**Stellar Clustering of stars in Celestial Space**

*Deenadayalan Dasarathan, Zijie Wei, Akhil Chowdary Maddipatla,*

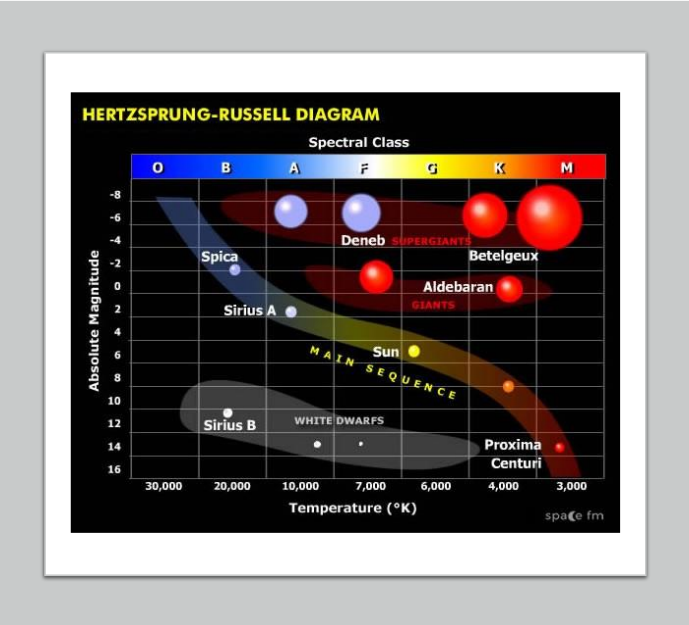
Master of Science in Data Science, Khoury College of Computer Sciences,
Northeastern University, Boston, Massachusetts

clustering_of_stars

---

## Objective

- Cluster stars based on its features.
- The stars in space follow certain graph in celestial space called Hertzsprung-Russell Diagram or simply HR Diagram.




Our project is about clustering stars based on its features.

**Background Check of Star classes!**

Stars in the universe can be classified into several groups. The groups I have used here are:

- **Brown Dwarf**
- **Red Dwarf**
- **White Dwarf**
- **Main Sequence**
- **Supergiant**
- **Hypergiant**

The classification of stars is done using a H-R Diagram as shown below above.

But in this project, we will cluster the stars based on all its features using different clustering algorithms like K-means and Agglomerative Hierarchical Clustering.

Before applying the algorithm, we will visualize the patterns hidden in the data using t-SNE. We will also use dimensionality reduction technique like PCA to obtain the lower dimensional data while preserving as much of the data variation as possible.

# Dataset

- Features: Temperature (K), Luminosity(L/Lo), Radius(R/Ro), Absolute magnitude(Mv), Star color, Spectral Class.
- Target: Star type [Integer between 0 and 5]
- Categorical features: Star color and Spectral Class
- Labels:
  - Brown Dwarf -> Star Type = 0
  - Red Dwarf -> Star Type = 1
  - White Dwarf-> Star Type = 2
  - Main Sequence -> Star Type = 3
  - Supergiant -> Star Type = 4
  - Hypergiant -> Star Type = 5

This page is just about the description of our dataset. We have 5 features and a target in this dataset, and there are 240 observations in total.

- The star groups are classified based on several characteristics of stars such as:
  - Luminosity(L/L0) **
  - Radius (R/R0) **
  - Surface Temperature(T/T0) **
  - Absolute Magnitude (Mv)
  - Spectral Class (O, B, A, F, G, K, M)
- Star Type (Red Dwarf, Brown Dwarf, White Dwarf, Main Sequence, Super Giants, Hyper Giants)

In this dataset, we have 4 numerical features and 2 categorical features. Categorical features are star color and spectral class.

We set star type as our target, which is only used for clustering performance evaluation like adjusted_rand_score, adjusted_mutual_info_score, homogeneity_score and completeness_score.

# Pre-processing

| | Temperature (K) | Luminosity(L/Lo) | Radius(R/Ro) | Absolute magnitude(Mv) | Star type | Star color | Spectral Class |
|---|---|---|---|---|---|---|---|
| 0 | 3342.1 | 352000.000 | 67.000 | -5.79 | 4 | Blue | O |
| 1 | 3667 | 0.022 | 0.380 | 10.12 | 1 | Red | M |
| 2 | 6380 | 1.350 | 0.980 | 2.93 | 3 | yellow-white | F |
| 3 | 3550 | 0.004 | 0.291 | 10.89 | 1 | Red | M |
| 4 | 24345 | 142000.000 | 57.000 | -6.24 | 4 | Blue | O |

```
### checking for class imbalance ###
data_wt_label['Star type'].value_counts()
```

```
0    40
1    40
2    40
3    40
4    40
5    40
Name: Star type, dtype: int64
```

```
### Missing values in each column ###
data_wt_label.isnull().sum()
```

```
Temperature (K)           0
Luminosity(L/Lo)          0
Radius(R/Ro)              0
Absolute magnitude(Mv)    0
Star type                 0
Star color                0
Spectral Class            0
dtype: int64
```

| | Star color_Blue | Star color_Blue White | Star color_Blue white | Star color_Blue white | Star color_Blue-White | Star color_Blue-white | Star color_Orange | Star color_Orange-Red | Star color_Pale yellow orange | Star color_Red | ... | Spectral Class_B | Spectral Class_F | Spectra Class_G |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 1 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |

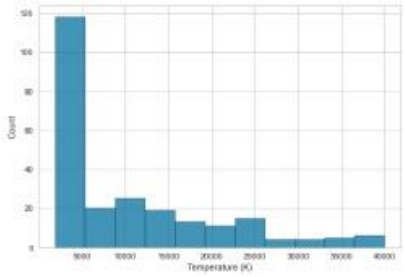
5 rows × 28 columns

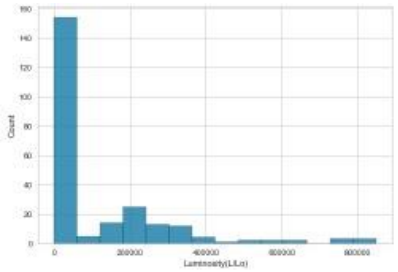
Now we are moving forward to do the pre-processing of our dataset. Since the original dataset follows a specific structure, which has already been sorted by using star type from 0 to 5, we need to shuffle the whole dataset and reset the so that the predicted model can work on random sequence of the dataset rather than follow the specific order.

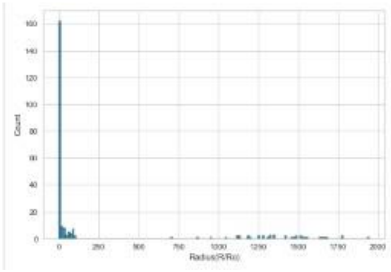
Then we were work on checking the missing value. Fortunately, we do not have any missing value in this dataset. Then we have also checked if there exist any imbalanced star types in this dataset because we do not want the dataset has too many specific star types. For example, if most of star types are main sequence, and if our model is bad, and the only result is main sequence, the final accuracy will still be good. That is why we do not want imbalanced star type. We noticed that all star types have shown up 40 times, which is perfect for us to start working on it.

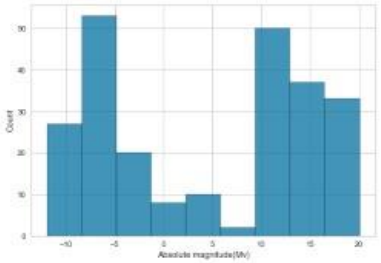
The last step of pre-processing is transfer categorical features into numerical features. So, we change the spectral class and star color into either 0 or 1. For example, if the color of the star is blue, then the column “Star color_Blue” will be 1. If not, then the value will be 0. Then we have 28 columns in total. And now, we can work on the EDA and the remaining parts.

# Univariate EDA



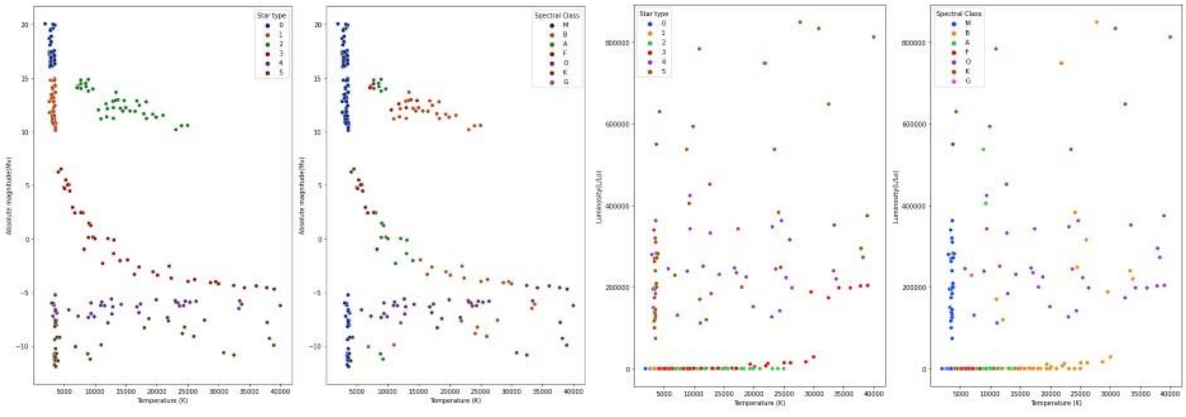



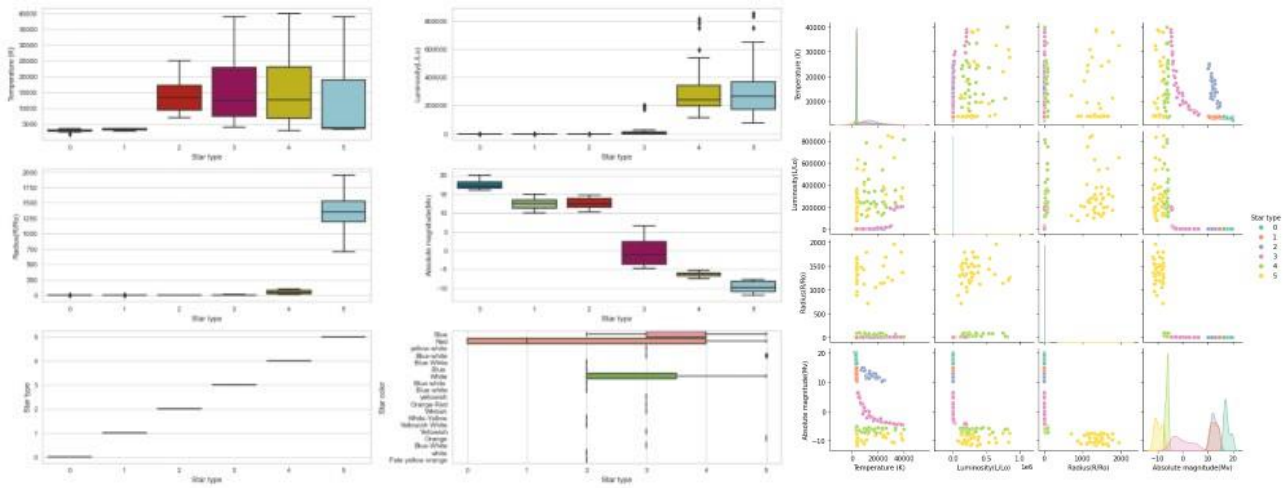

Here is the part of univariate EDA. The first image is about temperature. In this plot, we can see that most of stars' temperature are lower than 5000. The second plot shows the number of stars' luminosity. Most of stars also have relatively low luminosity. The third plot shows the radius, and almost all stars' radius is small, and some stars' radius is large. The last image is about absolute magnitude. In this part, the number of stars with 0-10 Mv are relatively low. Also, there are quite large number of stars which absolute magnitude locates between -5 to -10 and 10 to 15.

# Bi-variate EDA



In the plots above, we are trying to visualize the HR-Diagram. For the first image which is combined with Temerature vs. Absolute Magnitude which are filled by different star types and spectral classes as their hue. In the first image, we noticed that star type 0, 1, and 2 are really easy for us to separate. The star type 3, which is main sequence, follows a linear structure. In the previous slide, we have seen a HR-Diagram image which also shows that the main sequence follow a specific linear structure. So, in this case, we have successfully shown the HR-Diagram with our data. For the spectral class part, we noticed that spectral class M usually has low temerature no matter low or high of absolute magnitude, which is easy for us to separate them out from the group. But for the remaining classes O, F, B, A G, and K, a lot of them have overlapping classes. So, we think clustering stars by spectral class in Temperature vs. Absolute Magnitude graph is not available. However, no matter it is filled by star types or spectral classes, the main distribution follows the HR-Diagram.

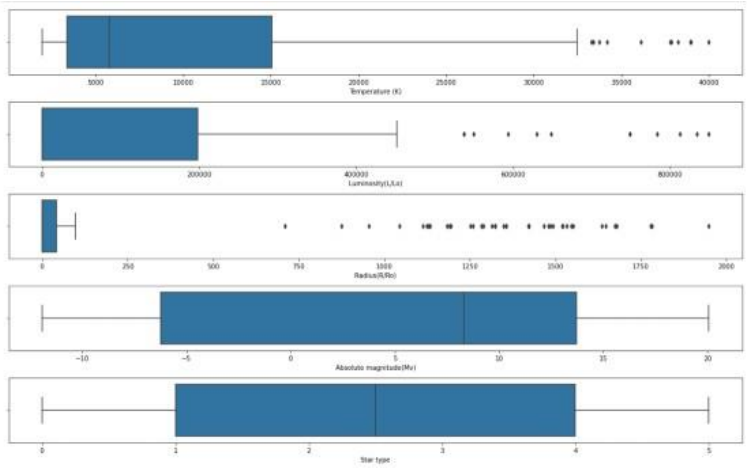
For the second graph, which is Temperature vs. Luminosity, we have totally different result. In this graph, the stars doesn't follow the HR-Diagram. For both images in second graph, we can see that most stars' locations are close to 0 luminosity, which leads to a lot of overlapping results.



In this part, we are creating several boxplots which shows the star types vs. all other 6 features. In this image, we got that most star types are red. In Star Type vs. Luminosity, it is clear to see that there are some outliers with type 3, 4, and 5. For absolute magnitude, we noticed that when the star types moving from 0 to 5, the average absolute maginitude decreases from around 17 to -10. For the radius part, we can clearly see that type 5, which is hypergiant has really large radius compare with the remaining 5 stars.

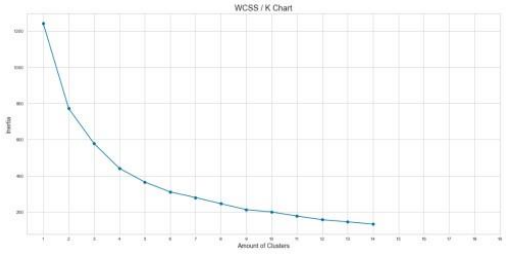
In the next graph, we are trying to show the relationship between each two of those four numerical features which are filled with star type. In that graph, we concluded that star type 0, 3, 5 are easier to separate compare with other stars since star type 1, 2, and 4 have too many overlapping features in this graph.

# Outliers



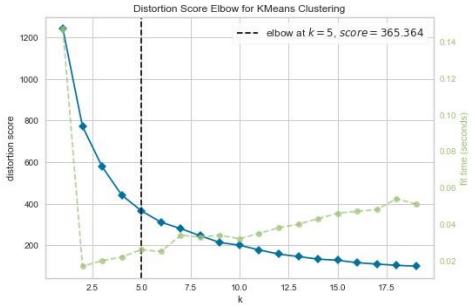

This part is about visualizing outliers. In this image, we have already done the histgrams to visualize most of the temperature, luminosity, and radius before. The mean of absolute magnitude is about 8 Mv, and most stars are usually type 1, 2, 3, and 4. But in these three boxplots, it will be more clear for us to notice that there are lots of outliers in these three boxplots. Even though these are outliers, we are not going to discard these stars since we want all these information to prove our results.

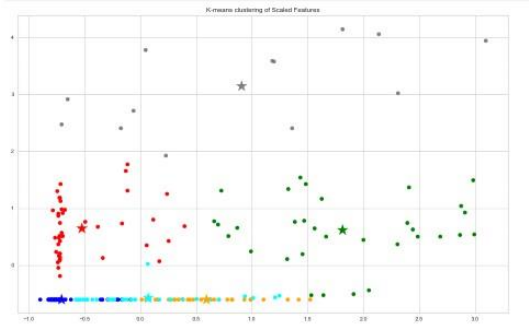
# K-Means for original dataset




```
clusterCount = np.bincount(km_labels)
print("Number of stars in each cluster: ", clusterCount)
print("ARI :", adjusted_rand_score(target, km_labels))
print("AMI :", metrics.adjusted_mutual_info_score(target, km_labels))
print("Homogeneity :", metrics.homogeneity_score(target, km_labels))
print("Completness :", metrics.completeness_score(target, km_labels))
```

```
Number of stars in each cluster:  [44 93 32 29 13 29]
ARI : 0.4393809587076734
AMI : 0.6100192392730575
Homogeneity : 0.592153132347736
Completness : 0.6566818825952188
```




In this part, we are working on the K-Means to cluster the stars. First, we plot this K chart to build an elbow plot. We only use temperature, luminosity, and absolute maginitude for K-means since HR-Diagram mentions that only these three features will be relavant in the study. Here we use 6 as our cluster numbers. Then we find that there are 44, 93, 32, 29, 13, and 29 stars in each cluster.

Adjusted random score (ARI): 0.4394

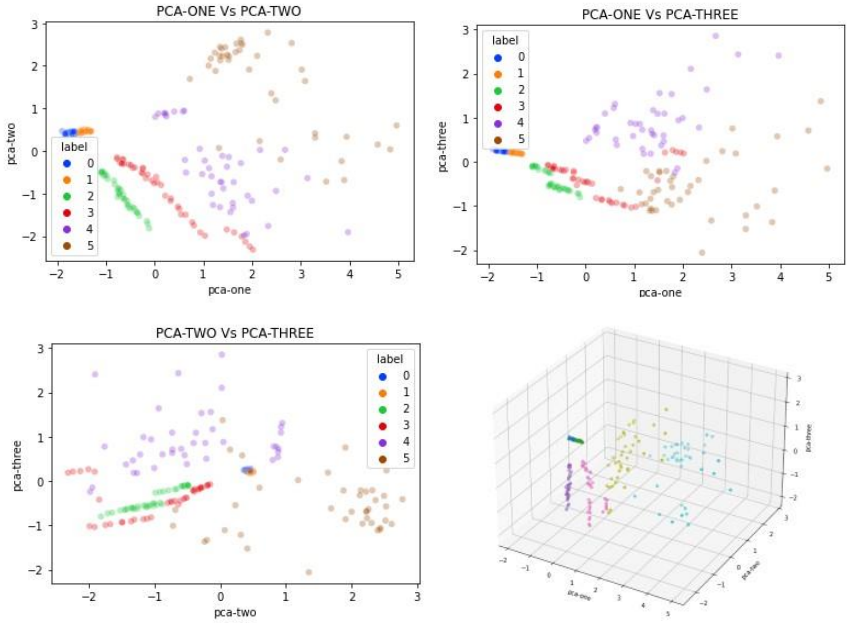
Adjusted mutual info score (AMI): 0.61

Homogeneity score: 0.5922

Completeness score: 0.6567

Given above numbers, we can see that the K-means with 6 clusters is not bad. Then we build the clustering visualization which is based on the model that we have just developed.

# PCA with 3 components

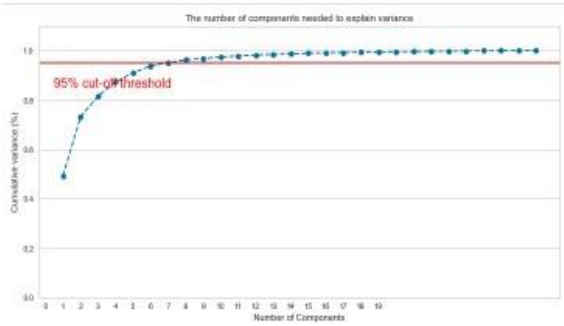


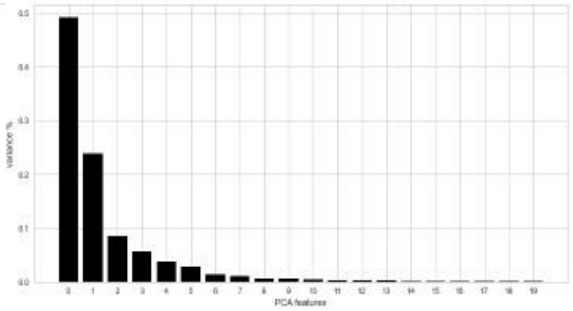

Now we are moving to the PCA part. First, we randomly selected 3 components as our PCA model to test the performance. Then we built 4 plots which are PCA-One vs. PCA-Two, PCA-One vs. PCA-Three, PCA-Two vs. PCA-Three, and a 3D plot which contains PCA-One, PCA-Two and PCA-Three.

In the first image which is PCA-One vs. PCA-Two, we can clearly see that for label 0, 1, 2, these three star types can be separated easily. For the star type 3, there are some parts which are easily separable. But there are some parts are overlapping with some star type 4. And for both star type 4 and 5, too many overlapping features exist in this image.

In the second and third image, we can still get the similar results especially in PCA-Two vs. PCA-Three since we can separate star type 0 and star type 1 easily, but the remaining features are in a mass.

# Find Number of components




```
print('Cumulative explained variation for 6 principal components: {}'.format(np.sum(pca.explained_variance_ratio_)))
```

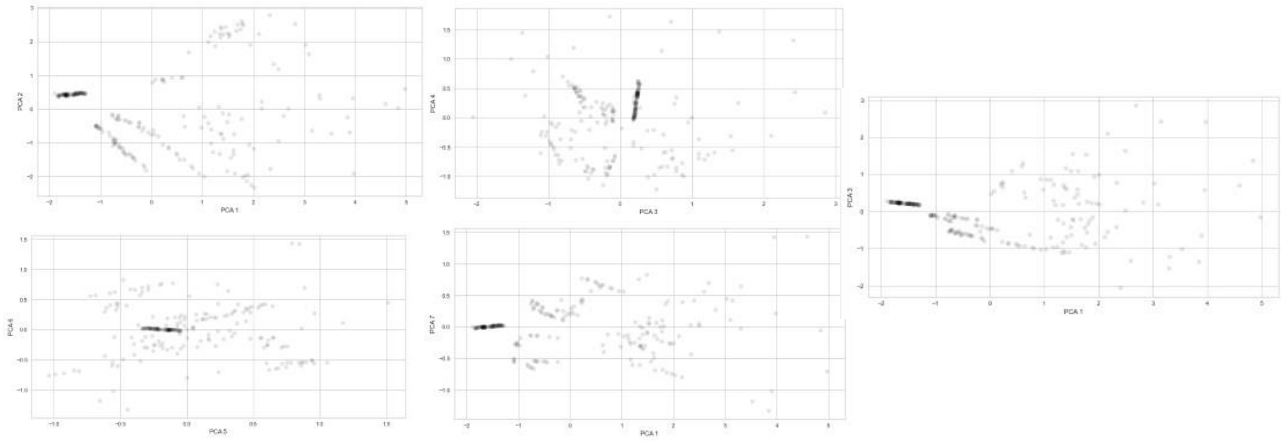
```
Cumulative explained variation for 6 principal components: 0.9375935886004889
```

A vital part of using PCA in practice is the ability to estimate how many components are needed to describe the data. This can be determined by looking at the cumulative explained variance ratio as a function of the number of components.

This curve quantifies how much of the total, 28-dimensional variance is contained within the first N components. For example, we see that with the digits the first 4 components contain approximately 80% of the variance, while we need around 15 components to describe close to 100% of the variance.

Here we see that our two-dimensional projection loses a lot of information (as measured by the explained variance) and that we would need about 6 components to retain 93% of the variance. Looking at this plot for a high-dimensional dataset can help us understand the level of redundancy present in multiple observations.

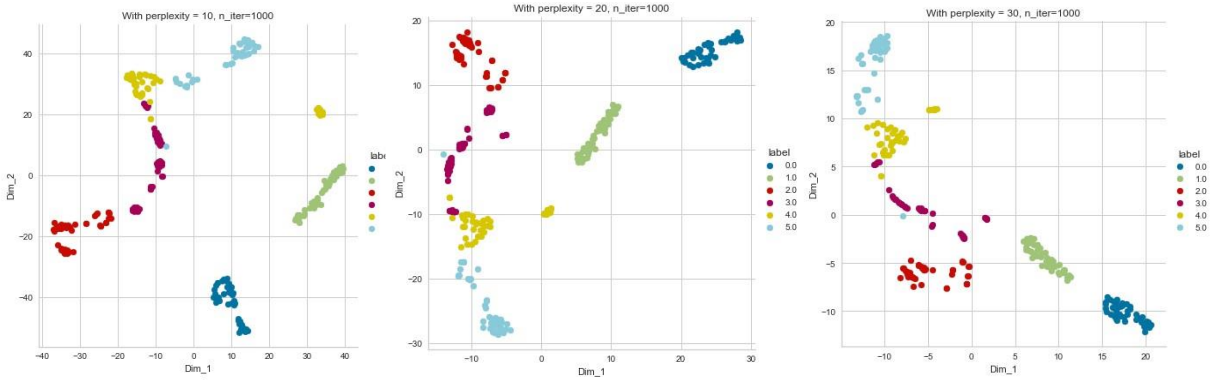
# PCA



We do 2D PCA scatter plot of differernt PCA components. The PCA plots show few clusters of samples based on their similarity. The biplot contains a lot of information and can be helpful in interpreting relationships between features. In the biplot shown, we can see that a few groups (the black dots) are close to each other indicating the possible cluster in the dimensinality reduced data. However, there are few dots that are not as close as in the other groups.

It is important to realise that if only those features that are significant are chosen, the PCA plot will be more likely to cluster stars according to their type. This is because a significant feature is one which exhibits differences between types, and PCA captures differences between star types. Therefore, using significant features for the PCA will always see some sort of grouping.

# t-SNE cluster visualization



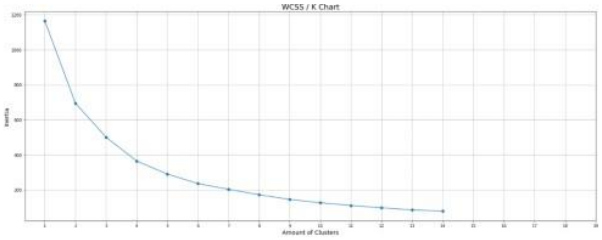

In this part, we are using t-SNE to visualize the clusters. We have built three different visualization plots which have different perplexity, which are 10, 20, and 30. All these three plots are based on the same number of components, which is 2.

We can figure out that for perplexity of 10, it is easy to separate star type 0, 1, and most parts of 5. But for type 2, 3, and 4, they are locating together which is hard for us to separate.

For the perplexity of 20, we noticed that the model performs better then perplexity of 10. Star type 0, 1, and most parts of 5 are still easily separable. We can also see different clusters of 2, 3, and 4 more clear than the previous plot.

For the perplexity of 30, the clusters become more clear than the previous two plots. There are still some overlapping features within type 2, 3, and 4. However, we can cluster most of them correctly.

# K-means after PCA




```
clusterCount = np.bincount(km5.labels_)
print("Number of stars in each cluster: ", clusterCount)
print("ARI :", adjusted_rand_score(target, km5.labels_))
print("AMI :", metrics.adjusted_mutual_info_score(target, km5.labels_))
print("Homogeneity :", metrics.homogeneity_score(target, km5.labels_))
print("Completness :", metrics.completeness_score(target, km5.labels_))
```

```
Number of stars in each cluster:  [86 28 40 75 11]
ARI : 0.5361511049393306
AMI : 0.6960518881017257
Homogeneity : 0.6317772221985913
Completness : 0.7961638839714259
```

```
clusterCount = np.bincount(km6.labels_)
print("Number of stars in each cluster: ", clusterCount)
print("ARI :", adjusted_rand_score(target, km6.labels_))
print("AMI :", metrics.adjusted_mutual_info_score(target, km6.labels_))
print("Homogeneity :", metrics.homogeneity_score(target, km6.labels_))
print("Completness :", metrics.completeness_score(target, km6.labels_))
```

```
Number of stars in each cluster:  [25 80 29 66 11 29]
ARI : 0.5716718999940754
AMI : 0.7400072502199706
Homogeneity : 0.7103582719798652
Completness : 0.7911225013502021
```

After doing the PCA, we then work on the K-means by using the PCA components.

Again, we plot this K chart to build an elbow plot. Then, we started to test with different number of clusters to build various models. The number of clusters will be either 5 or 6 here.

For the first model with cluster number of 5, we find that there are 86, 28, 40, 75, and 11 stars in each cluster.

Adjusted random score (ARI): 0.5362

Adjusted mutual info score (AMI): 0.6961

Homogeneity score: 0.6318

Completeness score: 0.7962

Then we move forward to the cluster number of 6. We find that there are 25, 80, 29, 66, 11, and 29 stars in each cluster.

Adjusted random score (ARI): 0.5717

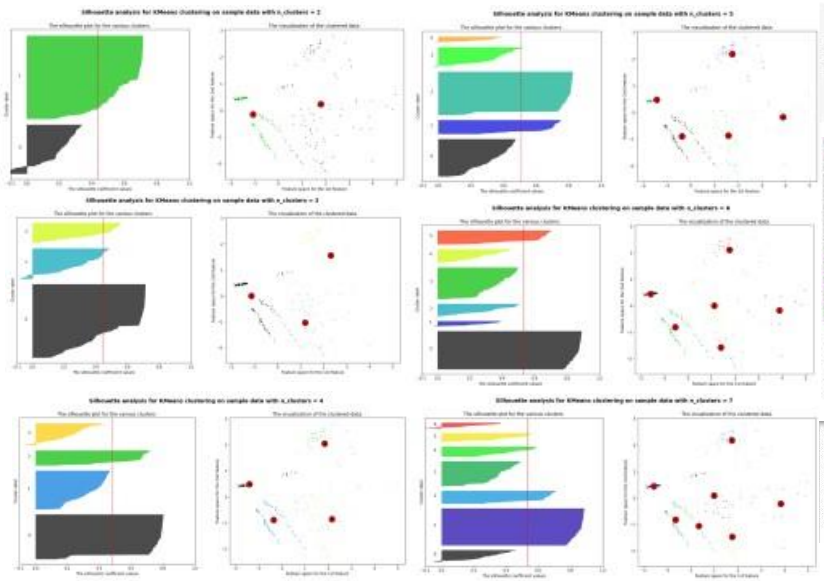
Adjusted mutual info score (AMI): 0.74

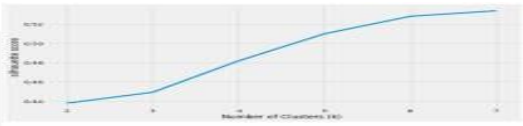
Homogeneity score: 0.7104

Completeness score: 0.7911

Given above numbers, we can see that the K-means with 6 clusters is better compare with 5 clusters.

# Silhouette analysis





```
### K-means with 6 clusters ###
kmeans = KMeans(n_clusters=6, random_state=0,max_iter=600, algorithm ='auto')
kmeans.fit(pca_matrix)
centroidr = kmeans.cluster_centers_
km_labels = kmeans.labels_

clusterCount = np.bincount(km_labels)
print("Number of stars in each cluster: ", clusterCount)
print("ARI :", adjusted_rand_score(target, km_labels))
print("AMI :", metrics.adjusted_mutual_info_score(target, km_labels))
print("Homogeneity :", metrics.homogeneity_score(target, km_labels))
print("Completness :", metrics.completeness_score(target, km_labels))
```

```
Number of stars in each cluster:  [25 80 29 66 11 29]
ARI : 0.5716718999940754
AMI : 0.7400072502199706
Homogeneity : 0.7103582719798652
Completness : 0.7911225013502021
```

```
For n_clusters = 2 The average silhouette_score is : 0.44145029979844885
For n_clusters = 3 The average silhouette_score is : 0.4492887607206467
For n_clusters = 4 The average silhouette_score is : 0.4818149958288363
For n_clusters = 5 The average silhouette_score is : 0.5107202399706945
For n_clusters = 6 The average silhouette_score is : 0.5277316158554296
For n_clusters = 7 The average silhouette_score is : 0.5283959815661007
```

Furthermore, we also did the silhouette analysis. Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually.

Here we are trying to test by using 2, 3, 4, 5, 6, 7 clusters.

For n_clusters = 2 The average silhouette_score is : 0.44145029979844885

For n_clusters = 3 The average silhouette_score is : 0.4492887607206467

For n_clusters = 4 The average silhouette_score is : 0.4818149958288363

For n_clusters = 5 The average silhouette_score is : 0.5107202399706945

For n_clusters = 6 The average silhouette_score is : 0.5277316158554296

For n_clusters = 7 The average silhouette_score is : 0.5283959815661007

From the above analysis, we can see there is not much increase in silhouette score after n_clusters = 6. Hence using the Kmeans model we can cluster the stars into 6 groups.
In the previous elbow method there was an ambiguty in n_clusters value between 5 and 6. But using the silhouette we confirmed the n_clusters value as 6. Kmeans n_clusters= 6,

Number of stars in each cluster:  [25 80 29 66 11 29]

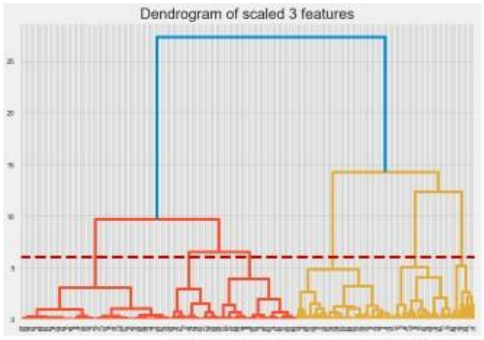
ARI : 0.5716718999940754

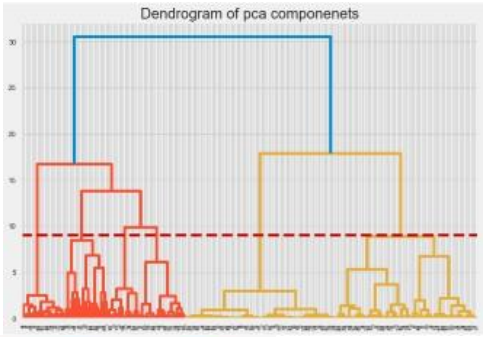
AMI : 0.7400072502199706

Homogeneity : 0.7103582719798652

Completness : 0.7911225013502021

# Hierarchical Clustering




```
### Hierarchical clustering ###
cluster = AgglomerativeClustering(n_clusters=6, affinity='euclidean', linkage='ward')
cluster.fit(X_3_features)
agg_labels = cluster.labels_
agg_labels

clusterCount = np.bincount(agg_labels)
print("Number of stars in each cluster: ", clusterCount)
print("ARI :", adjusted_rand_score(target, agg_labels))
print("AMI :", metrics.adjusted_mutual_info_score(target, agg_labels))
print("Homogeneity :", metrics.homogeneity_score(target, agg_labels))
print("Completness :", metrics.completeness_score(target, agg_labels))
```

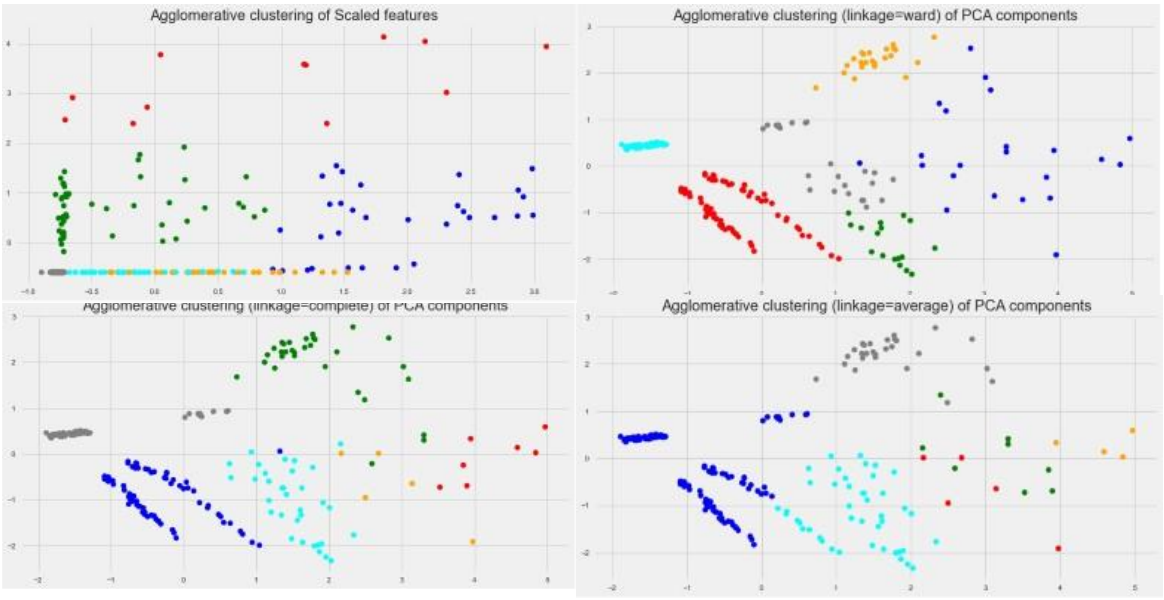
```
Number of stars in each cluster:  [12 31 51 40 80 26]
ARI : 0.968626842898522
AMI : 0.7144141908324848
Homogeneity : 0.6947878794088493
Completness : 0.7550052270946173
```

We can clearly visualize the steps of hierarchical clustering. More the distance of the vertical lines in the dendrogram, more the distance between those clusters. We apply hierarchical clustering model both scaled features as well as the dimensionality reduced PCA componenets.

In the dendrogram, we set a threshold distance and draw a horizontal line (Generally, we try to set the threshold in such a way that it cuts the tallest vertical line). Let's set this threshold as 6 and 10 for both the model and draw a horizontal line.

The number of clusters will be the number of vertical lines which are being intersected by the line drawn using the threshold. So we will have 6 clusters in both the scaled and pca component data.

# Hierarchical Clustering




We have analyzed the clusters using AgglomerativeClustering model with n_clusters=6 and different linkage attributes like ward, complete and average. Out of which AgglomerativeClustering(n_clusters=6, affinity='euclidean', linkage='ward') performed better with the following metric results,

Number of stars in each cluster: [74 22 17 23 24 80]

ARI: 0.5510078195356501

AMI: 0.7320035531298634

Homogeneity: 0.6992493106238975

Completeness: 0.7877040770645723

On comparing the results of Kmeans and AgglomerativeClustering models, both showed a similar performance, though we expected Kmeans to underperform. This could be because of some linearity in the features. Also, the stars of certain type are spread out, this might have caused Hierarchical clustering model to misclassify the start and slightly deteriorate its performance.