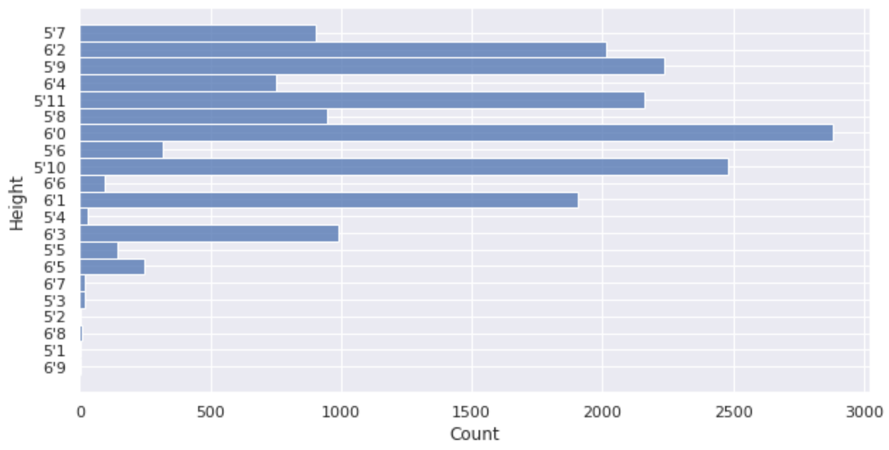
README.md

Clustering

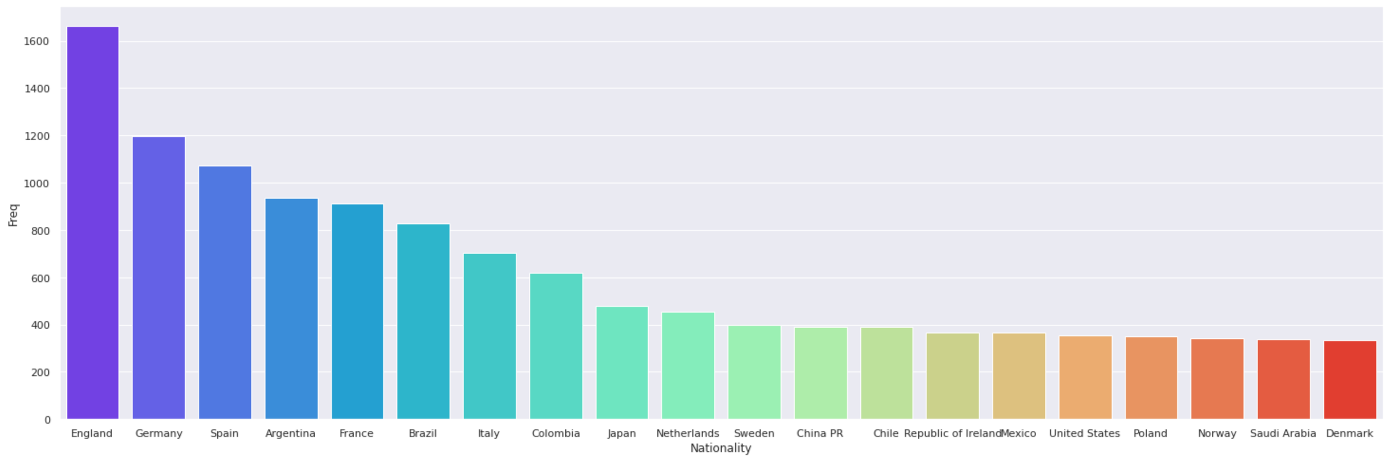
Task1: Data visualisation

`football.csv` has the information about 18K football players and their different features, abilities and skills in the game including other attributes like their club, nationality, height etc. Different features can be selected and the distribution can be visualised. Some such visualisations are as follows.

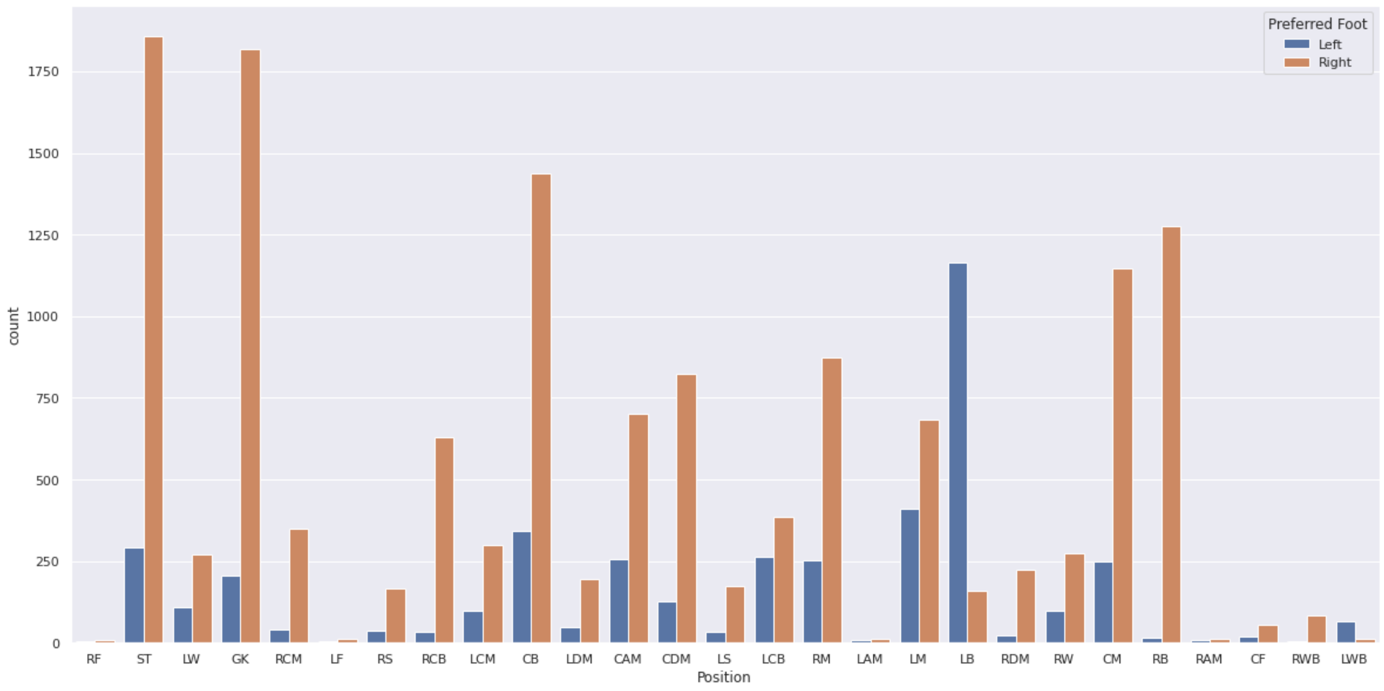
- Count of players wrt height
  - Histplot



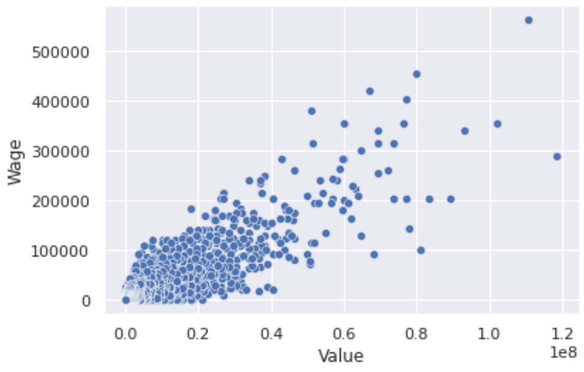
- Count of players according to Nationality
  - Barplot



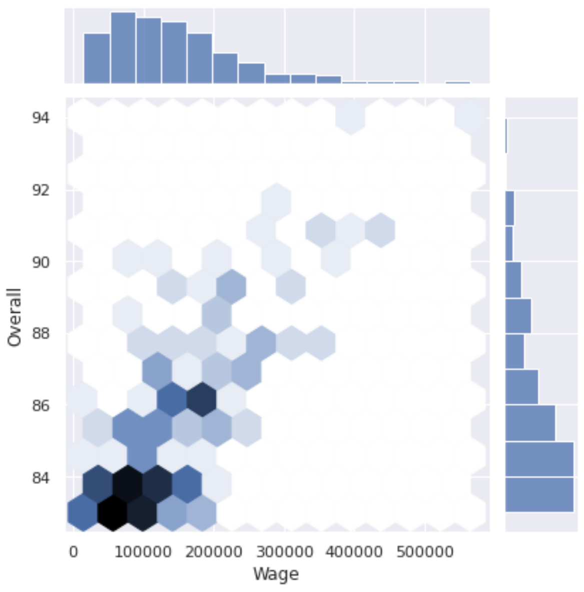
- Preferred foot according to the players position
  - Countplot



- Outlier Detection
  - Scatterplot
  - Visualizing wage vs value of players shows some outliers such as **Lionel Messi, Cristiano Ronaldo**, etc., as their stats are exceptional wrt to others.



- Joint Plot of Wage wrt Overall
  - Jointplot



Many such visualizations are done in the other tasks also.

Task2: K-means Clustering

About: K-means algorithm is an iterative algorithm that tries to partition the dataset into pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

Algorithm:

- Intialize k point from database as intial k cluster center for k-mean.
- Keep iterating until the all k centroid would not be changed.
    i. for every point in dataset compute the minimum distance from all centroid
    ii. select the centroid which give the minimum distance from data-point and assign this point to this cluster.
    iii. compute all new centroid by taking the mean of all data-point from one cluster.

Data Used And Code Explanation-:

The data we used are football player data. we selected all numberical data and some object/string type data likes Height, Value, Weight, Wage and all positions of football player (RW, LB, ST etc/). which are also included in the dataset after converting to numberical attribute.

Attributes like ID, Unamed 0:, Jersey no. is removed (nan value in some places).

After selecting all important attributes we converted it normalized data because some attributes contain big data and some contain small.For feature extraction we used pca dimension reduction method.

For all the task in this like finding silhouette score and elbow score , calculating the inter and intra cluster similarity we used normlized data.

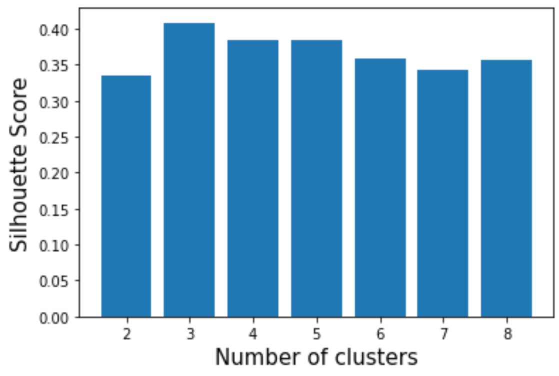
And finally for verifying over answer we compare it with pre-defined python k-means which give the almost same result.

We also did feature extraction for visualizing the data in 2-D. so we can plots all data-point

Cluster Ananlying-:

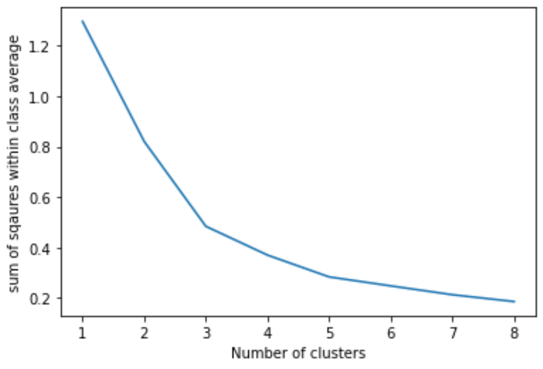
Results-:

1.Silhouette score-: Silhouetter scores for cluster size 2 to 8



```
[0.3349132987027245, 0.40846488347364157, 0.3848483673042597, 0.3838400572892312, 0.3582197184320133, 0.34248379191750
```

2. Elbow score-: Elbow score for cluster size 1 to 8

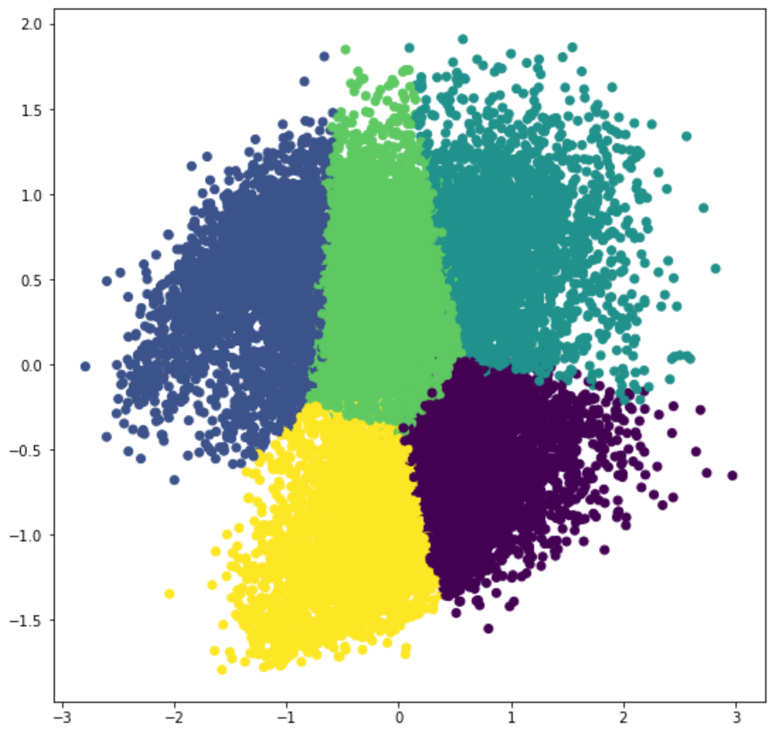


```
[1.296930023806731, 0.8207511340148618, 0.4839925967193346, 0.36981006261203053, 0.28312853622079376, 0.24754304332685
```

from the silhouette score and elbow score it;s clear that the best choice for cluster is 3. The higher silhouette represnt that the points are near in their own cluster than to the neighbouring cluster. The drop in elbow graph in much higher till 3 after that it is not that much big. So by looking at this scores it is clear that k=3 is the best and optimal choice for this dataset to cluster.

Intra and Inter cluster similarity for different k-:

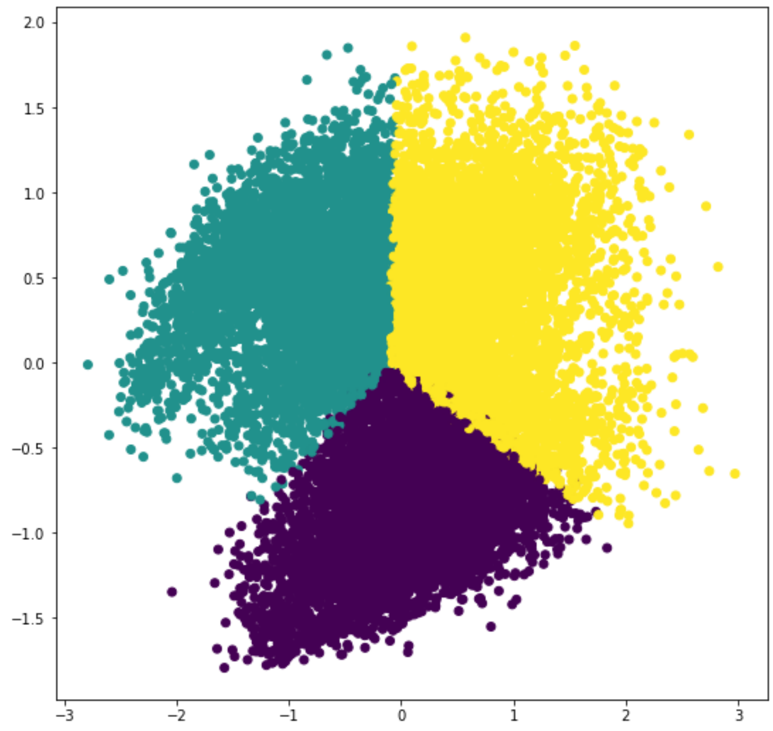
k=5



Similarities-:

```
{
'single link':
       array([[0.        , 1.05700264, 0.4082872 , 0.37385936, 0.41088283],
       [1.05700264, 0.        , 1.01742561, 0.39342583, 0.42610877],
       [0.4082872 , 1.01742561, 0.        , 0.42501287, 0.93891932],
       [0.37385936, 0.39342583, 0.42501287, 0.        , 0.3634143 ],
       [0.41088283, 0.42610877, 0.93891932, 0.3634143 , 0.        ]]),

'complete link':
       array([[3.69766233, 6.12073769, 4.05017083, 4.45351593, 5.36649952],
       [6.12073769, 3.47079308, 5.88369474, 3.89744022, 4.02272064],
       [4.05017083, 5.88369474, 3.55596818, 4.30850724, 5.5227987 ],
       [4.45351593, 3.89744022, 4.30850724, 2.89762204, 4.16606246],
       [5.36649952, 4.02272064, 5.5227987 , 4.16606246, 2.98279904]]),
'average link':
       array([[1.33654122, 2.6109472 , 1.85525791, 1.88463276, 1.77166166],
       [2.6109472 , 1.29826323, 2.58248928, 1.74738097, 2.167251  ],
       [1.85525791, 2.58248928, 1.31482567, 1.68795004, 2.52159333],
       [1.88463276, 1.74738097, 1.68795004, 1.28141754, 1.98059033],
       [1.77166166, 2.167251  , 2.52159333, 1.98059033, 1.35399273]]),

'mean link':
       array([[0.        , 2.27803958, 1.32107471, 1.38424175, 1.20143181],
       [2.27803958, 0.        , 2.2687664 , 1.20770755, 1.72548662],
       [1.32107471, 2.2687664 , 0.        , 1.12077407, 2.17285349],
       [1.38424175, 1.20770755, 1.12077407, 0.        , 1.50732975],
       [1.20143181, 1.72548662, 2.17285349, 1.50732975, 0.        ]])
}
```

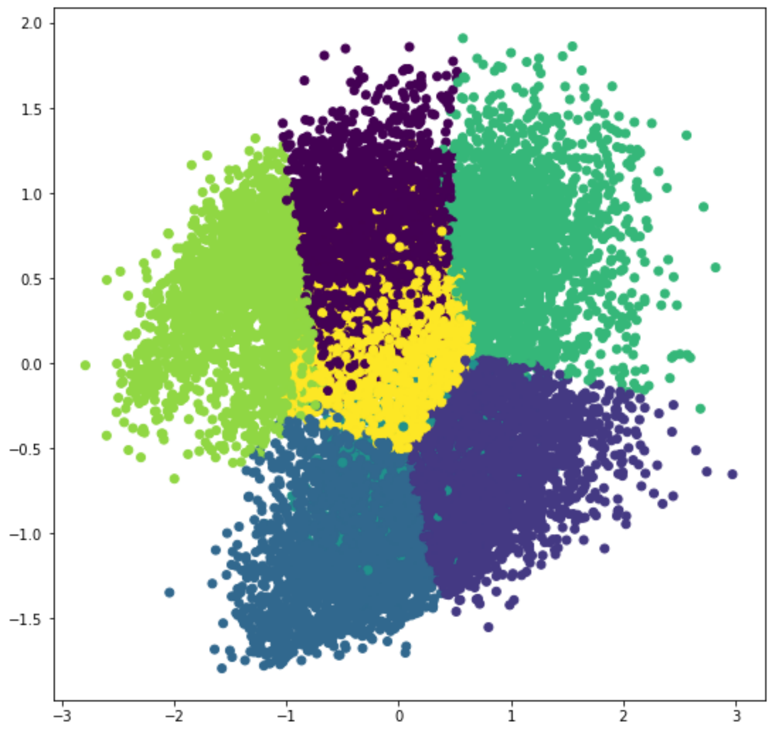
k=3



Similarities-:

```
{
'single link': array([[0.        , 0.32948011, 0.38466927],
       [0.32948011, 0.        , 0.38068313],
       [0.38466927, 0.38068313, 0.        ]]),

'complete link': array([[4.08494037, 5.03163916, 5.5227987 ],
       [5.03163916, 3.61179733, 6.12073769],
       [5.5227987 , 6.12073769, 4.07737143]]),

'average link': array([[1.46495429, 2.16169899, 2.04787512],
       [2.16169899, 1.425201  , 2.09691913],
       [2.04787512, 2.09691913, 1.45848094]]),

'mean link': array([[0.        , 1.62890652, 1.48015943],
       [1.62890652, 0.        , 1.60433893],
       [1.48015943, 1.60433893, 0.        ]])
}
```

k=7

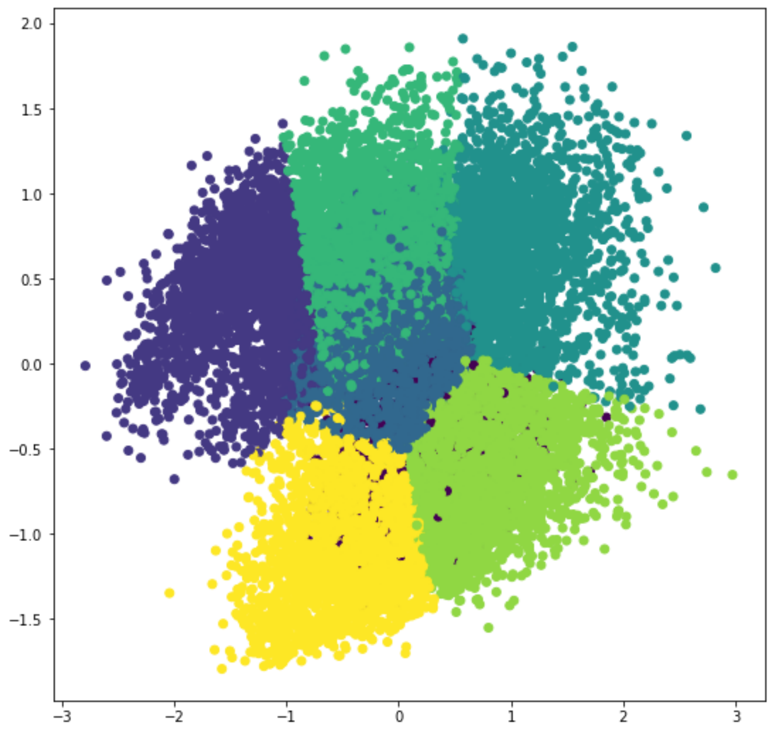


Similarities-:

```
{
'single link':
array([[0.        , 0.64805236, 0.50911048, 0.47702589, 0.4066313 ,
        0.42302821, 0.63813091],
       [0.64805236, 0.        , 0.45414958, 0.48298821, 0.42693697,
        1.27046843, 0.37889121],
       [0.50911048, 0.45414958, 0.        , 0.41879368, 1.10245158,
        0.41672941, 0.32948011],
       [0.47702589, 0.48298821, 0.41879368, 0.        , 0.4940168 ,
        0.66647586, 0.72247498],
       [0.4066313 , 0.42693697, 1.10245158, 0.4940168 , 0.        ,
        1.3859306 , 0.42131284],
       [0.42302821, 1.27046843, 0.41672941, 0.66647586, 1.3859306 ,
        0.        , 0.31097827],
       [0.63813091, 0.37889121, 0.32948011, 0.72247498, 0.42131284,
        0.31097827, 0.        ]]),

'complete link':
array([[2.85663007, 4.79698798, 4.35540982, 3.73043867, 4.47655414,
        4.16360501, 3.21630681],
       [4.79698798, 3.54517238, 5.36649952, 4.63631553, 3.80912516,
        6.12073769, 4.56490804],
       [4.35540982, 5.36649952, 2.84716487, 4.03143861, 5.5227987 ,
        3.80265157, 3.76345933],
       [3.73043867, 4.63631553, 4.03143861, 3.044631  , 4.77215798,
        4.63441057, 3.51524608],
       [4.47655414, 3.80912516, 5.5227987 , 4.77215798, 3.5523069 ,
        5.89348904, 4.52883748],
       [4.16360501, 6.12073769, 3.80265157, 4.63441057, 5.89348904,
        2.81730147, 3.98760732],
       [3.21630681, 4.56490804, 3.76345933, 3.51524608, 4.52883748,
        3.98760732, 2.41106149]]),
'average link':
array([[1.18744264, 2.16118165, 2.25841349, 1.94083148, 1.80140808,
        1.72301463, 1.52923427],
       [2.16118165, 1.23246378, 1.79487141, 1.66276586, 1.83071533,
        2.72960222, 1.74620219],
       [2.25841349, 1.79487141, 1.25231822, 1.70217282, 2.61607192,
        2.1674458 , 1.84451953],
       [1.94083148, 1.66276586, 1.70217282, 1.2381334 , 2.1863502 ,
        2.34480443, 1.96188391],
       [1.80140808, 1.83071533, 2.61607192, 2.1863502 , 1.29920607,
        2.73834424, 1.72064702],
       [1.72301463, 2.72960222, 2.1674458 , 2.34480443, 2.73834424,
        1.25144368, 1.86747078],
       [1.52923427, 1.74620219, 1.84451953, 1.96188391, 1.72064702,
        1.86747078, 1.12925829]]),
'mean link':
array([[0.        , 1.81407758, 1.9236845 , 1.53794128, 1.34008097, 1.26486743, 0.9967815 ],
       [1.81407758, 0.        , 1.36419066, 1.1226935 , 1.34814305, 2.45617021, 1.32356164],
       [1.9236845 , 1.36419066, 0.        , 1.17908157, 2.3203729 , 1.78628592, 1.44456231],
       [1.53794128, 1.1226935 , 1.17908157, 0.        , 1.81149154, 2.00333016, 1.57090687],
       [1.34008097, 1.34814305, 2.3203729 , 1.81149154, 0.        ,2.45726703, 1.27602744],
       [1.26486743, 2.45617021, 1.78628592, 2.00333016, 2.45726703, 0.        , 1.46567615],
       [0.9967815 , 1.32356164, 1.44456231, 1.57090687, 1.27602744,1.46567615, 0.        ]])
}
```

As we can see that graph are more clear for k=3 and the data points are almost equally divided into 3 clusters.

know look at python kmean library clusers-:

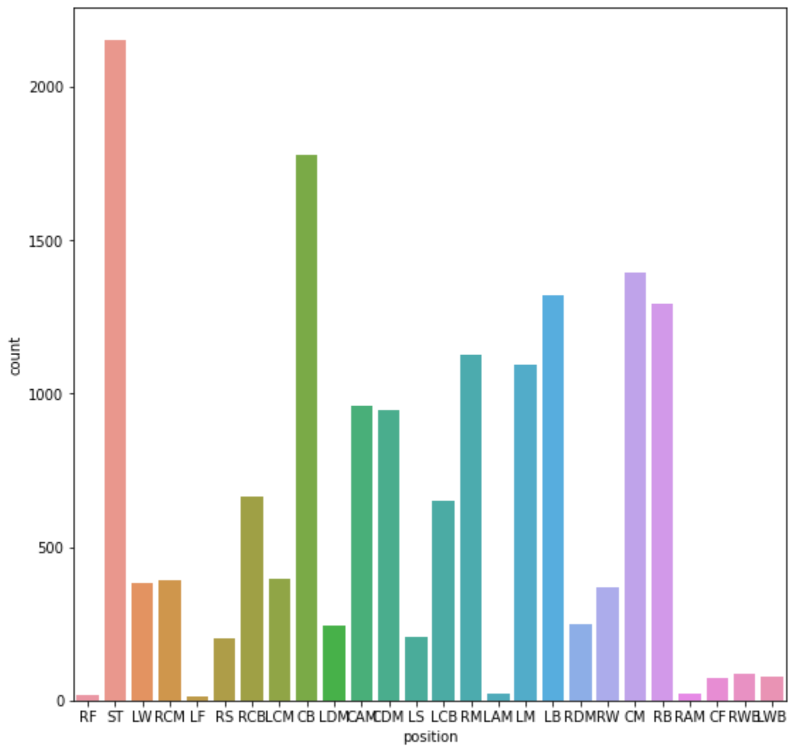


This the what we got from python kmean library which is almost similar to our k-mean output for k=7

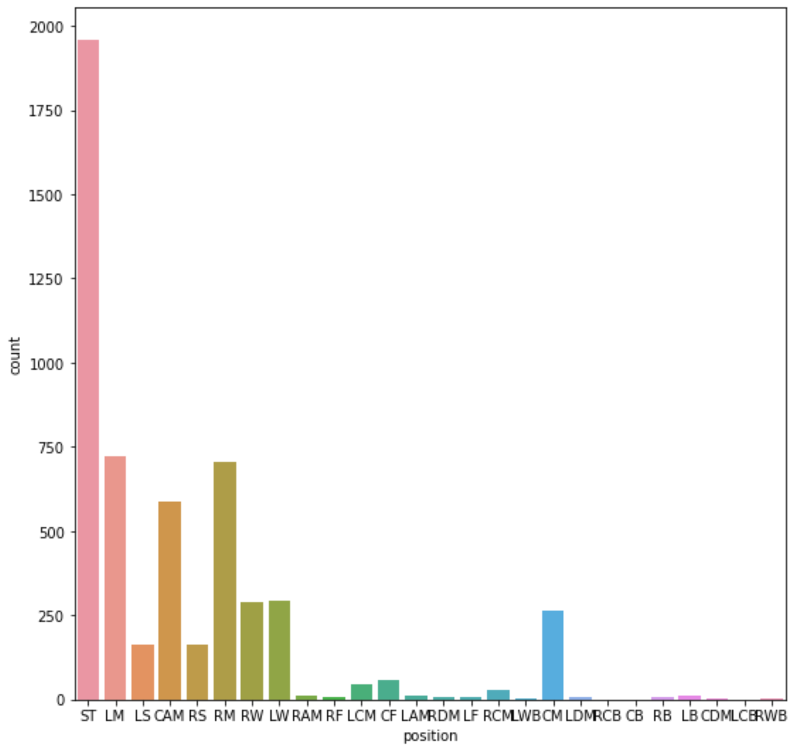
We can analyzing this cluster basis on position of players.

This is plot of all players position over field.

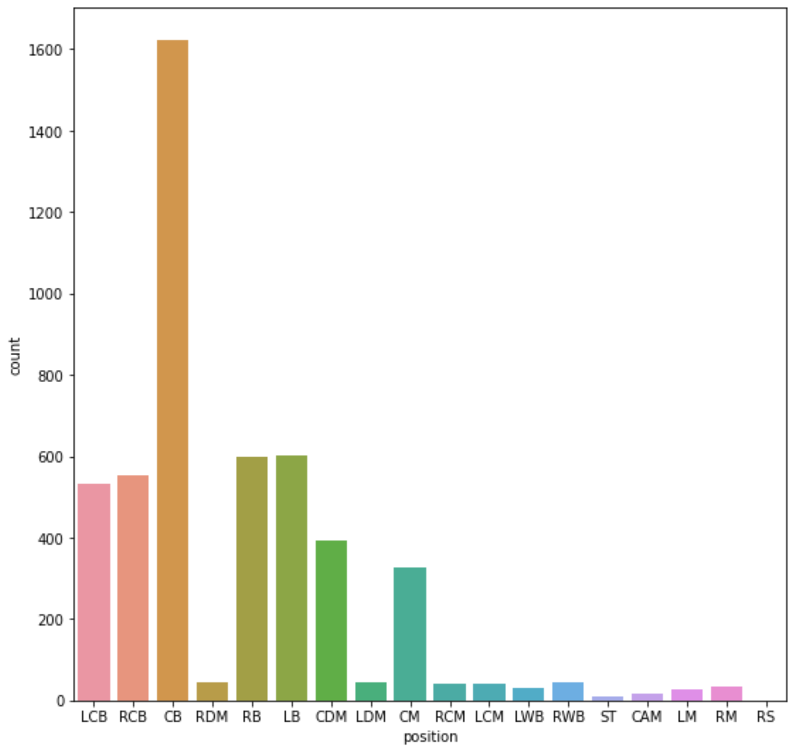
there are positon like ST, SDM, RB, LW etc.



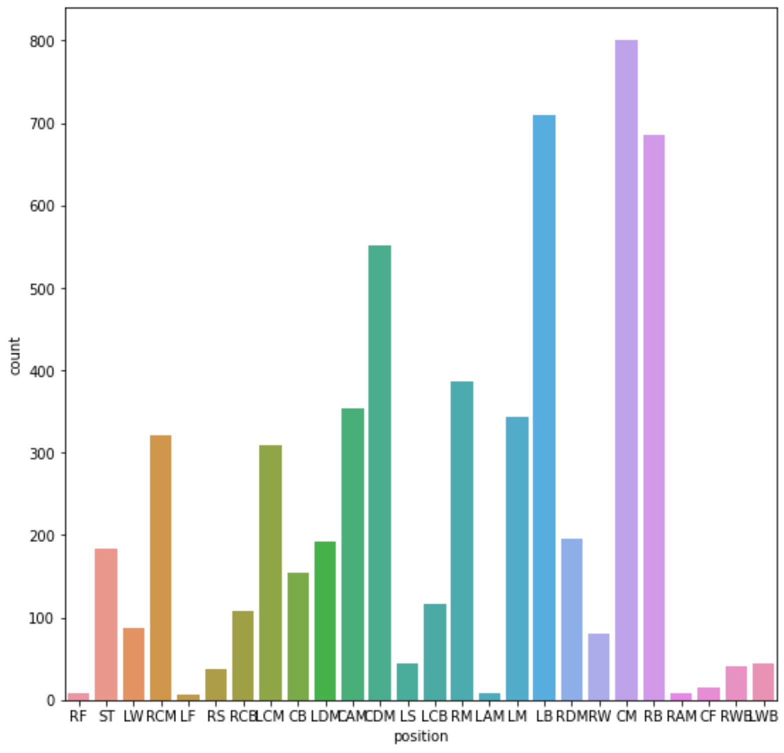
This cluster include almost all the stricker(ST) and it related field positions like (RS, LS, RM,LM etc.) into single cluster.



This cluster include most of Back positions players like CB, LCB, RCB, LB etc.



This cluster include most of the mid-fielder positions like Cm, RDM, LAM etc.



Linkage distance-:

This distances we used for finding similarities b/w clusters.

Single Link-:

minimum distance from all pair distance from one cluster to other cluster. Thus according to the definition, the closest distance between two points in different clusters was maximum for K = 3 This implies thaton increasing the value of K, there were points in different clusterswhich had very less distance between them. Intra cluster similarity is zero b/w same class.

Complete Link-:

maximum distance from all pair distance from one cluster to other cluster. The minimum value of the maximum distance between any two points in different clusters was maximum for K = 3. On increasing the value of K, the maximum distance between any top points belonging to different clusters remained almost the same, however the minimum value of this same metric decreased as K increased which implies thatcertain clusters came closer to each other which is not ideal for clustering. For intra cluster the distance increses b/w the minimum and maximum value of each metric.

Average Link-:

average of sum of all distances of all pairs between 2 clusters.

for inter cluster distance there is significant drop for k=3 to k=5. it means cluster are going near to each other. which is not ideal for clustering. inter cluster distance is same as complete link.

Mean Link-:

distance between 2 cluster means or centroid point.

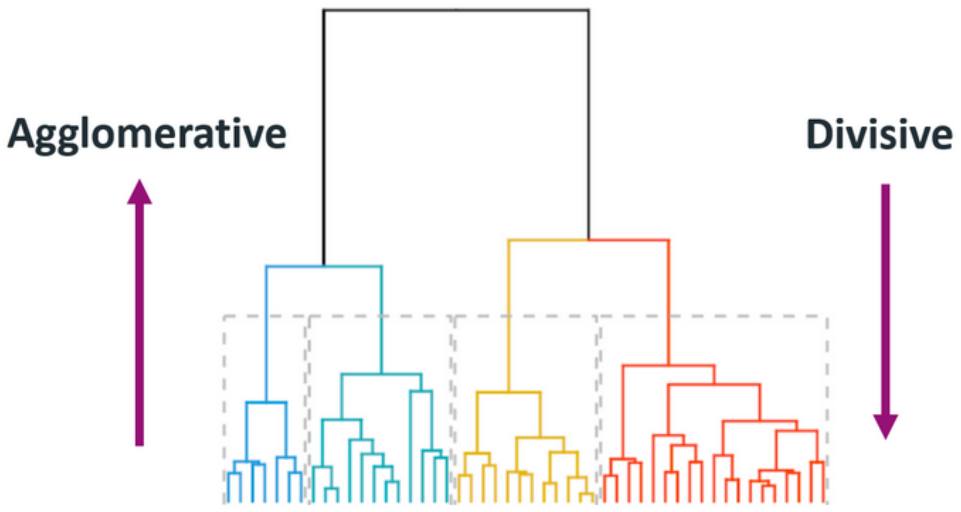
As increasing the value of k, the maximum centroid linkage dis also increses which is ideal for clustering. also increases which is an ideal case for clustering, however this happens at the cost of the minimum value of this same metric, which implies that although there was a pair of cluster which were further away from each other, but at the same time there were clusters whichcame closer to each other which is not an ideal case. for inter class similarity the distances increses b/w the min and maximum value of each metric.

Implementation-:

you can find implementation of our code in **k_mean.ipynb**

Task3: Hierarchical Clustering Algorithm

Hierarchical Clustering creates clusters in a hierarchical tree-like structure (also called a **Dendrogram**). Meaning, a subset of similar data is created in a tree-like structure in which the root node corresponds to entire data, and branches are created from the root node to form several clusters.



Agglomerative Clustering (Bottom-Up)

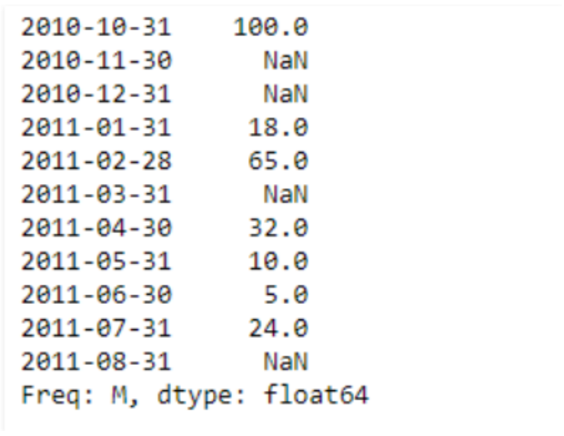
How Agglomerative Hierarchical clustering Algorithm Works:

1. Start assigning each observation as a single point cluster, so that if we have N observations, we have N clusters, each containing just one observation.
2. Find the closest pair of clusters and make them into one cluster, we now have N-1 clusters. This can be done in various ways to identify similar and dissimilar measures.
3. Find the two closest clusters and make them to one cluster. We now have N-2 clusters. This can be done using agglomerative clustering linkage techniques.
4. Repeat steps 2 and 3 until all observations are clustered into one single cluster of size N.

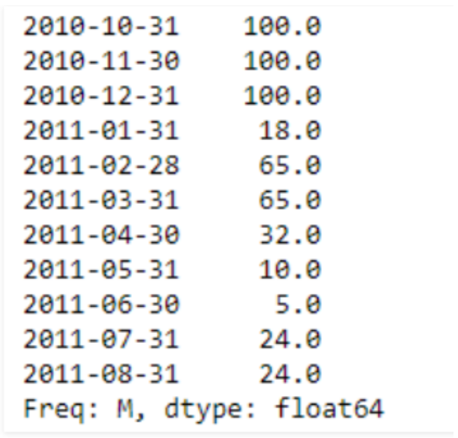
Implementation

Loading and Data Cleaning

- Only the numeric data from the csv file is loaded and also some attributes such as height, weight, value and wage have been converted from string to numeric data type.
- There are some cells with *NaN* values. To counter that we tested 2 methods:
    - droping rows and columns which contain NaN values.
    - Forward fill used to fill the NA/empty values. How it works (reference: `GeeksForGeeks`):



```
2010-10-31    100.0
2010-11-30      NaN
2010-12-31      NaN
2011-01-31     18.0
2011-02-28     65.0
2011-03-31      NaN
2011-04-30     32.0
2011-05-31     10.0
2011-06-30      5.0
2011-07-31     24.0
2011-08-31      NaN
Freq: M, dtype: float64
```

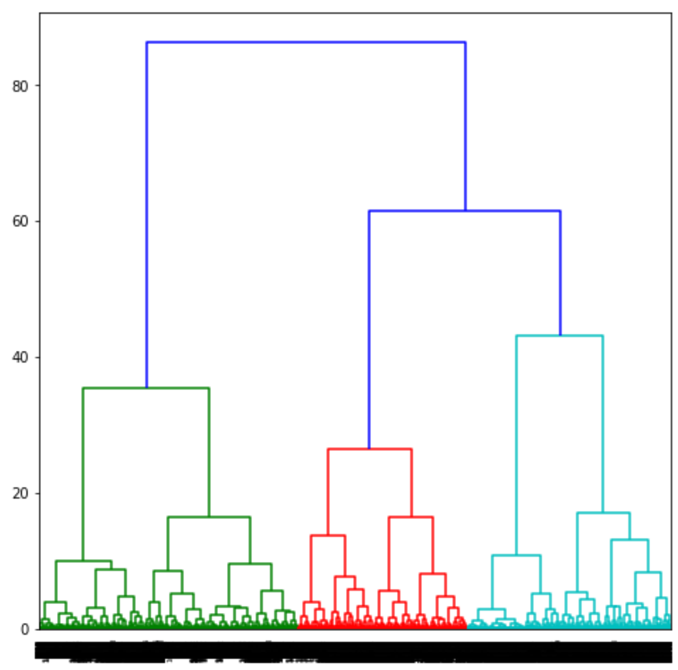


```
2010-10-31    100.0
2010-11-30    100.0
2010-12-31    100.0
2011-01-31     18.0
2011-02-28     65.0
2011-03-31     65.0
2011-04-30     32.0
2011-05-31     10.0
2011-06-30      5.0
2011-07-31     24.0
2011-08-31     24.0
Freq: M, dtype: float64
```

Pre-processing data Data is scaled and then normalized and used as input for clustering.

Dimensionality Reduction To better visualize clustering in 2D, the dimension is reduced using PCA.

Dendogram Visualization

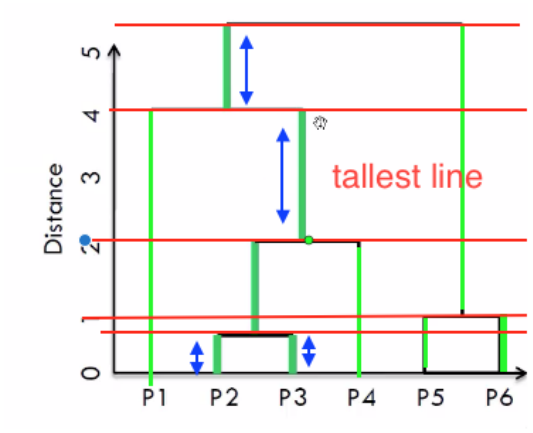


How to choose optimal number of clusters based on dendogram?

Look for the clusters with the longest ‘branches’, the shorter they are, the more similar they are to following ‘twigs’ and ‘leaves’.

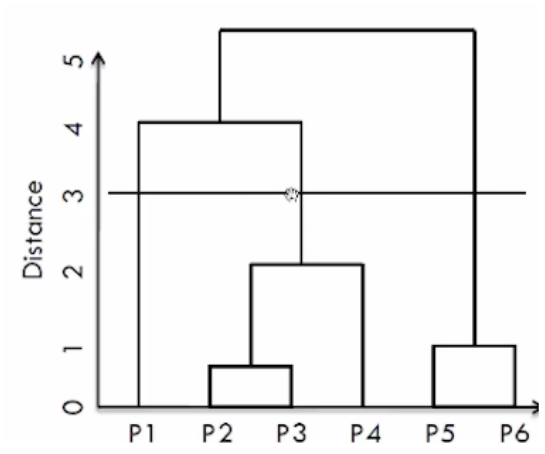
Although finally we need to check the best no of clusters practically, this is just to visualize how data is connected and the hierarchy in it.

Example of working of dendogram:



– find the tallest line that remains uncut

– Number of clusters here is then 3 as it cuts 3 VERTICAL LINES

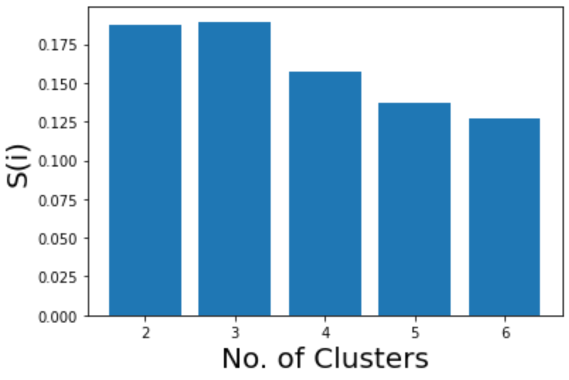


Visualizing Clusters We did agglomerative clustering with a number of clusters = {2,3,4,5,6}.

- Method 1: dropping rows and columns with NaN values

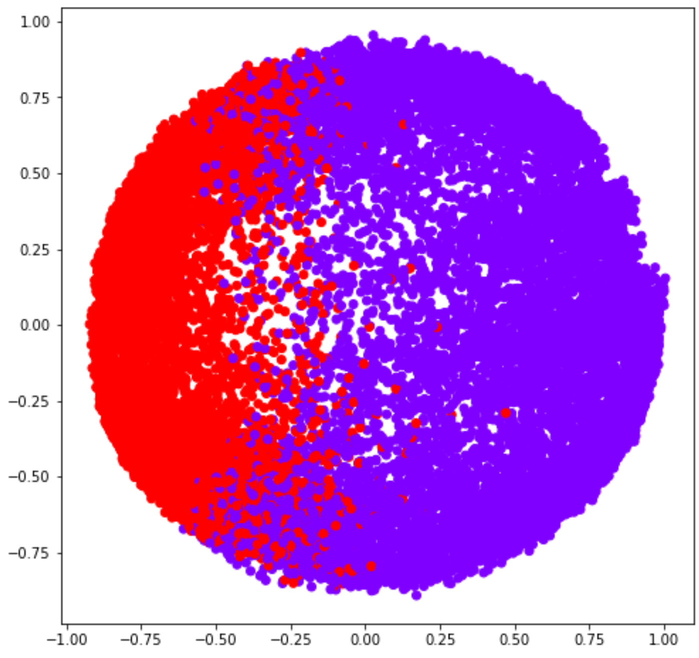
  When they are dropped, the best no of clusters with maximum silhouette score was `k=3` .

  - Silhouette Scores
  -



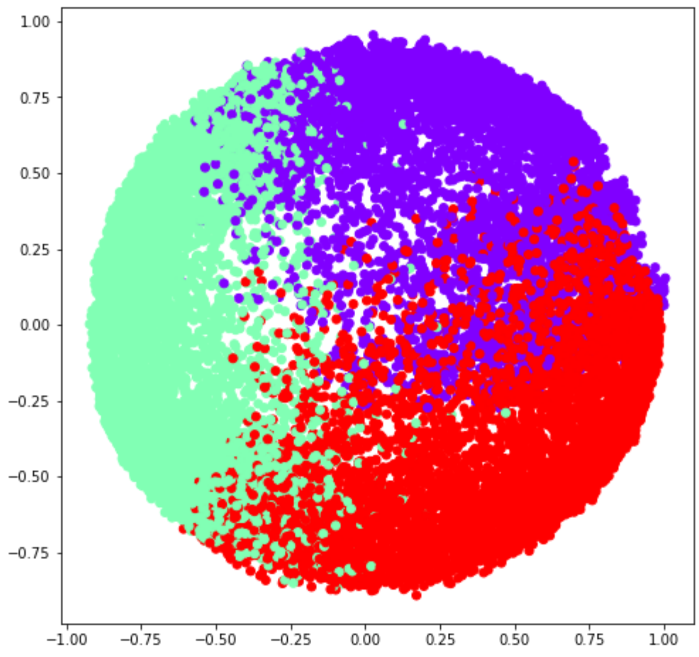
  Scores: `[0.22111164675866438, 0.22096734698286216, 0.24945516810697613, 0.21685147685568962, 0.1980648052767617]` These scores shows that 3 is the best choice for no of clusters.

  - No of clusters = 2



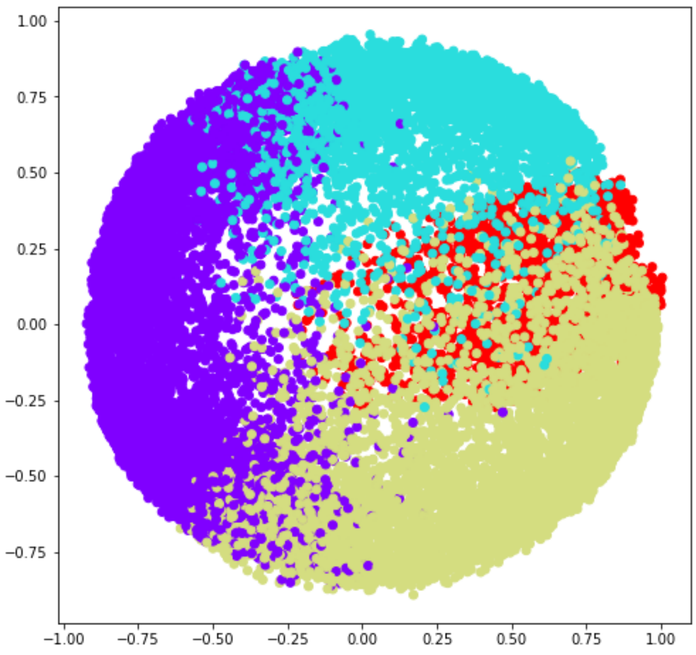
```
{
'single link': array([[0.        , 0.28883367],
       [0.28883367, 0.        ]]),

'complete link': array([[1.95835921, 1.97440153],
       [1.97440153, 1.90559826]]),

'average link': array([[1.27853684, 1.53718776],
       [1.53718776, 1.02875783]]),

'mean link': array([[0.        , 0.99482601],
       [0.99482601, 0.        ]])}
```

- No of clusters = 3



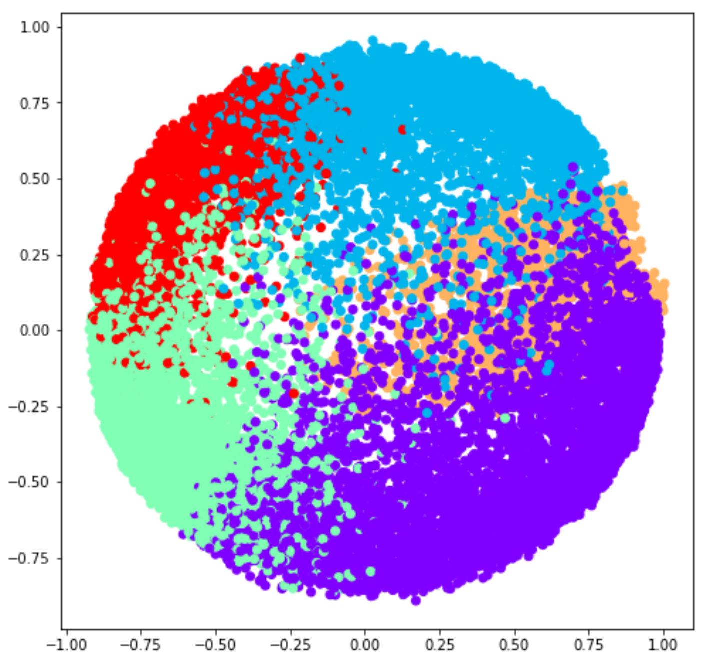
```
{
'single link': array([[0.        , 0.29372368, 0.25967829],
       [0.29372368, 0.        , 0.28883367],
       [0.25967829, 0.28883367, 0.        ]]),

'complete link': array([[1.86220883, 1.97440153, 1.95835921],
       [1.97440153, 1.90559826, 1.97013664],
       [1.95835921, 1.97013664, 1.93070074]]),

'average link': array([[1.13534235, 1.53555182, 1.4610596 ],
       [1.53555182, 1.02875783, 1.53870254],
       [1.4610596 , 1.53870254, 1.063296  ]]),

'mean link': array([[0.        , 1.07140143, 0.93343019],
       [1.07140143, 0.        , 1.12341824],
       [0.93343019, 1.12341824, 0.        ]])}
```

- No of clusters = 4



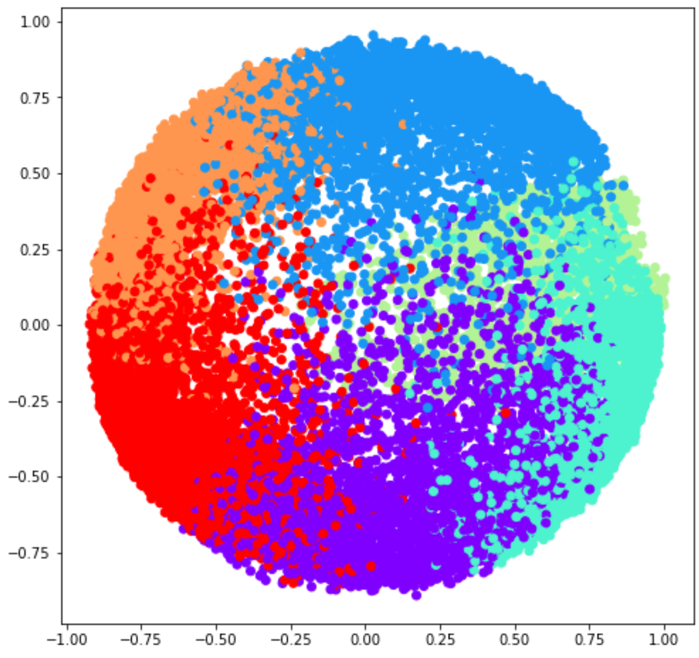
```
{'single link': array([[0.        , 0.29372368, 0.28883367, 0.6631247 ],
       [0.29372368, 0.        , 0.25967829, 0.56730896],
       [0.28883367, 0.25967829, 0.        , 0.4652471 ],
       [0.6631247 , 0.56730896, 0.4652471 , 0.        ]]), 'complete link': array([[1.90559826, 1.96878413, 1.97013664
       [1.96878413, 1.83702006, 1.95835921, 1.86220883],
       [1.97013664, 1.95835921, 1.93070074, 1.92819066],
       [1.97440153, 1.86220883, 1.92819066, 1.54237967]]), 'average link': array([[1.02875783, 1.45459397, 1.53870254,
       [1.45459397, 0.95803807, 1.49444627, 1.42552534],
       [1.53870254, 1.49444627, 1.063296  , 1.40646432],
       [1.66793749, 1.42552534, 1.40646432, 0.66041807]]), 'mean link': array([[0.        , 1.06699645, 1.12341824, 1.
       [1.06699645, 0.        , 1.09195452, 1.15604061],
       [1.12341824, 1.09195452, 0.        , 1.08422908],
       [1.4115937 , 1.15604061, 1.08422908, 0.        ]])}
```

- No of clusters = 5



```
{
'single link': array([[0.        , 0.25967829, 0.28883367, 0.4652471 , 0.59157585],
      [0.25967829, 0.        , 0.49026794, 0.56730896, 0.29372368],
      [0.28883367, 0.49026794, 0.        , 0.78323965, 0.22020801],
      [0.4652471 , 0.56730896, 0.78323965, 0.        , 0.6631247 ],
      [0.59157585, 0.29372368, 0.22020801, 0.6631247 , 0.        ]]),

'complete link': array([[1.93070074, 1.95835921, 1.97013664, 1.92819066, 1.96589465],
      [1.95835921, 1.83702006, 1.96878413, 1.86220883, 1.9321    ],
      [1.97013664, 1.96878413, 1.83344958, 1.97440153, 1.90559826],
      [1.92819066, 1.86220883, 1.97440153, 1.54237967, 1.96238273],
      [1.96589465, 1.9321    , 1.90559826, 1.96238273, 1.67779655]]),

'average link': array([[1.063296  , 1.49444627, 1.44474035, 1.40646432, 1.66695795],
      [1.49444627, 0.95803807, 1.60566112, 1.42552534, 1.24839209],
      [1.44474035, 1.60566112, 0.89319431, 1.68590527, 1.17948558],
      [1.40646432, 1.42552534, 1.68590527, 0.66041807, 1.64341205],
      [1.66695795, 1.24839209, 1.17948558, 1.64341205, 0.86985422]]),

'mean link': array([[0.        , 1.09195452, 1.06309636, 1.08422908, 1.34278176],
      [1.09195452, 0.        , 1.30313767, 1.15604061, 0.85760687],
      [1.06309636, 1.30313767, 0.        , 1.48119444, 0.79082844],
      [1.08422908, 1.15604061, 1.48119444, 0.        , 1.44178224],
      [1.34278176, 0.85760687, 0.79082844, 1.44178224, 0.        ]])}
```

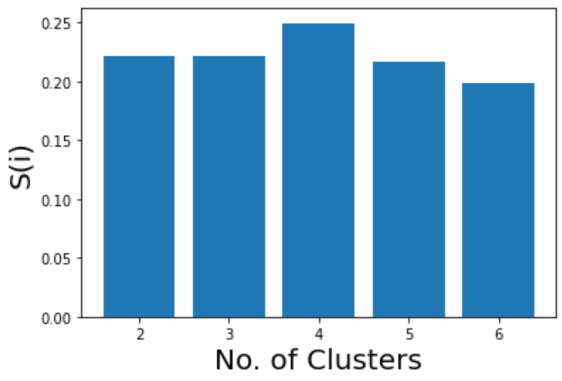
- No of clusters = 6



```
{'single link': array([[0.        , 0.46286764, 0.29401399, 0.75979074, 0.59157585,
        0.28883367],
       [0.46286764, 0.        , 0.25967829, 0.56730896, 0.29372368,
        0.49026794],
       [0.29401399, 0.25967829, 0.        , 0.4652471 , 0.8435469 ,
        0.46039289],
       [0.75979074, 0.56730896, 0.4652471 , 0.        , 0.6631247 ,
        0.78323965],
       [0.59157585, 0.29372368, 0.8435469 , 0.6631247 , 0.        ,
        0.22020801],
       [0.28883367, 0.49026794, 0.46039289, 0.78323965, 0.22020801,
        0.        ]]), 'complete link': array([[1.83041312, 1.95835921, 1.93070074, 1.92819066, 1.94598652,
        1.94032017],
       [1.95835921, 1.83702006, 1.94440736, 1.86220883, 1.9321    ,
        1.96878413],
       [1.93070074, 1.94440736, 1.74954665, 1.81133336, 1.96589465,
        1.97013664],
       [1.92819066, 1.86220883, 1.81133336, 1.54237967, 1.96238273,
        1.97440153],
       [1.94598652, 1.9321    , 1.96589465, 1.96238273, 1.67779655,
        1.90559826],
       [1.94032017, 1.96878413, 1.97013664, 1.97440153, 1.90559826,
        1.83344958]]), 'average link': array([[1.11600472, 1.5708461 , 1.16135061, 1.51141515, 1.58398219,
        1.28651436],
       [1.5708461 , 0.95803807, 1.40486215, 1.42552534, 1.24839209,
        1.60566112],
       [1.16135061, 1.40486215, 0.76087401, 1.28340215, 1.76425281,
        1.63027136],
       [1.51141515, 1.42552534, 1.28340215, 0.66041807, 1.64341205,
        1.68590527],
       [1.58398219, 1.24839209, 1.76425281, 1.64341205, 0.86985422,
        1.17948558],
       [1.28651436, 1.60566112, 1.63027136, 1.68590527, 1.17948558,
        0.89319431]]), 'mean link': array([[0.        , 1.16979749, 0.67080034, 1.1909087 , 1.22130788,
        0.80093329],
       [1.16979749, 0.        , 1.1085025 , 1.15604061, 0.85760687,
        1.30313767],
       [0.67080034, 1.1085025 , 0.        , 1.06480742, 1.55278477,
        1.39500623],
       [1.1909087 , 1.15604061, 1.06480742, 0.        , 1.44178224,
        1.48119444],
       [1.22130788, 0.85760687, 1.55278477, 1.44178224, 0.        ,
        0.79082844],
       [0.80093329, 1.30313767, 1.39500623, 1.48119444, 0.79082844,
        0.        ]])}
```

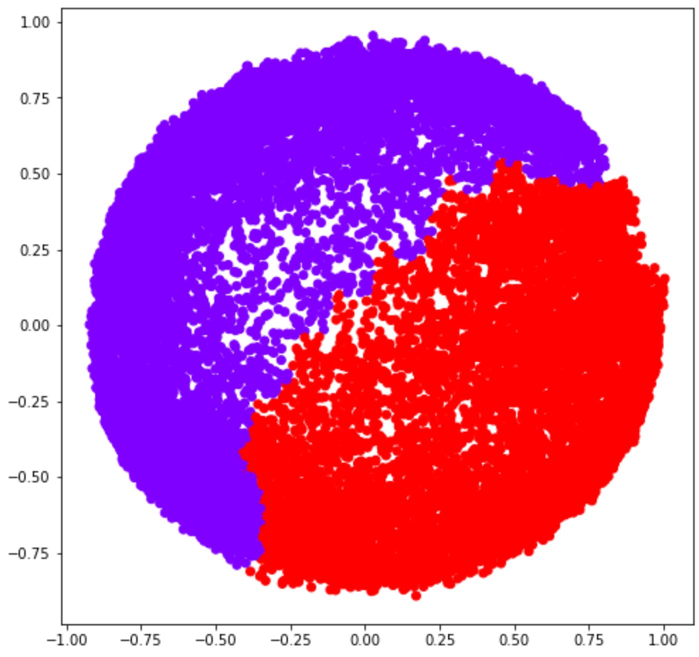
- Method 2: Forward fill to fill NaN values Suprisingly the answer in this is for `k=4` which is different from Method 1.

- Silhouette Scores

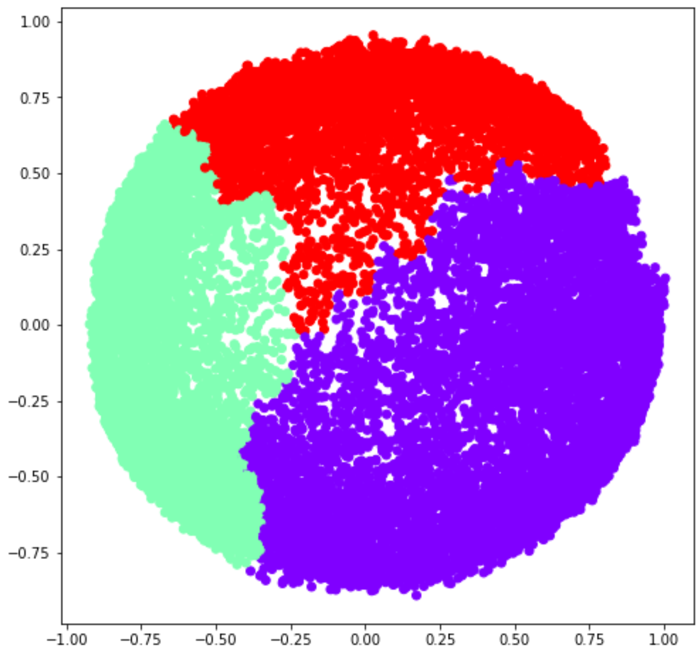


Scores: `[0.22111164675866438, 0.22096734698286216, 0.24945516810697613, 0.21685147685568962, 0.1980648052767617]`
These scores shows that 4 is the best choice for no of clusters.

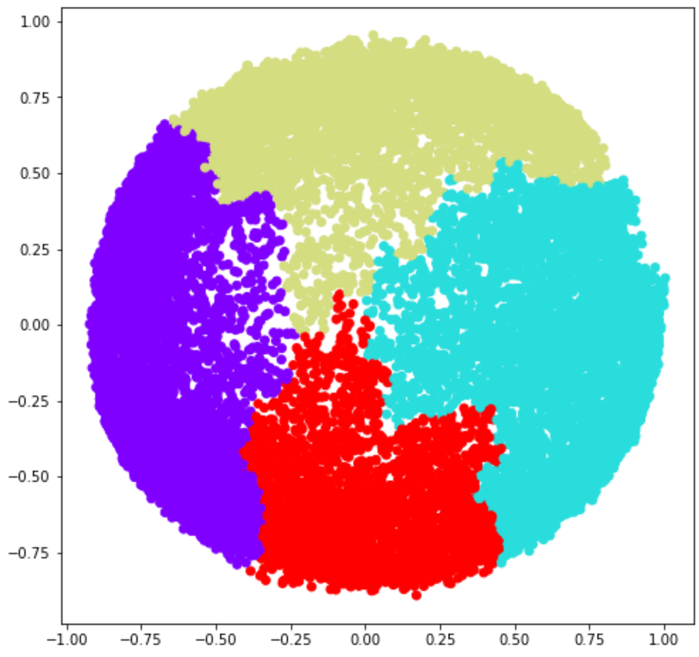
- No of clusters = 2



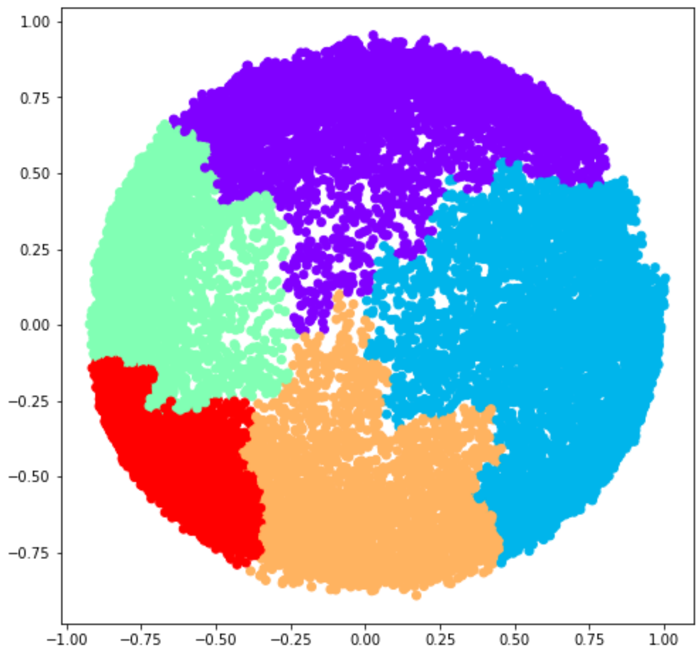
- No of clusters = 3



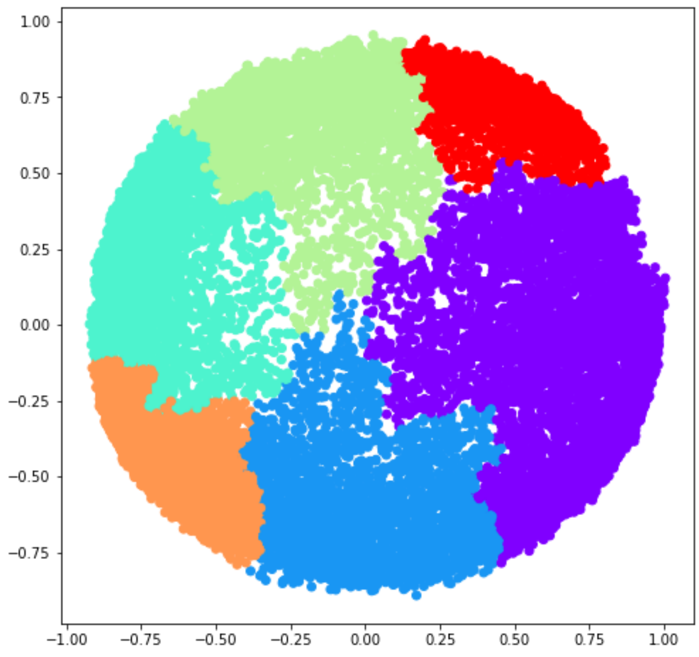
- No of clusters = 4



- No of clusters = 5



- No of clusters = 6



Out of both methods we chose the filling method because we feel that in the first method we are losing data and as we can see that the visualizations are kind of sparse relatively.

Divisive Clustering (Top-Bottom)

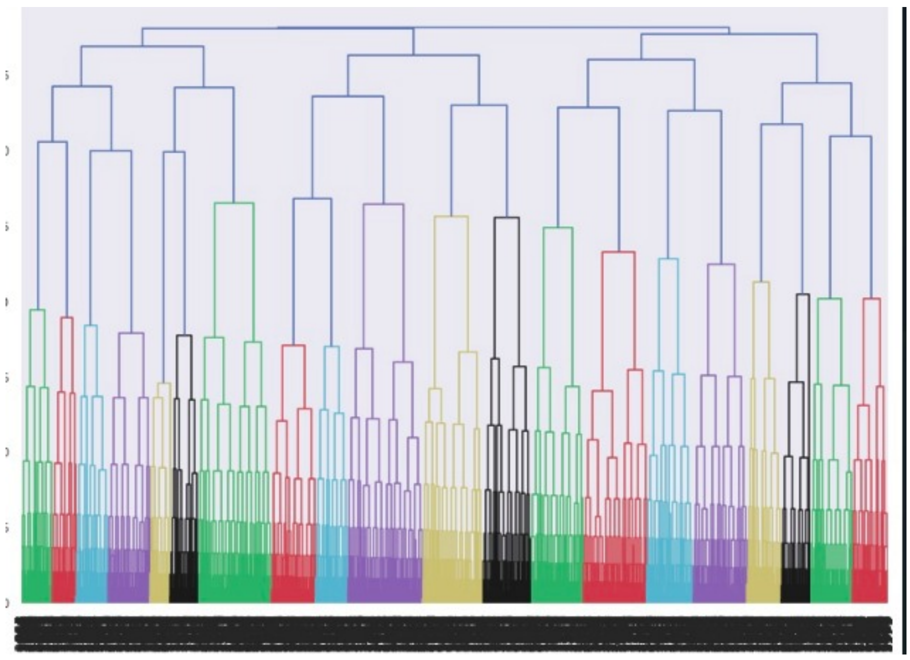
This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been splitted into singleton clusters.

Implementation

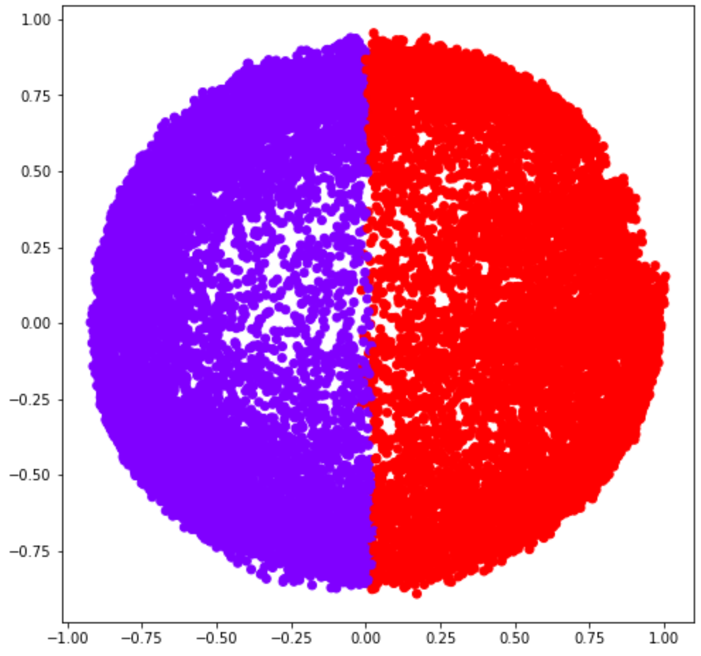
- All points are taken into 1 cluster initially.
- Recursively we then choose a cluster and divide it into 2 clusters using KMeans inbuilt function in sklearn library.
- At each step we choose the cluster with maximum sum of squared errors, i.e, sum of distance between all points in cluster and centroid of that cluster.
- To implement the above step we made use of priority queue in which we push `(-cluster_quality,index of cluster)` . Cluster quality is basically squared error and it is multipled by -1 because priority queue sorts data in increasing fashion whereas we need decreasing.

- After breaking the cluster, we add a new cluster into the 2D cluster array. We also calculate their squared mean error and push it into priority queue.
- This is done till all clusters contain only single point.

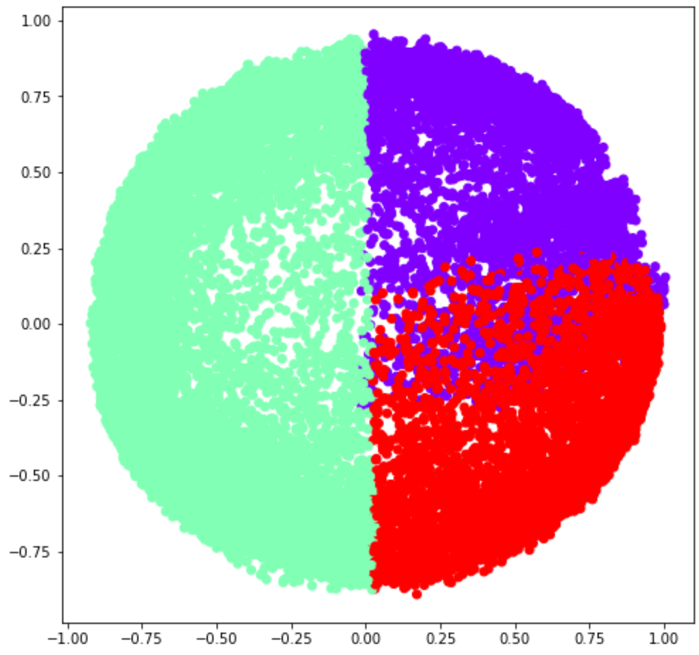
Dendogram Divisive



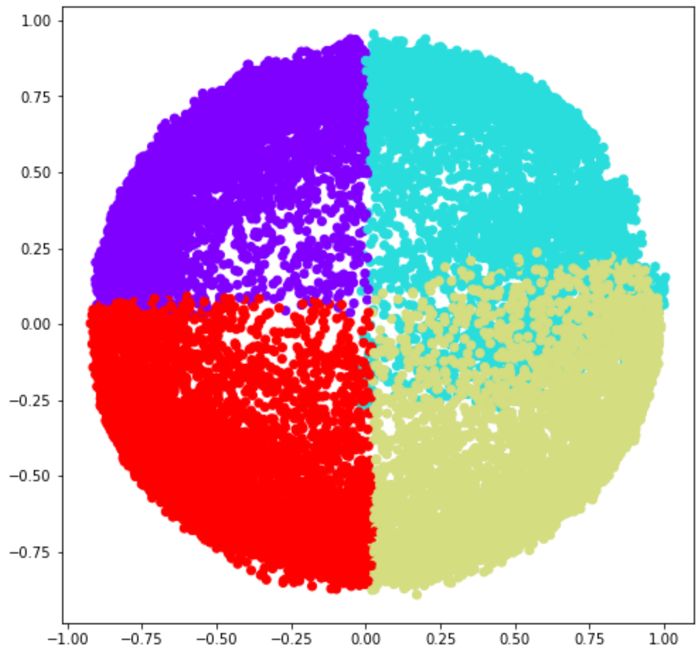
Visualization



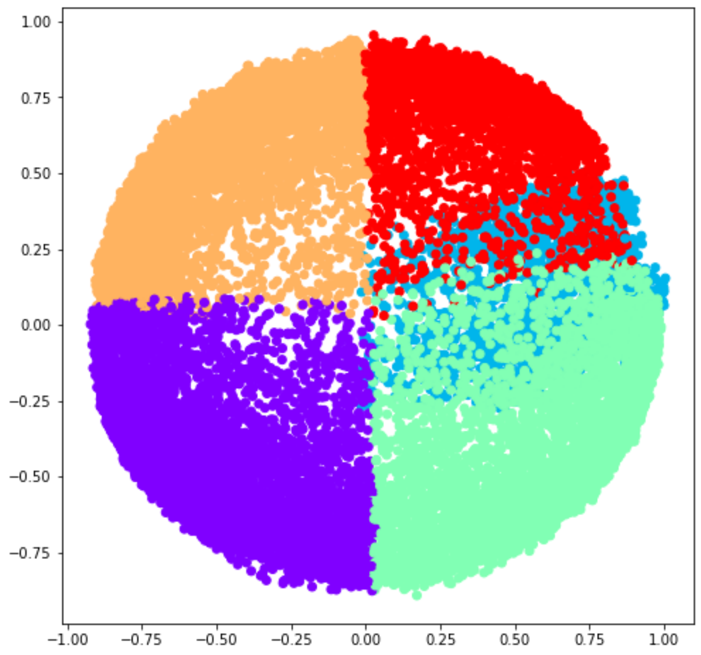
k=2


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

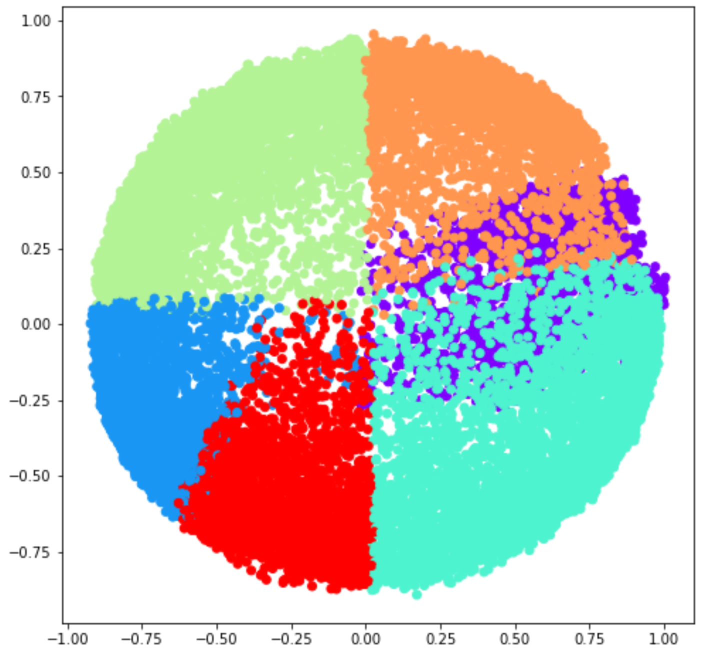
k=3


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

k=4



k=5



k=6

Agglomerative v/s Divisive Clustering

- Both don't have same dendogram but are somewhat similar.
- Agglomerative completes in polynomial time whereas divisive takes exponential time.
- Divisive is giving much better and more accurate results than agglomerative.
    - Why?
    - Agglomerative clustering makes decisions by considering the local patterns or neighbor points without initially taking into account the global distribution of data. These early decisions cannot be undone.
    - Whereas clustering takes into consideration the global distribution of data when making top-level partitioning decisions.

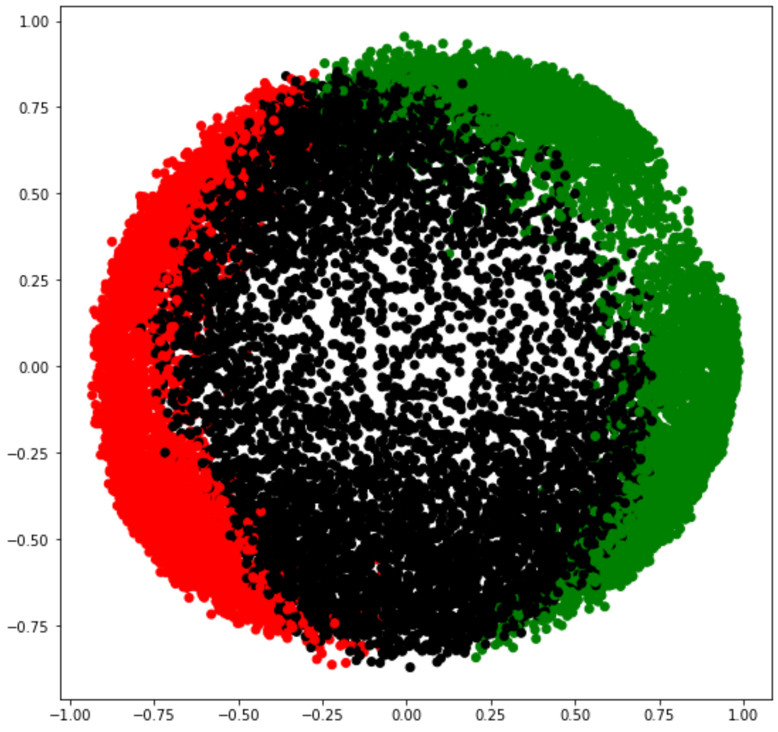
Task4: DBSCAN Clustering Algorithm

This is Density-Based Spatial Clustering of Applications with Noise clustering method.
It is based on the idea of "clusters” and “noise"

It requires 2 parameters:

- `eps` :

  - If the distance between two points is lower or equal to 'eps' then they are considered as neighbors.
  - If eps is taken too small then a large part of the data will be considered as outliers.
  - If it is chosen very large then the clusters will merge and majority of the data points will be in the same clusters.
- `MinPts` :
  - Minimum number of neighbors (data points) within eps radius.
  - Larger the dataset, the larger value of MinPts must be chosen.
  - General rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, `MinPts >= D+1` .
  - The minimum value of MinPts must be chosen at least 3.
- We have run over code for different parameter of eps on normalized data. we didn't run it on PCA data because it is loosing too much infomation about data. almost above 30%.
- for eps=0.5 and samples=128 we got this graph after projecting tha data into 2-D.



In this graph we got three differnt cluster.the size of three cluster are around 2000,3000,10000.

```
{
'single link':
       array([[0.        , 0.53643984, 0.24479287],
       [0.53643984, 0.        , 0.24920522],
       [0.24479287, 0.24920522, 0.        ]]),
'complete link':
       array([[1.0952359 , 1.5392969 , 1.9748635 ],
       [1.5392969 , 0.77590356, 1.97561555],
       [1.9748635 , 1.97561555, 1.97193189]]),
'average link':
       array([[0.54406358, 1.04298007, 1.42310079],
       [1.04298007, 0.47445689, 1.40775967],
       [1.42310079, 1.40775967, 1.37425542]]),
'mean link':
       array([[0.        , 0.91664027, 1.00805224],
       [0.91664027, 0.        , 1.00496957],
       [1.00805224, 1.00496957, 0.        ]])}
```

Finally compare the clusters formed by each of the above technique. Which method is the best according to you for clustering the given dataset? Which clustering technique made the most meaningful clusters?

- For the question `How good are the clusters?` , Inter and Intra similarities have been done in jupyter notebook of specific tasks.
- However, DBScan made the most meaningful clusters in terms of identifying arbitrary shaped clusters and the outliers. DBScan clustering efficiently handles outliers and noisy datasets. Although still kmeans is better and proves itself that it is more efficient for large datasets.
- Since DBSCAN is unidirectional, hierarchical has an advantage over DBSCAN. If a threshold error value is known below which the error is required, hierarchical is the best approach to follow because the error keeps on decreasing while traversing down the tree
- According to us, **KMeans was the best clustering method**. After analysing inter/intra class cluster similarity and visualizing the clusters, the outcome was very neat. The maximum silhouette score was also more than a considerable margin than the others. The data was also more suited for kmeans and it made the most **meaningful clusters**.