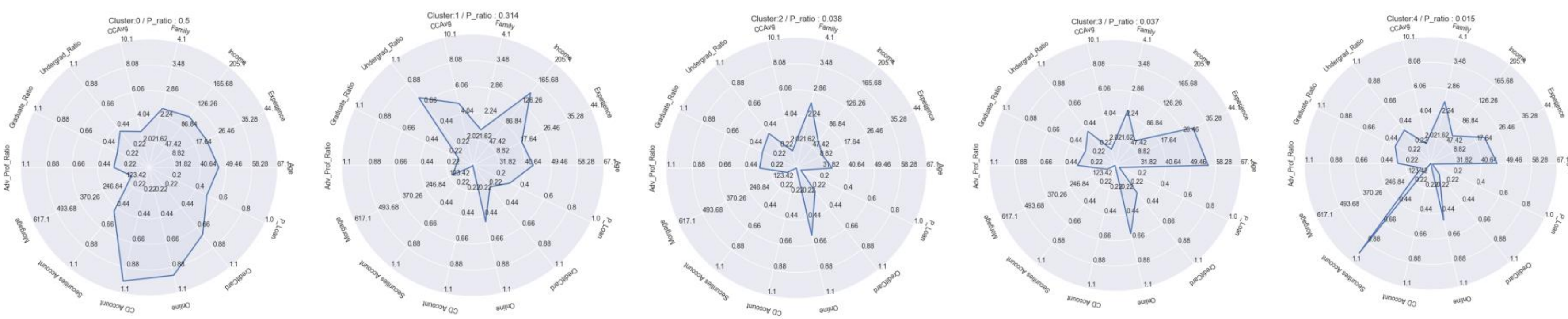
# Clustering 실습

# 클러스터링 실습

- **클러스터링 실습1 - 2D 인공데이터**
  - 3가지의 인공 데이터를 기반으로 배운 클러스터링 기법을 이해해 봄


- **클러스터링 실습2 – Personal Loan**
  - 실습목표 : Personal Loan 데이터를 기반으로 silhouette score기반 최적의 k-Means clustering 생성 후 해석



# 클러스터링 실습1 – 2D 인공데이터

▪ 3가지 사용할 데이터 불러오며 기본정보확인/정규화를 수행하고, 각 데이터를 시각화 함

먼저 모든 데이터에 대하여 standardization 한 후의 데이터의 구조를 살펴봄

In [2]:
```
def standardization(Data):
    Input = ((Data[['X', 'Y']] - np.mean(Data[['X', 'Y']], axis=0)) / np.std(Data[['X', 'Y']], axis=0))
    return(pd.concat([Input, Data['Class']], axis=1))


WellSeparated = standardization(pd.read_csv(os.getcwd()+'/Dataset/wellseparated.csv'))
Twomoon = standardization(pd.read_csv(os.getcwd()+'/Dataset/Twomoon.csv'))
Seperated = standardization(pd.read_csv(os.getcwd()+'/Dataset/Seperated.csv'))

Artificial_Dataset={'WellSeparated':WellSeparated,  'Twomoon': Twomoon,'Seperated':Seperated}

def Data_Info(Data,NAME):
    print(NAME ,": ",np.shape(Data)[0],"/ Class : ",len(collections.Counter(np.array(Data)[:,2])))

print("각각의 2차원의 데이터 갯수는 아래와 같음")
for i in range(len(Artificial_Dataset)):
    Data_Info(Artificial_Dataset[list(Artificial_Dataset.keys())[i]],list(Artificial_Dataset.keys())[i])
```

```
각각의 2차원의 데이터 갯수는 아래와 같음
Seperated :  600 / Class :  3
WellSeparated :  500 / Class :  5
Twomoon :  600 / Class :  2
```

In [3]:
```
def Simple_Scatter(Data,Name):
    G=sns.pairplot(x_vars=['X'], y_vars=['Y'], data=Data, hue="Class", size=3)
    G.fig.suptitle("Data : "+Name, fontsize=10, color='black', alpha=0.8)

fig, axes = plt.subplots(1,3,figsize=(15,4))
for i in range(len(Artificial_Dataset)):
    Data=Artificial_Dataset[list(Artificial_Dataset.keys())[i]]
    mglearn.discrete_scatter(Data['X'], Data['Y'], Data['Class'], ax=axes[i], s=5)
    axes[i].set_title("Data:" + list(Artificial_Dataset.keys())[i])
```

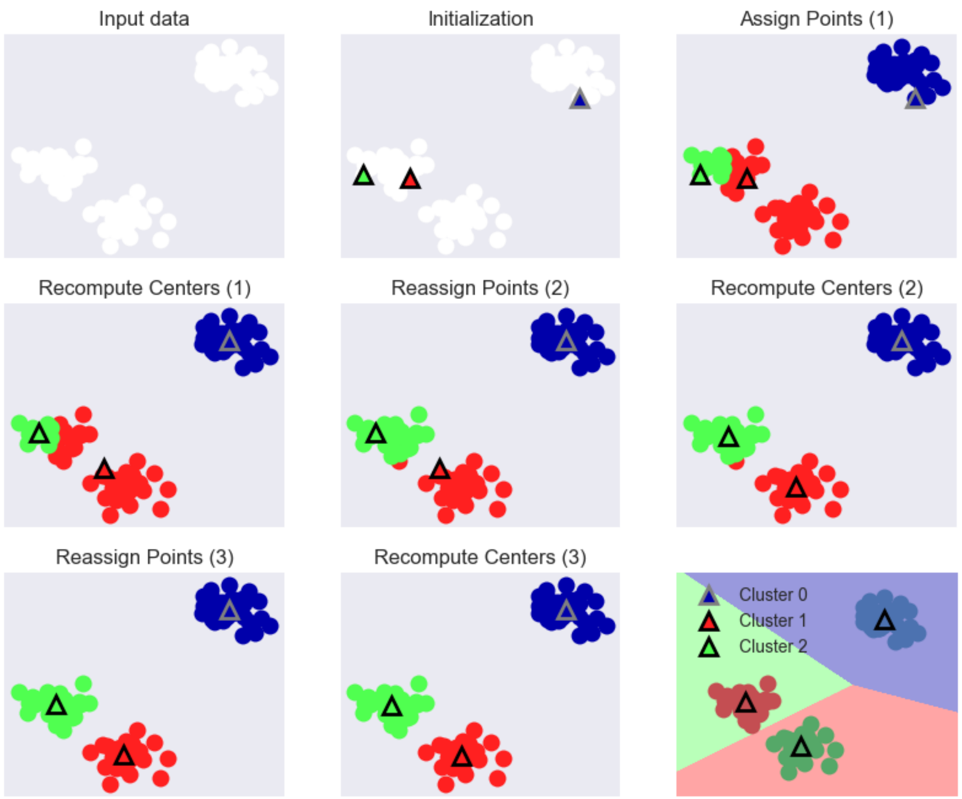
# 클러스터링 실습1 – 2D 인공데이터

▪ **k-Means clustering의 간단한 도식화를 통해 메커니즘 확인**

## Algorithm1 : k-Means clustering

**다시 한번 k-Means 클러스터링의 메커니즘을 위해 간단한 도식화를 확인하자**

In [4]:
```
mglearn.plots.plot_kmeans_algorithm()
mglearn.plots.plot_kmeans_boundaries()
```



# 클러스터링 실습1 – 2D 인공데이터

- **k-Means clustering visualization을 위하여 함수 생성**

**k-Means Visualization을 위해 함수생성**

In [6]:
```
def k_Means_Plot(Data,Select_k,NAME,Init_Method='k-means++',Num_Init=10):
    Data2 = Data[['X', 'Y']]
    fig, axes = plt.subplots(1, (np.max(list(Select_k))-np.min(list(Select_k)))+1, figsize=(15, 4))
    for i in Select_k:
        Kmeans_Clustering = KMeans(n_clusters=i,init=Init_Method,random_state=100,n_init=Num_Init)
        Kmeans_Clustering.fit(Data2)
        mglearn.discrete_scatter(Data2['X'], Data2['Y'], Kmeans_Clustering.labels_, ax=axes[i - 2], s=5)
        mglearn.discrete_scatter(Kmeans_Clustering.cluster_centers_[:, 0], Kmeans_Clustering.cluster_centers_[:, 1],
                                 list(range(i)), markeredgewidth=3, ax=axes[i - 2], s=10)
        Score=np.round(silhouette_score(Data2,Kmeans_Clustering.labels_),3)
        axes[i - 2].set_title( NAME + ' / k = ' + str(i)+' / S_Score:'+str(Score))
```

# 클러스터링 실습1 – 2D 인공데이터

- 사용할 2D 데이터셋을 객체로 만들어줌
- 여러장의 플롯을 위하여 plt.subplot을 사용

**k-Means Visualization을 위해 함수생성**

In [6]:
```
def k_Means_Plot(Data,Select_k,NAME,Init_Method='k-means++',Num_Init=10):
    Data2 = Data[['X', 'Y']]
    fig, axes = plt.subplots(1, (np.max(list(Select_k))-np.min(list(Select_k)))+1, figsize=(15, 4))
    for i in Select_k:
        Kmeans_Clustering = KMeans(n_clusters=i,init=Init_Method,random_state=100,n_init=Num_Init)
        Kmeans_Clustering.fit(Data2)
        mglearn.discrete_scatter(Data2['X'], Data2['Y'], Kmeans_Clustering.labels_, ax=axes[i - 2], s=5)
        mglearn.discrete_scatter(Kmeans_Clustering.cluster_centers_[:, 0], Kmeans_Clustering.cluster_centers_[:, 1],
                                 list(range(i)), markeredgewidth=3, ax=axes[i - 2], s=10)
        Score=np.round(silhouette_score(Data2,Kmeans_Clustering.labels_),3)
        axes[i - 2].set_title( NAME + ' / k = ' + str(i)+' / S_Score:'+str(Score))
```

# 클러스터링 실습1 – 2D 인공데이터

- **사용자가 지정한 k값의 범위만큼 반복수행**

**k-Means Visualization을 위해 함수생성**

In [6]:
```
def k_Means_Plot(Data,Select_k,NAME,Init_Method='k-means++',Num_Init=10):
    Data2 = Data[['X', 'Y']]
    fig, axes = plt.subplots(1, (np.max(list(Select_k))-np.min(list(Select_k)))+1, figsize=(15, 4))
    for i  in Select_k:
        Kmeans_Clustering = KMeans(n_clusters=i,init=Init_Method,random_state=100,n_init=Num_Init)
        Kmeans_Clustering.fit(Data2)
        mglearn.discrete_scatter(Data2['X'], Data2['Y'], Kmeans_Clustering.labels_, ax=axes[i - 2], s=5)
        mglearn.discrete_scatter(Kmeans_Clustering.cluster_centers_[:, 0], Kmeans_Clustering.cluster_centers_[:, 1],
                                 list(range(i)), markeredgewidth=3, ax=axes[i - 2], s=10)
        Score=np.round(silhouette_score(Data2,Kmeans_Clustering.labels_),3)
        axes[i - 2].set_title( NAME + ' / k = ' + str(i)+' / S_Score:'+str(Score))
```

# 클러스터링 실습1 – 2D 인공데이터

- **실제로 sklearn에서 사용하는  k-means 함수는 KMeans 함수임**

**k-Means Visualization을 위해 함수생성**

In [6]:
```
def k_Means_Plot(Data,Select_k,NAME,Init_Method='k-means++',Num_Init=10):
    Data2 = Data[['X', 'Y']]
    fig, axes = plt.subplots(1, (np.max(list(Select_k))-np.min(list(Select_k)))+1, figsize=(15, 4))
    for i in Select_k:
        Kmeans_Clustering = KMeans(n_clusters=i,init=Init_Method,random_state=100,n_init=Num_Init)
        Kmeans_Clustering.fit(Data2)
        mglearn.discrete_scatter(Data2['X'], Data2['Y'], Kmeans_Clustering.labels_, ax=axes[i - 2], s=5)
        mglearn.discrete_scatter(Kmeans_Clustering.cluster_centers_[:, 0], Kmeans_Clustering.cluster_centers_[:, 1],
                                 list(range(i)), markeredgewidth=3, ax=axes[i - 2], s=10)
        Score=np.round(silhouette_score(Data2,Kmeans_Clustering.labels_),3)
        axes[i - 2].set_title( NAME + ' / k = ' + str(i)+' / S_Score:'+str(Score))
```

## sklearn.cluster.KMeans

*class* sklearn.cluster. KMeans (*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto'*) [source]

http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

# Sklearn.cluster.KMeans

- 본 코드에서 사용하는 Kmeans의 하이터 파라미터의 대한 설명은 아래와 같음

## sklearn.cluster.KMeans

*class* sklearn.cluster. KMeans (*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto'*) [source]

**n_clusters** : int, optional, default: 8

The number of clusters to form as well as the number of centroids to generate.

**init** : {'k-means++', 'random' or an ndarray}

Method for initialization, defaults to 'k-means++':

'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.

'random': choose k observations (rows) at random from data for the initial centroids.

If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

**n_init** : int, default: 10

Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

- **n_clusters**

➔ k-Means의 군집 개수를 설정

- **Init (k-means++, random)**

➔ k-Means의 초기화 방법을 설정함
➔ 'random'의 경우는 임의적으로 초기중심값을 할당
➔ 'k-Means++'의 경우는 초기값을 좀 더 효율적으로 주기 위한 방법임 (2007)

**2.2 The k-means++ algorithm**

We propose a specific way of choosing centers for the **k-means** algorithm. In particular, let $D(x)$ denote the shortest distance from a data point to the closest center we have already chosen. Then, we define the following algorithm, which we call **k-means++**.

1a. Take one center $c_1$, chosen uniformly at random from $\mathcal{X}$.
1b. Take a new center $c_i$, choosing $x \in \mathcal{X}$ with probability $\frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2}$.
1c. Repeat Step 1b. until we have taken $k$ centers altogether.
2-4. Proceed as with the standard **k-means** algorithm.

Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

# Sklearn.cluster.KMeans

- **본 코드에서 사용하는 Kmeans의 하이터 파라미터의 대한 설명은 아래와 같음**

## sklearn.cluster.KMeans

*class* sklearn.cluster.KMeans(*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto'*) [source]

**n_clusters** : int, optional, default: 8

The number of clusters to form as well as the number of centroids to generate.

**init** : {'k-means++', 'random' or an ndarray}

Method for initialization, defaults to 'k-means++':

'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.

'random': choose k observations (rows) at random from data for the initial centroids.

If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

**n_init** : int, default: 10

Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

- **n_init**

➔ k-Means를 다른 초기값을 이용하여 클러스터링을 몇 번을 반복시행할 것인지를 의미 최종 값은 within-cluster sum-of-sqaures의 최적값으로 결정함

http://scikit-learn.org/stable/modules/clustering.html#k-means

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_j - \mu_i||^2)$$

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares.

http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding."
*Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

# 클러스터링 실습1 – 2D 인공데이터

- 각 k값에 대한
  (1) k-Means clustering scatter plot
  (2) k-Means clustering 중심 시각화
  (3) k-Means clustering silhouette score 도출
  (4) 각각의 시각화에 대하여 제목을 입력

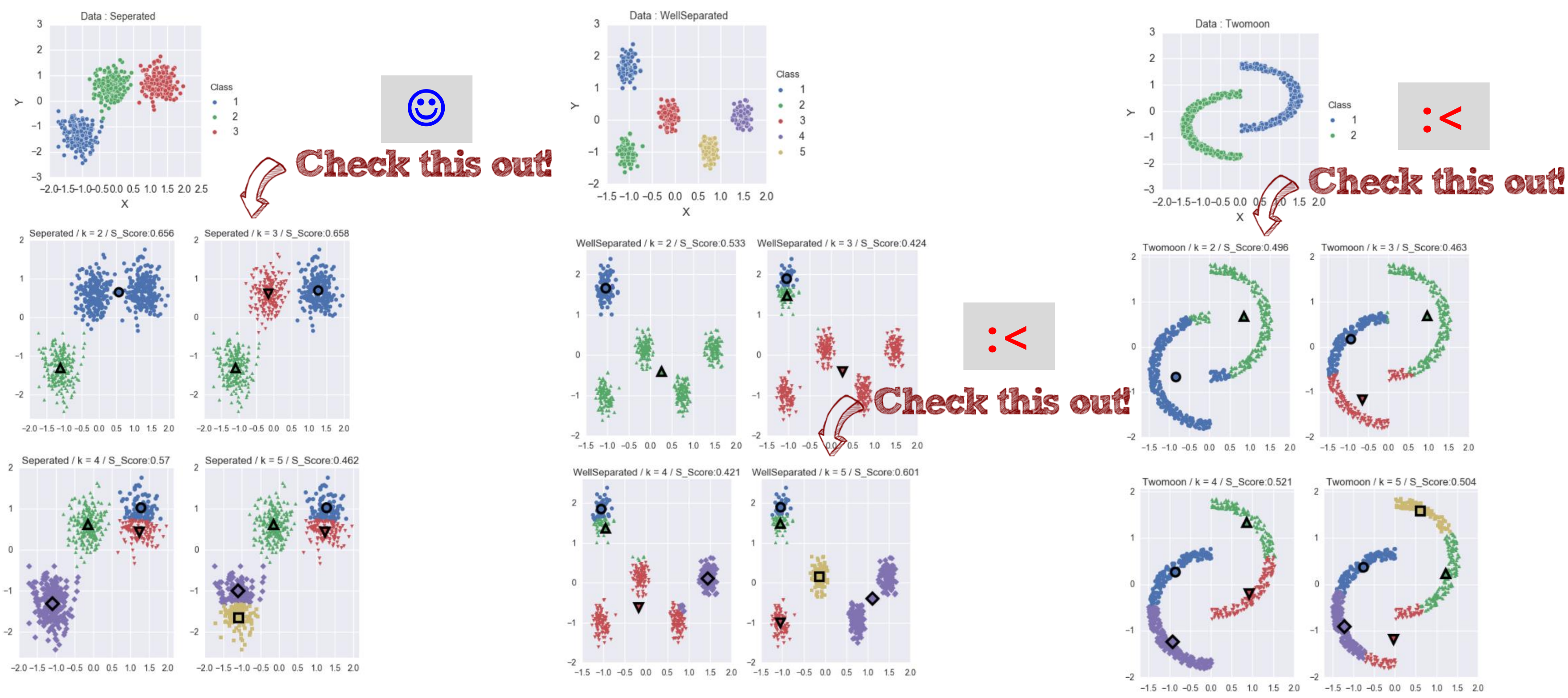
**k-Means Visualization을 위해 함수생성**

In [6]:
```python
def k_Means_Plot(Data,Select_k,NAME,Init_Method='k-means++',Num_Init=10):
    Data2 = Data[['X', 'Y']]
    fig, axes = plt.subplots(1, (np.max(list(Select_k))-np.min(list(Select_k)))+1, figsize=(15, 4))
    for i  in Select_k:
        Kmeans_Clustering = KMeans(n_clusters=i,init=Init_Method,random_state=100,n_init=Num_Init)
        Kmeans_Clustering.fit(Data2)
        mglearn.discrete_scatter(Data2['X'], Data2['Y'], Kmeans_Clustering.labels_, ax=axes[i - 2], s=5)
        mglearn.discrete_scatter(Kmeans_Clustering.cluster_centers_[:, 0], Kmeans_Clustering.cluster_centers_[:, 1],
                                 list(range(i)), markeredgewidth=3, ax=axes[i - 2], s=10)
        Score=np.round(silhouette_score(Data2,Kmeans_Clustering.labels_),3)
        axes[i - 2].set_title( NAME + ' / k = ' + str(i)+' / S_Score:'+str(Score))
```

# 클러스터링 실습1 – 2D 인공데이터

- **Initialization method = ‘random’ , Num_init=1**

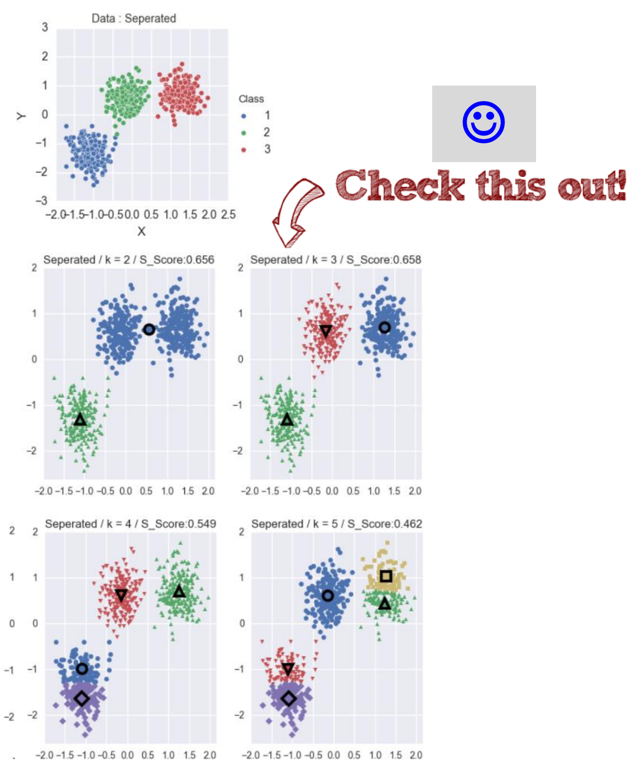
```
for i in range(0,3):
    Simple_Scatter(Artificial_Dataset[list(Artificial_Dataset.keys())[i]],list(Artificial_Dataset.keys())[i])
    k_Means_Plot(Data=Artificial_Dataset[list(Artificial_Dataset.keys())[i]] ,Select_k=range(2, 6),NAME=list(Artificial_Dataset.keys())[i],
                 Init_Method='random',Num_Init=1)
```

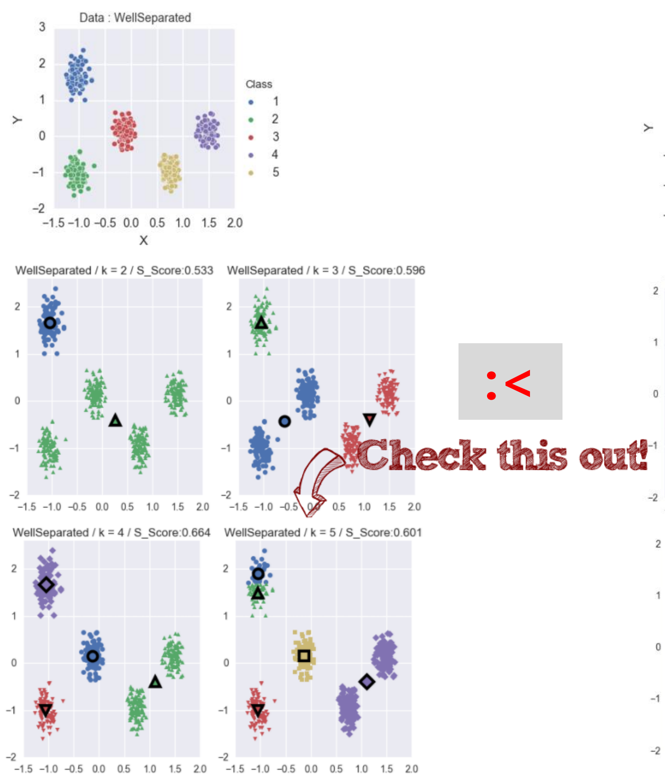


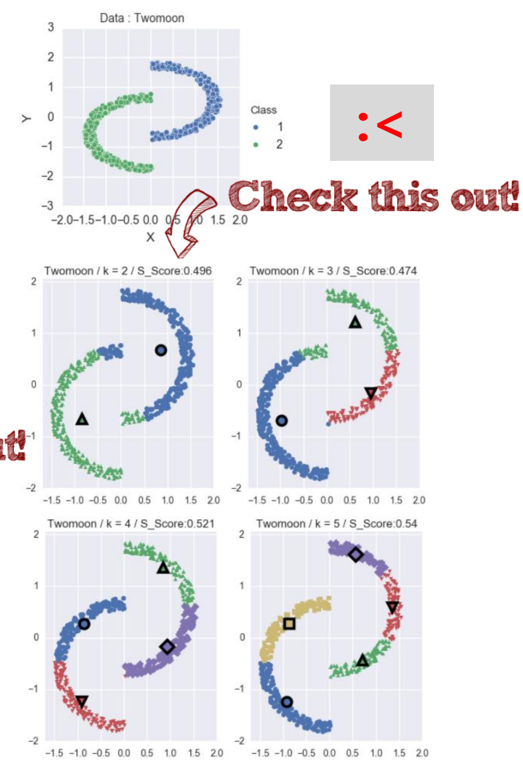
# 클러스터링 실습1 – 2D 인공데이터

- **Initialization method = 'random' , Num_init=5**

```python
for i in range(0,3):
    Simple_Scatter(Artificial_Dataset[list(Artificial_Dataset.keys())[i]],list(Artificial_Dataset.keys())[i])
    k_Means_Plot(Data=Artificial_Dataset[list(Artificial_Dataset.keys())[i]] ,Select_k=range(2, 6),
                 NAME=list(Artificial_Dataset.keys())[i],Init_Method='random',Num_Init=5)
```



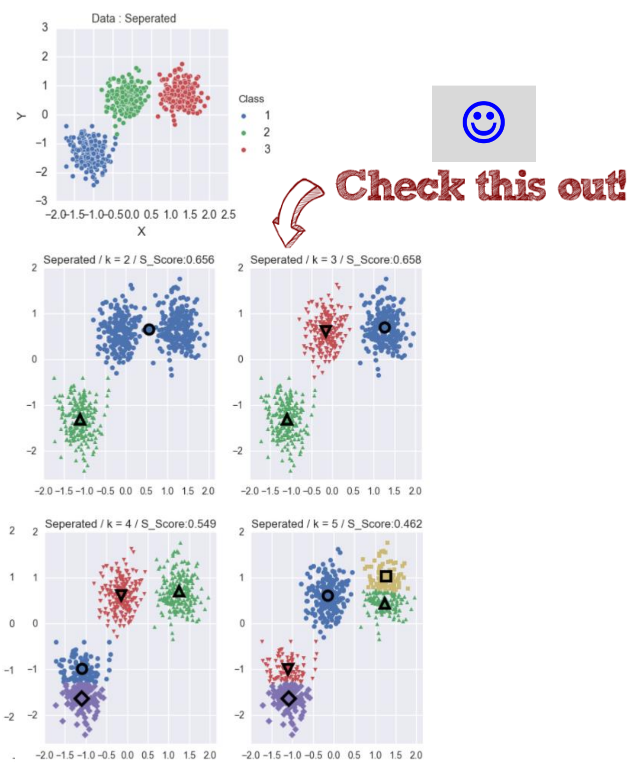



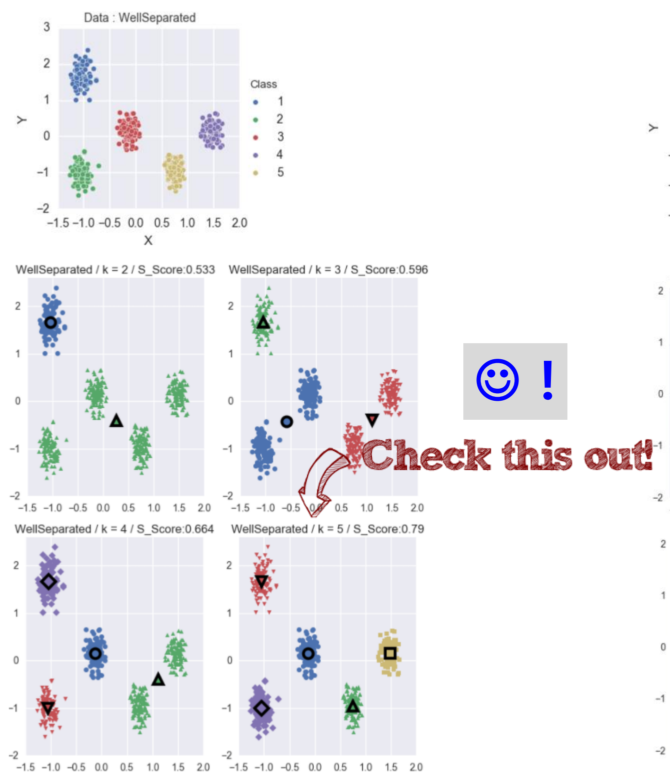



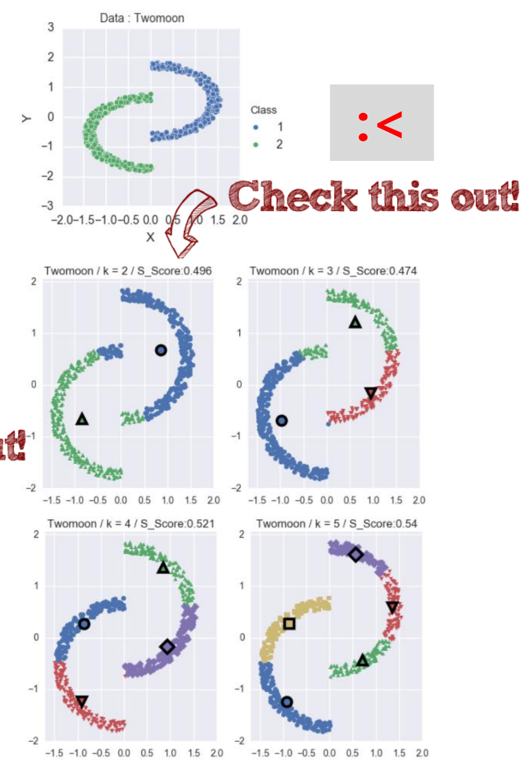
# 클러스터링 실습1 – 2D 인공데이터

- **Initialization method = ‘random’ , Num_init=10**

```
for i in range(0,3):
    Simple_Scatter(Artificial_Dataset[list(Artificial_Dataset.keys())[i]],list(Artificial_Dataset.keys())[i])
    k_Means_Plot(Data=Artificial_Dataset[list(Artificial_Dataset.keys())[i]] ,Select_k=range(2, 6),
                 NAME=list(Artificial_Dataset.keys())[i],Init_Method='random',Num_Init=5)
```



# 클러스터링 실습1 – 2D 인공데이터

- Initialization method = 'kmeans++' , Num_init=5
- 기본적으로 sklearn에서는 initialization method를 k-means++로 사용함

## sklearn.cluster.KMeans

*class* sklearn.cluster.KMeans(*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto'*) [source]

Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.
http://dl.acm.org/citation.cfm?id=128338

### 2.2 The k-means++ algorithm

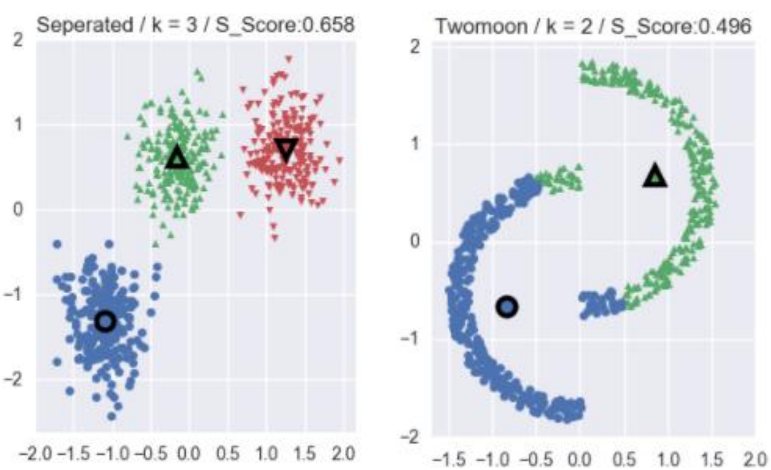
We propose a specific way of choosing centers for the **k-means** algorithm. In particular, let $D(x)$ denote the shortest distance from a data point to the closest center we have already chosen. Then, we define the following algorithm, which we call **k-means++**.

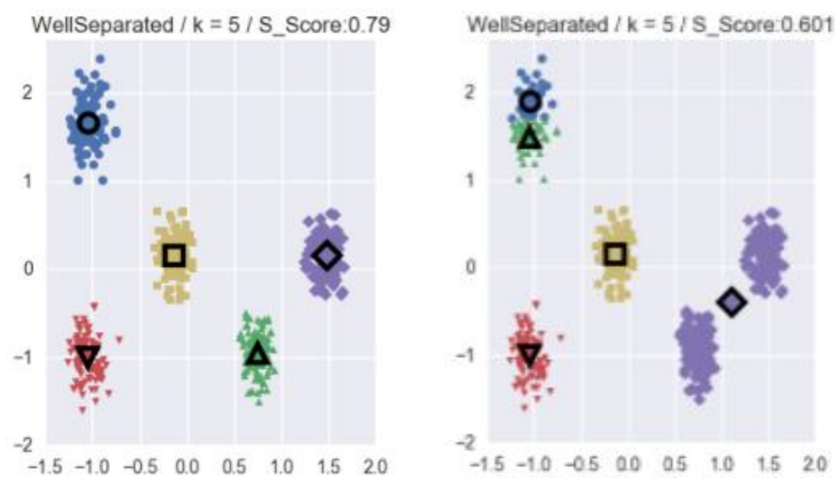
1a. Take one center $c_1$, chosen uniformly at random from $\mathcal{X}$.
1b. Take a new center $c_i$, choosing $x \in \mathcal{X}$ with probability $\frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2}$.
1c. Repeat Step 1b. until we have taken $k$ centers altogether.
2-4. Proceed as with the standard **k-means** algorithm.

Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.
http://dl.acm.org/citation.cfm?id=128338

# 클러스터링 실습1 – 2D 인공데이터

- **Initialization method = 'kmeans++' , Num_init=5**
- **횟수가 5번 이지만 random 대신에 k-Means++을 초기값으로 사용 하는것이 상대적으로 우수함을 알 수 있음**
- **여전히 구형의 모양이 아닌 Twomoon같은 형태에서는 k-Means가 잘 작동하지 않는것을 확인**




[1] Init = 'kmeans++'
[2] Num_init=5

[1] Init = 'random'
[2] Num_init=5

Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.
http://dl.acm.org/citation.cfm?id=128338

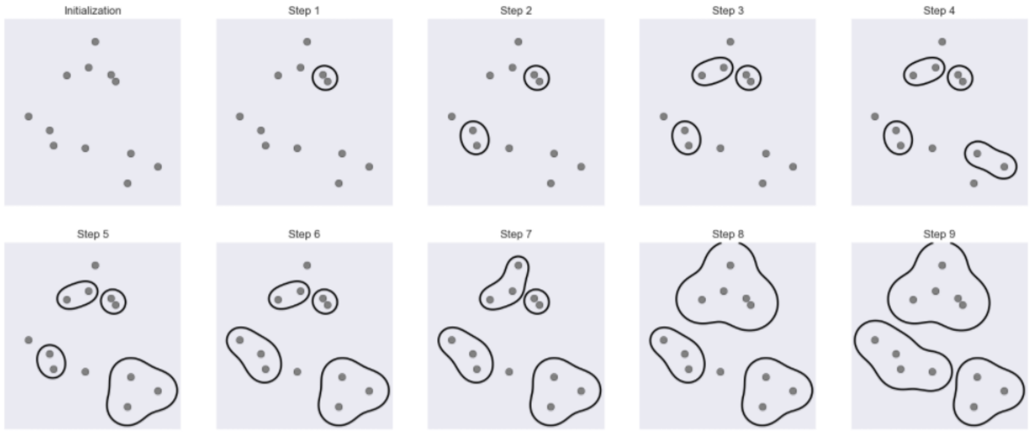
# 클러스터링 실습1 – 2D 인공데이터

- 다시 한번 Hierarchical clustering(agglomerative)의 메커니즘을 위해 간단한 도식화를 확인함

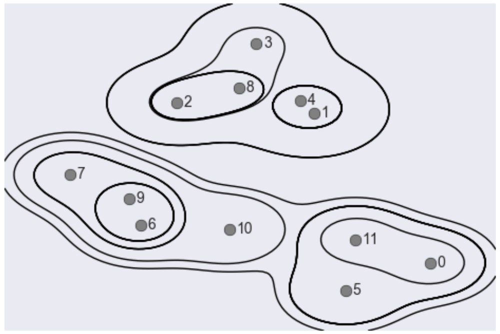
### Algorithm2 : Hierarchical clustering

다시 한번 Hierarchical clustering(agglomerative)의 메커니즘을 위해 간단한 도식화를 확인함

In [166]:
```
mglearn.plots.plot_agglomerative_algorithm()
```




In [167]:
```
mglearn.plots.plot_agglomerative()
```



# 클러스터링 실습1 – 2D 인공데이터

- **Sklearn에서는 dendrogram을 지원하지 않기 때문에 scipy에서 불러옴**

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

**WellSeparated dataset을 기준으로 complete linkage를 이용하여 dendrogram을 시각화**

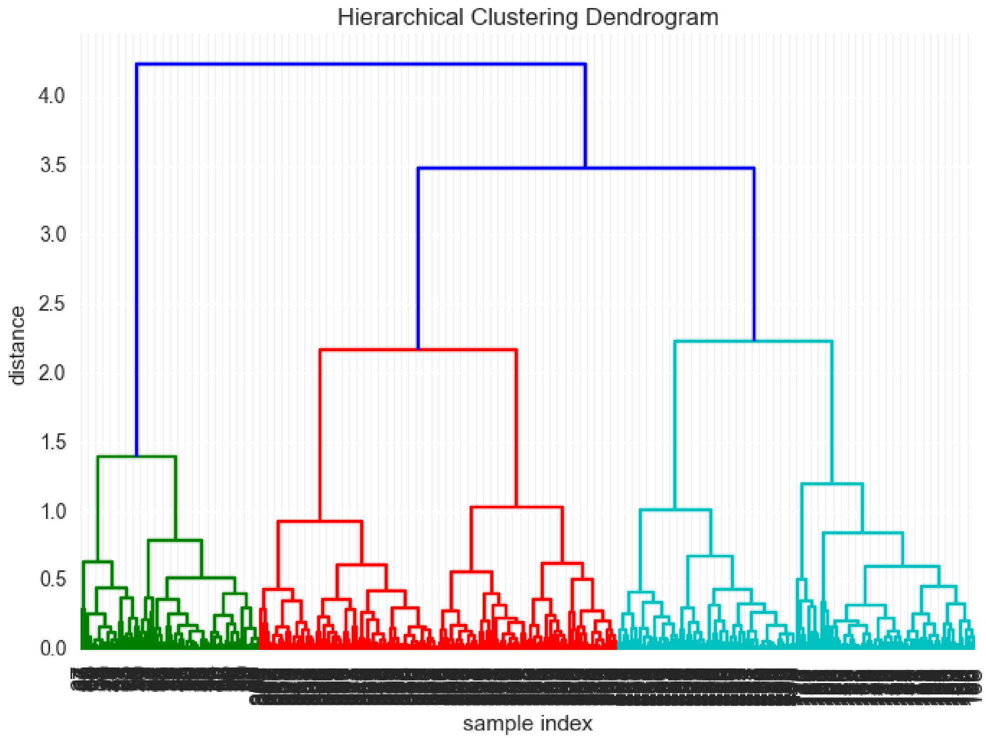
```
Simple_Scatter(Artificial_Dataset[list(Artificial_Dataset.keys())[0]],list(Artificial_Dataset.keys())[0])
def Fixed_Dendrogram(Data,Num_Viz_Leaf_Cluster,Full_Use):
    Linkage_Matrix = linkage(Data,'complete')
    if(Full_Use==True):
        Num_Viz_Leaf_Cluster=np.shape(Data)[0]
        plt.title('Hierarchical Clustering Dendrogram')
        plt.xlabel('sample index')
    else:
        plt.title('Hierarchical Clustering Dendrogram (truncated)')
        plt.xlabel('sample index or (cluster size)')
    plt.ylabel('distance')
    dendrogram(
        Linkage_Matrix,
        truncate_mode='lastp',
        p=Num_Viz_Leaf_Cluster,
        leaf_rotation=90.,
        leaf_font_size=12.,
        color_threshold='default'
    )
    plt.show()
```

# 클러스터링 실습1 – 2D 인공데이터

- **전체 datapoint를 모두 dendrogram에 시각화**

**전체 datapoint를 모두 dendrogram에 시각화**

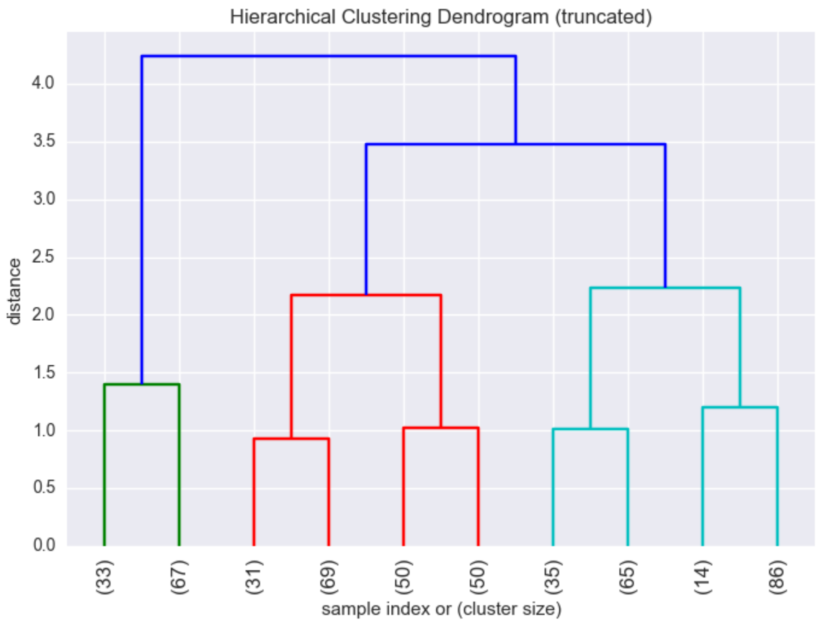
```
Fixed_Dendrogram(WellSeparated[['X','Y']],0,True)
```



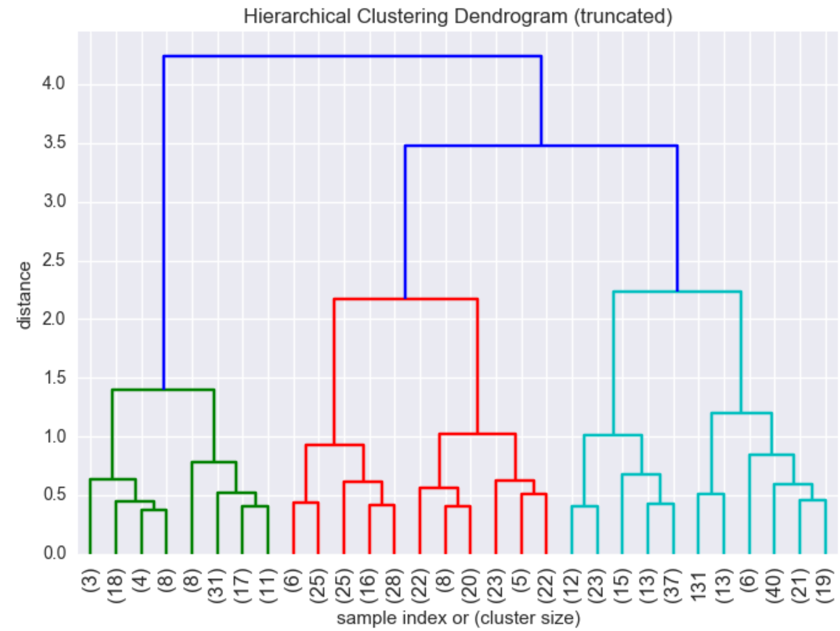
# 클러스터링 실습1 – 2D 인공데이터

▪ **30개와 10개의 cluster를 dendrogram에 시각화**

```
Fixed_Dendrogram(WellSeparated[['X','Y']],10,False)
```




```
Fixed_Dendrogram(WellSeparated[['X','Y']],30,False)
```



# 클러스터링 실습1 – 2D 인공데이터

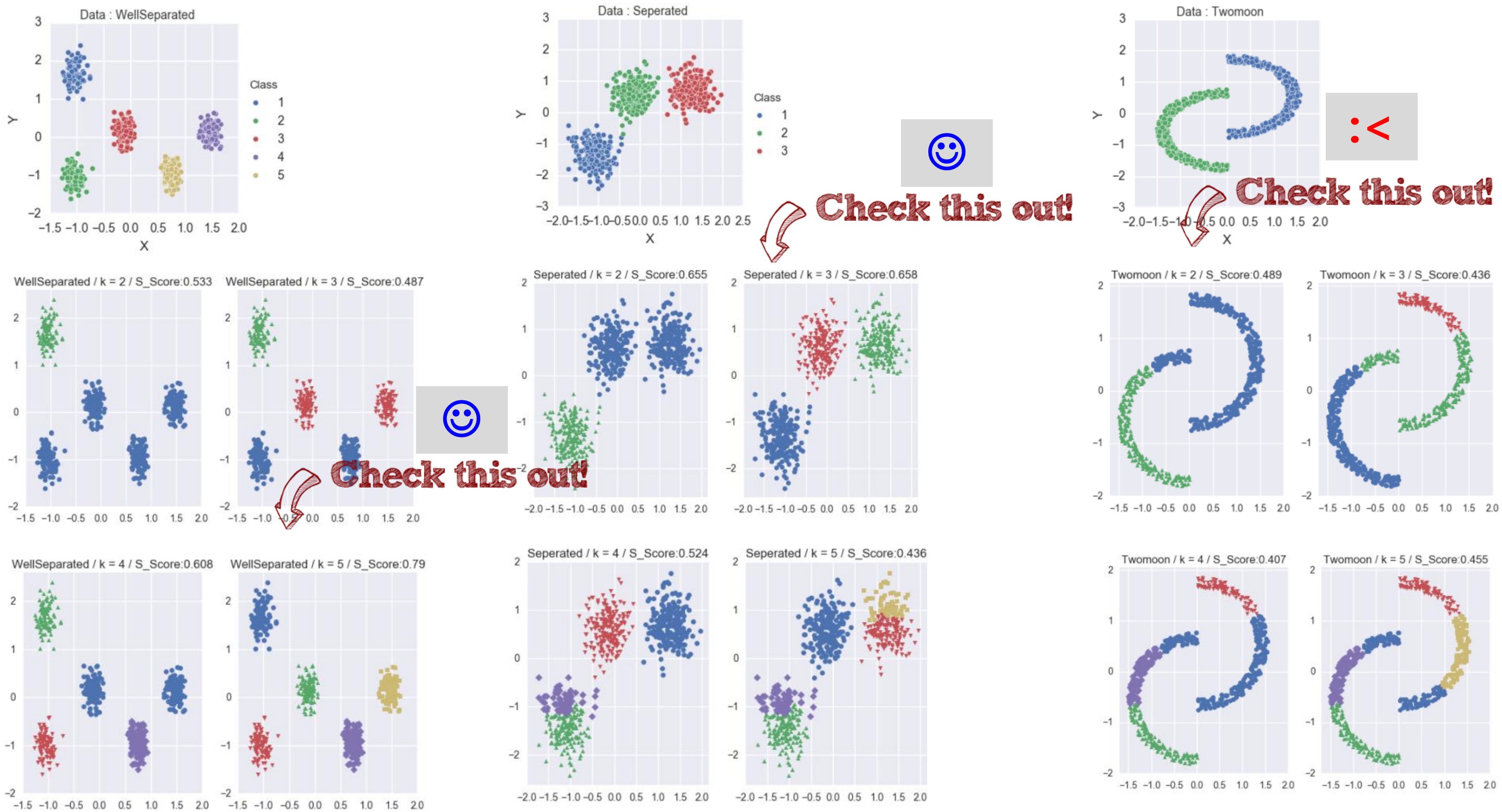
- **Hierarchical clustering 결과를 보도록 하자 complete linkage를 사용**
  - >해당 방법론도 Twomoon dataset에서는 잘 작동하지 않음을 볼 수 있음

```python
def Hclust_Plot(Data,Select_k,NAME):
    Data2 = Data[['X', 'Y']]
    fig, axes = plt.subplots(1, (np.max(list(Select_k))-np.min(list(Select_k)))+1, figsize=(15, 4))
    for i in Select_k:
        H_Clustering = AgglomerativeClustering(n_clusters=i,linkage="complete")
        P_Labels=H_Clustering.fit_predict(Data2)
        mglearn.discrete_scatter(Data2['X'], Data2['Y'], P_Labels, ax=axes[i - 2], s=5)
        axes[i - 2].set_title("Data:" + NAME + ' / k = ' + str(i))
        Score=np.round(silhouette_score(Data2,P_Labels),3)
        axes[i - 2].set_title( NAME + ' / k = ' + str(i)+' / S_Score:'+str(Score))
```

- **K-Means에서 달라진 함수는 2줄임**
- **Sklearn에 있는 AgglomerativeClustring을 사용하였으며**
  **[1] n_cluster :몇 개 기준의 cluster를 만들것인가에 대한 파라미터임**
  **[2] linkage : 어떤 linkage method를 사용 할 것인가에 대한 파랄미터임**

# 클러스터링 실습1 – 2D 인공데이터

- **Hierarchical clustering 결과를 보도록 하자 complete linkage를 사용**
  - \>해당 방법론도 Twomoon dataset에서는 잘 작동하지 않음을 볼 수 있음



# 클러스터링 실습1 – 2D 인공데이터

- 다시 한번 DBSCAN의 메커니즘 확인을 위해 간단한 도식화를 확인함

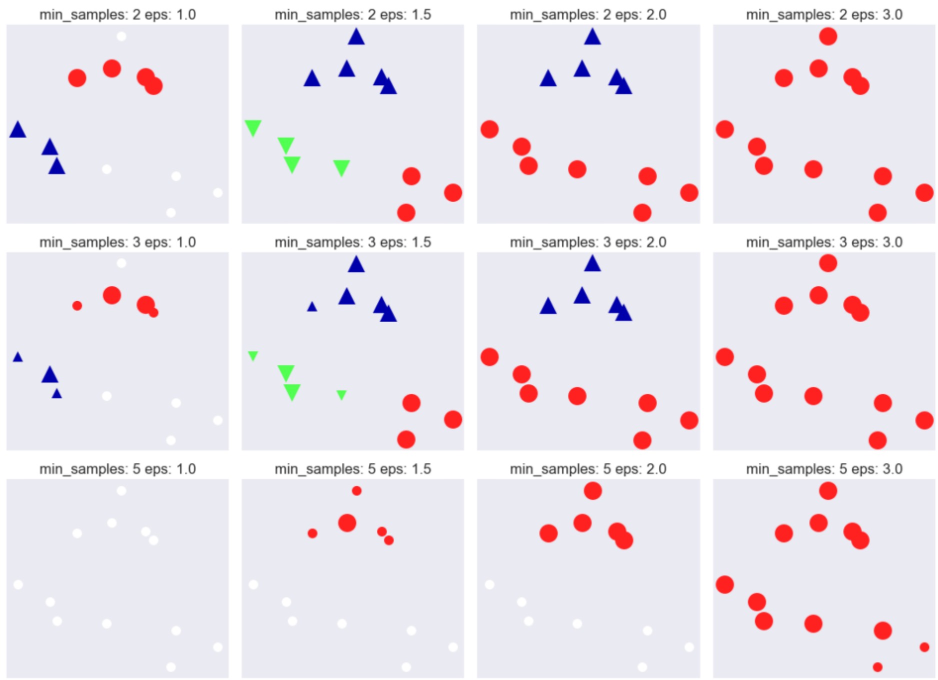
### Algorithm3 : DBSCAN

**다시 한번 DBSCAN의 메커니즘 확인을 위해 간단한 도식화를 확인하자**

75]:

```
mglearn.plots.plot_dbscan()
```

```
min_samples: 2 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 2 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 2 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 2 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 3 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 3 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 3 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 3 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 5 eps: 1.000000  cluster: [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
min_samples: 5 eps: 1.500000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 2.000000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
```



# 클러스터링 실습1 – 2D 인공데이터

- **DBSCAN 결과를 보도록 하자**

```
def DBSCAN_Plot(Data,NAME,min_samples=5,eps=0.4):
    Data2 = Data[['X', 'Y']]
    Append_k_Means_Results = list()
    fig, axes = plt.subplots(1, 2, figsize=(15, 4))
    Set_DBSCAN_Hyperparameter=DBSCAN(min_samples=min_samples,eps=eps)
    Results = Set_DBSCAN_Hyperparameter.fit_predict(Data2)
    Score=np.round(silhouette_score(Data2,Results),3)
    mglearn.discrete_scatter(Data2['X'], Data2['Y'], Data['Class'], ax=axes[0], s=5)
    axes[0].set_title("Data:" + NAME + 'GroundTruth')
    mglearn.discrete_scatter(Data2['X'], Data2['Y'], Results, ax=axes[1], s=5)
    axes[1].set_title("Data:" + NAME + ' DBSCAN/ eps:'+str(eps)+' / min_sample:'+str(min_samples)+'/ S_score:'+str(Score))
```

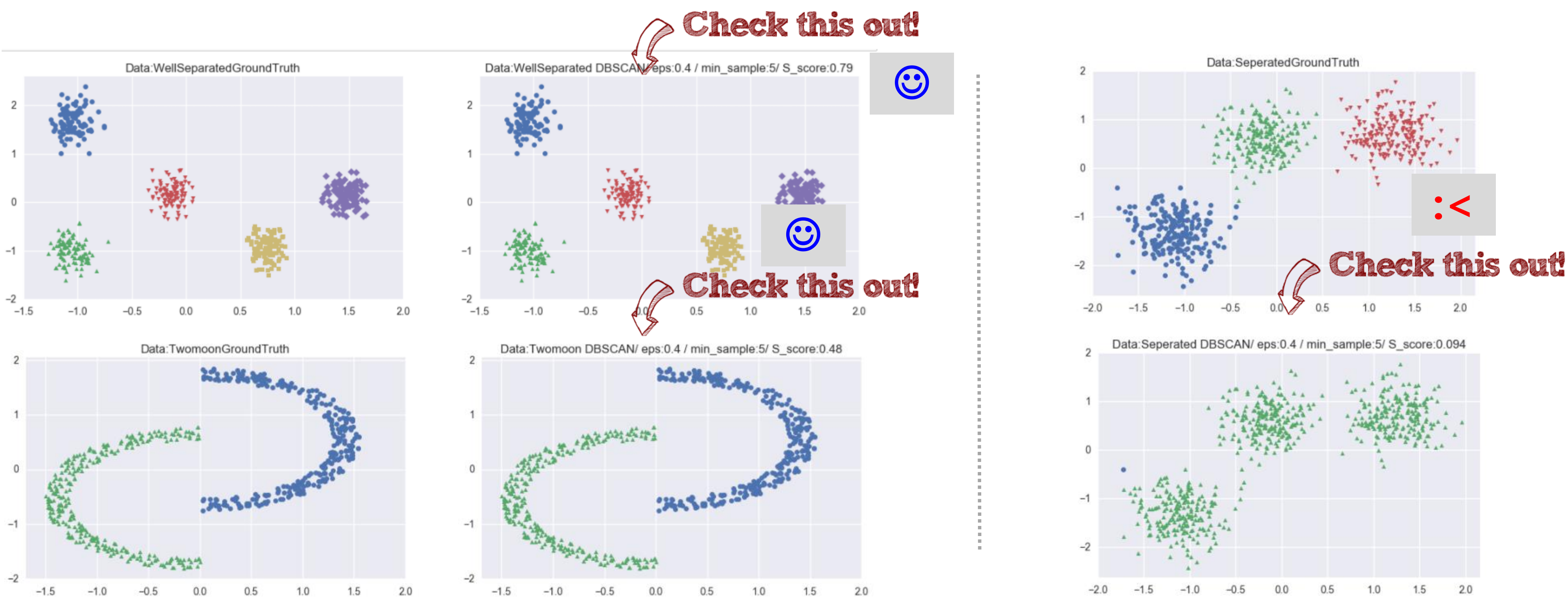
```
class sklearn.cluster. DBSCAN (eps=0.5, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto',
leaf_size=30, p=None, n_jobs=1)
```
[source

- **sklearn에서의 기본값으로는 min_sample : 5, eps : 0.5임 해당 데이터에서 잘 맞지 않아서 eps를 0.4로 변경**
- **특히, DBSCAN의 경우 eps의 distance개념이 중요하기 때문에 standardization 하는것이 중요**

# 클러스터링 실습1 – 2D 인공데이터

## ▪ DBSCAN 결과를 보도록 하자

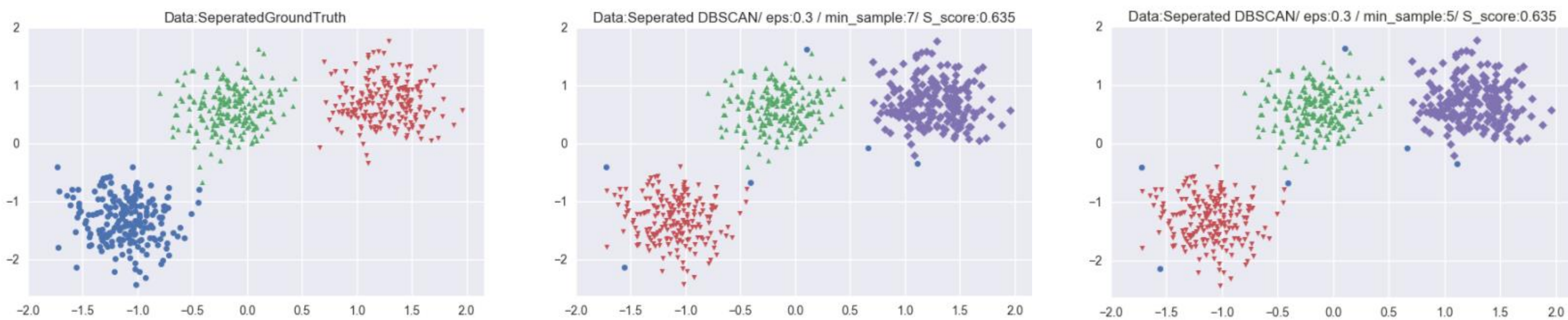
1. WellSeparated 아주 잘 작동함
2. Seperated에서는 eps가 커서 전체적인 하나의 결과로 나온것을 확인
3. Twomoon에서 이전 알고리듬과 달리 아주 잘 작동 하는것을 확인할 수 있음
   + silhouette score가 구형의 데이터가 아니기 때문에 다소 낮게 산출 됨을 확인



# 클러스터링 실습1 – 2D 인공데이터

- **다양한 min_sample과 eps를 통하여 DBSCAN결과 확인**

1. Seperated의 경우 eps가 0.35이상이면 ground truth의 cluster2 개가 뭉쳐짐을 확인 (silhouette score 낮아짐)
2. Best hyper-parameter는 {eps = 0.3, min_smaple = 7}, {eps = 0.3, min_smaple = 5}



# 클러스터링 실습2 – Personal Loan

- **Personal Loan 데이터 불러오며 기본정보확인/정규화를 수행함**

## Clustering 실습2 - Personal Loan

> **실습목표 : Personal Loan 데이터를 기반으로 silhouette score기반 최적의 k-Means clustering 생성 후 해석**

**사용한 PersonalLoan 데이터셋은 다음과 같이 구성되어 있으며 ID와 ZIP code와 Personal Loan은 제외함**

| | |
|---|---|
| ID | Customer ID |
| Age | Customer's Age in completed years |
| Experience | #years of professional experience |
| Income | Annual income of the customer ($000) |
| ZIPCode | Home Address ZIP code. |
| Family | Family size (dependents) of the customer |
| CCAvg | Avg. Spending on Credit Cards per month ($000) |
| Education | Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional |
| Mortgage | Value of house mortgage if any. ($000) |
| Personal Loan | Did this customer accept the personal loan offered in the last campaign? |
| Securities Account | Does the customer have a Securities account with the bank? |
| CD Account | Does the customer have a Certificate of Deposit (CD) account with the bank? |
| Online | Does the customer use internet banking facilities? |
| CreditCard | Does the customer use a credit card issued by UniversalBank? |

# 클러스터링 실습2 – Personal Loan

## ▪ Personal Loan 데이터 불러오며 기본정보확인/정규화를 수행함

**사용할 데이터를 추출 후, standardization함**

```
# 사용할 Personal Loan 데이터셋을 불러옵니다.
Rawdata = pd.read_csv('dataset/Personal Loan.csv')
# Print Column names

print("'Personal Loan' data column name : ", list(Rawdata.columns.values))
print("ID와 ZIP Code는 사용하지 않습니다")
# Allocate column index based on Input and Output varaibles

Input_Column_Index = np.concatenate((range(1,4),range(5,9),range(10,14)))
Target_Column_Index = np.array([9])

# Distance를 이용한 similarity를 구할것이므로 모든 변수를 standardization 한다.
Input_Rawdata = np.array(Rawdata)[:,Input_Column_Index]
Personal_Loan_Data = np.array(Rawdata)[:,Target_Column_Index]

def standardization(Data):
 return ((Data - np.mean(Data, axis=0)) / np.std(Data, axis=0))

Input_Std = standardization(Input_Rawdata)
```

```
'Personal Loan' data column name :  ['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg', 'Education', 'Mortgage', 'Personal
 Loan', 'Securities Account', 'CD Account', 'Online', 'CreditCard']
ID와 ZIP Code는 사용하지 않습니다
```

# 클러스터링 실습2 – Personal Loan

- **k-Means clustering을 시행하며 silhouette score를 사용하여 best k는 5로 확인**

## k-Means clustering을 시행

> 1. Hyper-parameter인 k는 2~10까지 생성
> 2. 평가지표는 silhouette score를 사용

```
def k_Means_Ploan(Data,Select_k,Init_Method='k-means++',Num_Init=100):
    Result_List = list()
    Parameter_List = list()
    Silhouette_List=list()
    for i in Select_k:
        Kmeans_Clustering = KMeans(n_clusters=i,init=Init_Method,n_init=Num_Init,random_state=RANDOM_STATE)
        Kmeans_Clustering.fit(Data)
        Result_List.append(Kmeans_Clustering.labels_)
        Silhouette_List.append(np.round(silhouette_score(Data,Kmeans_Clustering.labels_),3))
        Parameter_List.append(str(i))
    print("Complete!")
    return(Result_List, Parameter_List,Silhouette_List)
```

```
Cluster_Results, Parameter_K, Shilhouette_Score=k_Means_Ploan(Input_Std,range(2,11))
```

```
Complete!
```

# 클러스터링 실습2 – Personal Loan

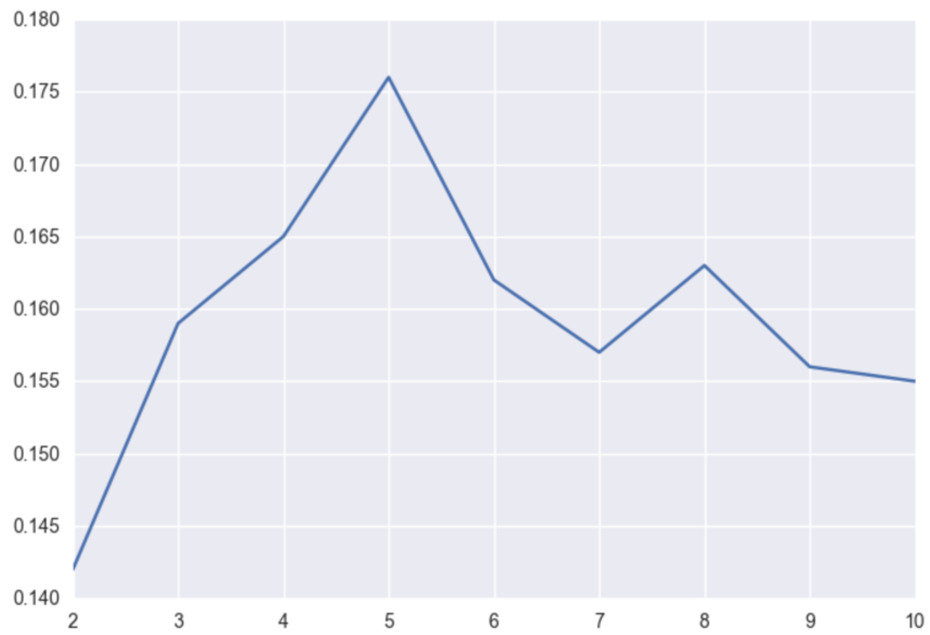
- k-Means clustering을 시행하며 silhouette score를 사용하여 best k는 5로 확인

**k=5일때가 Best인것을 확인**

In [21]:
```
Best_K_based_on_Shilhouette= Parameter_K[np.where(Shilhouette_Score==np.max(Shilhouette_Score))[0][0]]
Best_Cluster_Results = Cluster_Results[np.where(Shilhouette_Score==np.max(Shilhouette_Score))[0][0]]
print("Best k is : " + Best_K_based_on_Shilhouette)
print(collections.Counter(Best_Cluster_Results))
plt.plot(Parameter_K,Shilhouette_Score)
```

```
Best k is : 5
Counter({2: 918, 1: 890, 3: 341, 0: 195, 4: 156})
```

Out[21]: `[<matplotlib.lines.Line2D at 0x1e247f075c0>]`



# 클러스터링 실습2 – Personal Loan

### k=5일때의 각 클러스터 변수의 평균값을 Table로 시각화

```
FULL_Append=list()
for i in range(int(Best_K_based_on_Shilhouette)):

    if (i == 1):
        MinMax_Append = list()

    A = Input_Rawdata[np.where(Best_Cluster_Results == i)]
    B = Personal_Loan_Data[np.where(Best_Cluster_Results == i)]
    P_Loan_Ratio=round(np.sum(B)/np.shape(B)[0],3)

    AVG_List=list()

    for j in range(np.shape(A)[1]):
        if (j == 1):
            AVG_List.append(np.mean(A[:, j]+1e-8))
            if (i == 1):
                MinMax_Append.append((min(Input_Rawdata[:, j]+2+1e-10), max(Input_Rawdata[:, j]+2.1)))
        elif (j==5):
            Undergrad_Ratio =list(collections.Counter(A[:, 5]).values())[0] / np.shape(A)[0]
            Graduate_Ratio = list(collections.Counter(A[:, 5]).values())[1] / np.shape(A)[0]
            Adv_Prof_Ratio = list(collections.Counter(A[:, 5]).values())[2] / np.shape(A)[0]
            AVG_List.append(Undergrad_Ratio)
            AVG_List.append(Graduate_Ratio)
            AVG_List.append(Adv_Prof_Ratio)
            if(i==1):
                for z in range(3):
                    MinMax_Append.append((0+1e-10, 1+0.1))
        else:
            AVG_List.append(np.mean(A[:,j]+11e-8))
            if (i == 1):
                MinMax_Append.append((min(Input_Rawdata[:, j]+1e-10),max(Input_Rawdata[:, j])+0.1))

    AVG_List.append(P_Loan_Ratio)
    FULL_Append.append(AVG_List)


Col_Name=np.concatenate((list(Rawdata.columns.values)[1:4],
list(Rawdata.columns.values)[5:7],
['Undergrad_Ratio','Graduate_Ratio','Adv_Prof_Ratio','Morgage'],
list(Rawdata.columns.values)[10:14],['P_Loan']))

MinMax_Append.append((0+1e-10,1))
FULL_Append = pd.DataFrame(FULL_Append)
FULL_Append.columns=Col_Name
```

# 클러스터링 실습2 – Personal Loan

| | |
|---|---|
| ID | Customer ID |
| Age | Customer's Age in completed years |
| Experience | #years of professional experience |
| Income | Annual income of the customer ($000) |
| ZIPCode | Home Address ZIP code. |
| Family | Family size (dependents) of the customer |
| CCAvg | Avg. Spending on Credit Cards per month ($000) |
| Education | Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional |
| Mortgage | Value of house mortgage if any. ($000) |
| Personal Loan | Did this customer accept the personal loan offered in the last campaign? |
| Securities Account | Does the customer have a Securities account with the bank? |
| CD Account | Does the customer have a Certificate of Deposit (CD) account with the bank? |
| Online | Does the customer use internet banking facilities? |
| CreditCard | Does the customer use a credit card issued by UniversalBank? |

```
FULL_Append.round(2)
```

| | Age | Experience | Income | Family | CCAvg | Undergrad_Ratio | Graduate_Ratio | Adv_Prof_Ratio | Morgage | Securities Account | CD Account | Online | CreditCard | P_Loan |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 46.18 | 20.86 | 62.73 | 2.55 | 1.62 | 0.37 | 0.34 | 0.29 | 52.48 | 1.00 | 0.0 | 0.51 | 0.12 | 0.02 |
| **1** | 35.06 | 9.84 | 60.65 | 2.59 | 1.37 | 0.37 | 0.32 | 0.31 | 48.67 | 0.00 | 0.0 | 0.58 | 0.25 | 0.04 |
| **2** | 55.62 | 30.31 | 57.57 | 2.40 | 1.33 | 0.38 | 0.29 | 0.33 | 43.42 | 0.00 | 0.0 | 0.61 | 0.31 | 0.04 |
| **3** | 43.31 | 18.30 | 147.43 | 1.86 | 4.91 | 0.73 | 0.13 | 0.14 | 104.15 | 0.01 | 0.0 | 0.49 | 0.23 | 0.31 |
| **4** | 46.95 | 21.87 | 107.62 | 2.46 | 2.89 | 0.40 | 0.30 | 0.30 | 93.24 | 0.48 | 1.0 | 0.95 | 0.74 | 0.50 |

# 클러스터링 실습2 – Personal Loan

### Rader Chart를 위한 함수 생성

```
def _invert(x, limits):
    """inverts a value x on a scale from
    limits[0] to limits[1]"""
    return limits[1] - (x - limits[0])

def _scale_data(data, ranges):
    """scales data[1:] to ranges[0],
    inverts if the scale is reversed"""
    for d, (y1, y2) in zip(data[1:], ranges[1:]):
        assert (y1 <= d <= y2) or (y2 <= d <= y1)
    x1, x2 = ranges[0]
    d = data[0]
    if x1 > x2:
        d = _invert(d, (x1, x2))
        x1, x2 = x2, x1
    sdata = [d]
    for d, (y1, y2) in zip(data[1:], ranges[1:]):
        if y1 > y2:
            d = _invert(d, (y1, y2))
            y1, y2 = y2, y1
        sdata.append((d-y1) / (y2-y1)
                     * (x2 - x1) + x1)
    return sdata

class ComplexRadar():
    def __init__(self, fig, variables, ranges,
                 n_ordinate_levels=6):
        angles = np.arange(0, 360, 360./len(variables))

        axes = [fig.add_axes([0.1,0.1,0.9,0.9],polar=True,
                label = "axes{}".format(i))
                for i in range(len(variables))]
        l, text = axes[0].set_thetagrids(angles,
                                         labels=variables)
        [txt.set_rotation(angle-90) for txt, angle
             in zip(text, angles)]
        for ax in axes[1:]:
            ax.patch.set_visible(False)
            ax.grid("off")
            ax.xaxis.set_visible(False)
        for i, ax in enumerate(axes):
            grid = np.linspace(*ranges[i],
                               num=n_ordinate_levels)
            gridlabel = ["{}".format(round(x,2))
                         for x in grid]
            if ranges[i][0] > ranges[i][1]:
                grid = grid[::-1] # hack to invert grid
                          # gridlabels aren't reversed
            gridlabel[0] = "" # clean up origin
            ax.set_rgrids(grid, labels=gridlabel,
                          angle=angles[i])
            #ax.spines["polar"].set_visible(False)
            ax.set_ylim(*ranges[i])
        # variables for plotting
        self.angle = np.deg2rad(np.r_[angles, angles[0]])
        self.ranges = ranges
        self.ax = axes[0]
    def plot(self, data, *args, **kw):
        sdata = _scale_data(data, self.ranges)
        self.ax.plot(self.angle, np.r_[sdata, sdata[0]], *args, **kw)
    def fill(self, data, *args, **kw):
        sdata = _scale_data(data, self.ranges)
        self.ax.fill(self.angle, np.r_[sdata, sdata[0]], *args, **kw)
```

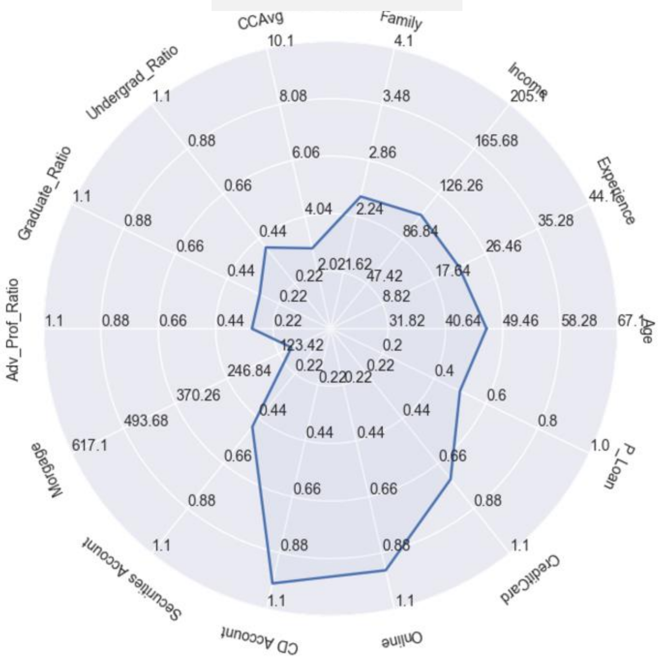
## Rader chart를 통한 Cluster 해석

```
for i in range(5):
    variables = FULL_Append.columns.values
    data = np.array(FULL_Append)[i,:]
    ranges = MinMax_Append
    # plotting
    fig1 = plt.figure(figsize=(6,6))
    radar = ComplexRadar(fig1, variables, ranges)
    radar.plot(data)
    radar.fill(data, alpha=0.05)
    plt.title("Cluster:"+str(i)+" / P_ratio : "+ str(data[13]))
    plt.show()
```

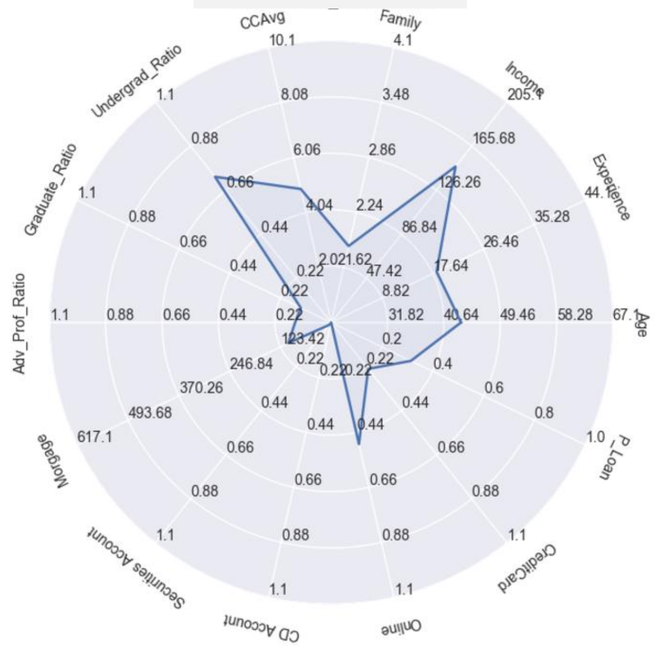
# 클러스터링 실습2 – Rader chart

| Age | Experience | Income | Family | CCAvg | Undergrad_Ratio | Graduate_Ratio | Adv_Prof_Ratio | Morgage | Securities Account | CD Account | Online | CreditCard | P_Loan |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 46.95 | 21.87 | 107.62 | 2.46 | 2.89 | 0.40 | 0.30 | 0.30 | 93.24 | 0.48 | 1.0 | 0.95 | 0.74 | 0.50 |
| 43.31 | 18.30 | 147.43 | 1.86 | 4.91 | 0.73 | 0.13 | 0.14 | 104.15 | 0.01 | 0.0 | 0.49 | 0.23 | 0.31 |
| 35.06 | 9.84 | 60.65 | 2.59 | 1.37 | 0.37 | 0.32 | 0.31 | 48.67 | 0.00 | 0.0 | 0.58 | 0.25 | 0.04 |
| 55.62 | 30.31 | 57.57 | 2.40 | 1.33 | 0.38 | 0.29 | 0.33 | 43.42 | 0.00 | 0.0 | 0.61 | 0.31 | 0.04 |
| 46.18 | 20.86 | 62.73 | 2.55 | 1.62 | 0.37 | 0.34 | 0.29 | 52.48 | 1.00 | 0.0 | 0.51 | 0.12 | 0.02 |

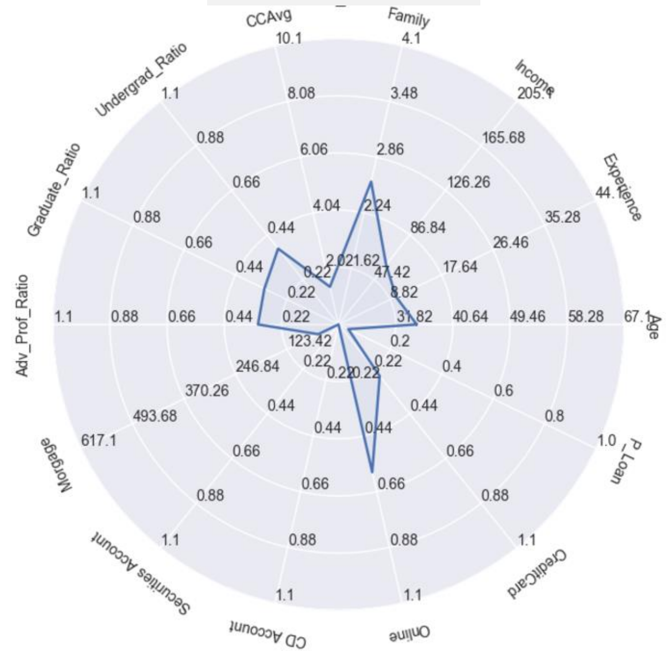
P_ratio : 0.5




P_ratio : 0.31



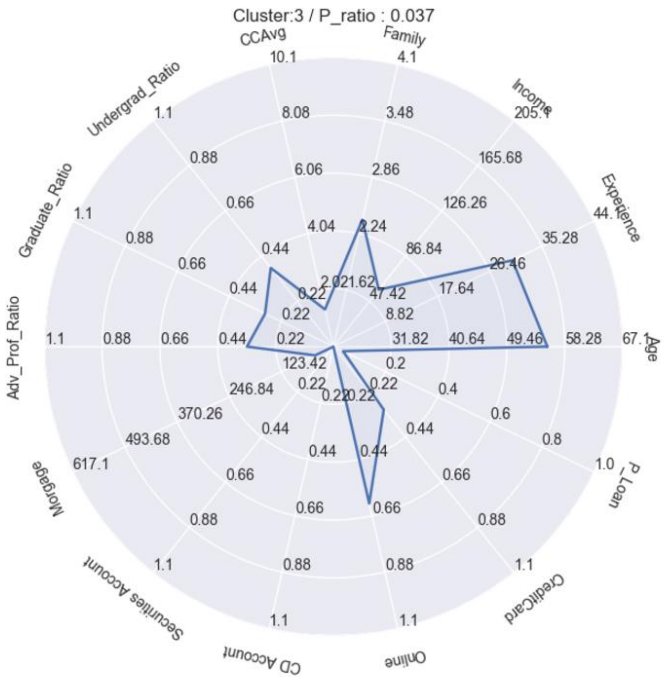

P_ratio : 0.04



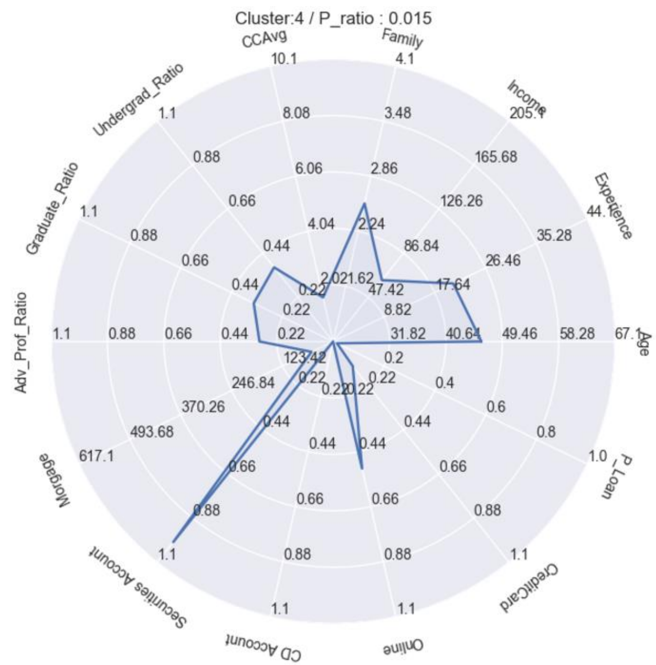
# 클러스터링 실습2 – Rader chart

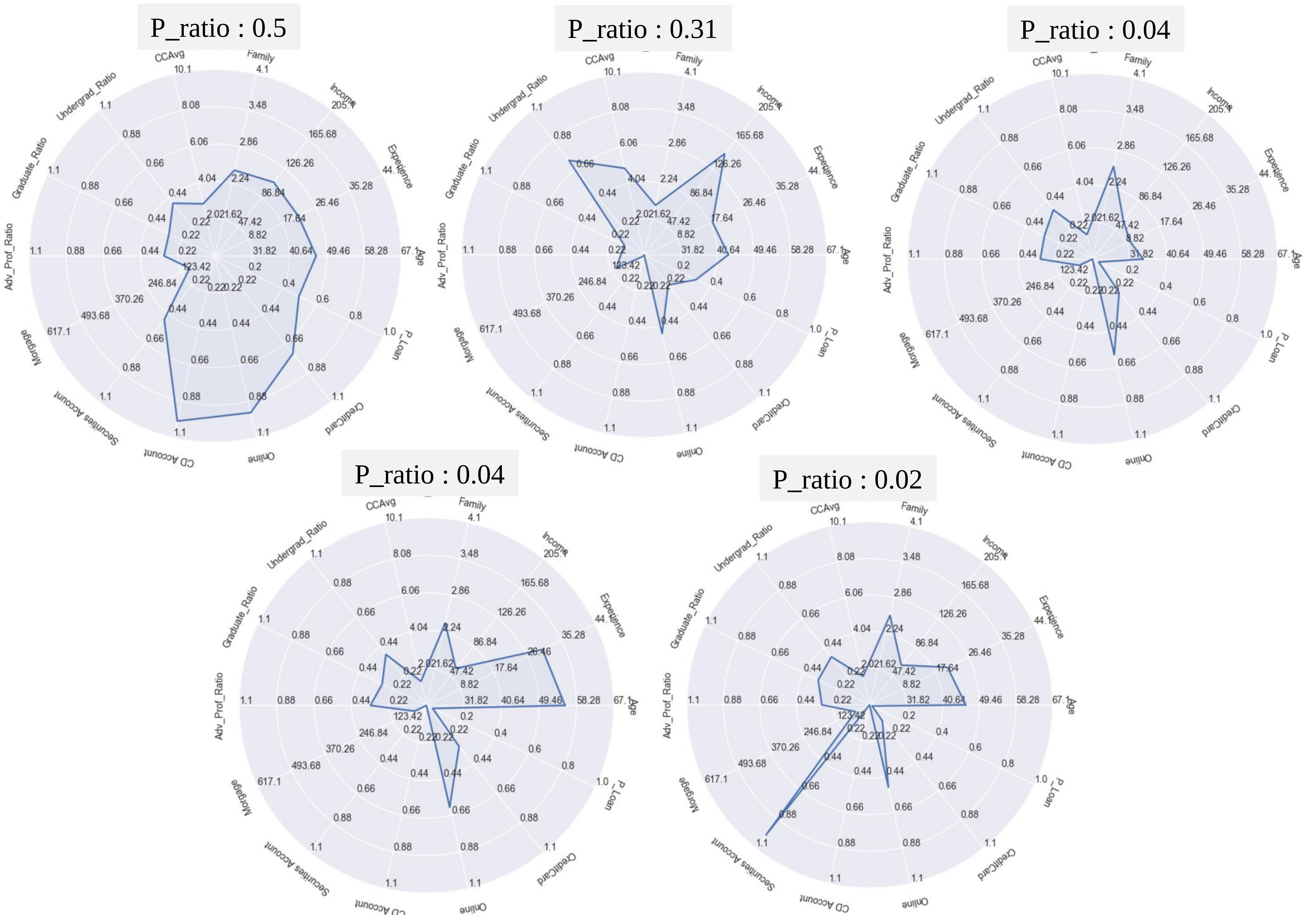
| Age | Experience | Income | Family | CCAvg | Undergrad_Ratio | Graduate_Ratio | Adv_Prof_Ratio | Morgage | Securities Account | CD Account | Online | CreditCard | P_Loan |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 46.95 | 21.87 | 107.62 | 2.46 | 2.89 | 0.40 | 0.30 | 0.30 | 93.24 | 0.48 | 1.0 | 0.95 | 0.74 | 0.50 |
| 43.31 | 18.30 | 147.43 | 1.86 | 4.91 | 0.73 | 0.13 | 0.14 | 104.15 | 0.01 | 0.0 | 0.49 | 0.23 | 0.31 |
| 35.06 | 9.84 | 60.65 | 2.59 | 1.37 | 0.37 | 0.32 | 0.31 | 48.67 | 0.00 | 0.0 | 0.58 | 0.25 | 0.04 |
| 55.62 | 30.31 | 57.57 | 2.40 | 1.33 | 0.38 | 0.29 | 0.33 | 43.42 | 0.00 | 0.0 | 0.61 | 0.31 | 0.04 |
| 46.18 | 20.86 | 62.73 | 2.55 | 1.62 | 0.37 | 0.34 | 0.29 | 52.48 | 1.00 | 0.0 | 0.51 | 0.12 | 0.02 |

P_ratio : 0.04




P_ratio : 0.02




P_ratio : 0.5
P_ratio : 0.31
P_ratio : 0.04
P_ratio : 0.04
P_ratio : 0.02
CCAvg
Family
Income
Experience
Age
P_Loan
CreditCard
Online
CD Account
Securities Account
Mortgage
Adv_Prof_Ratio
Graduate_Ratio
Undergrad_Ratio

# Q & A