Links

- https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/
- https://medium.com/@sametgirgin/hierarchical-clustering-model-in-5-steps-with-python-6c45087d4318

Kmeans

Steps

- S1: Pick K random points as cluster centers called centeriod
  - We randomly pick K cluster centers(centroids). Let's assume these are c1,c2,...,ck, and we can say that;
    - C is the set of all centroids.
    - C=c1,c2,...,ck
- S2: Assign each xi to nearest cluster by calculating its distance to each centeriod
  - In this step we assign each input value to closest center. This is done by calculating Euclidean(L2) distance between the point and the each centroid.
    - ci∈C
    - arg {min: dist(ci,x)^2} — dist(.) is the Euclidean distance.
- S3: Find new cluster center by taking the average of the assigned points
  - find the new centroid by taking the average of all the points assigned to that cluster.
    - ci=1/ |Si| * ∑xi — Si is the set of all points assigned to the ith cluster.
- S4: Repeat S2 & S3 until non of the cluster assignment change
  - ntil our clusters remain stable, we repeat the algorithm.

Others

- Choosing the Value of K
  - We often know the value of K. In that case we use the value of K. Else we use the Elbow Method.
  - run the algorithm for different values of K(say K = 10 to 1) and plot the K values against SSE(Sum of Squared Errors). And select the value of K for the elbow point as shown in the figure.

Hierarchical

Divisive

- Initially, all data is in the same cluster, and the largest cluster is split until every object is separate. There can be many methods of splitting
- https://www.analyticsvidhya.com/blog/2019/05/beginners-guide-hierarchical-clustering

Agglomerative

- method allows the clusters to be read from bottom to top and it follows this approach so that the program always reads from the sub-component first then moves to the parent whereas, divisive uses top-bottom approach in which the parent is visited first then the child
- https://www.geeksforgeeks.org/implementing-agglomerative-clustering-using-sklearn/
- x
  - from sklearn.cluster import AgglomerativeClustering
  - from sklearn.preprocessing import StandardScaler, normalize
  - from sklearn.metrics import silhouette_score
  - import scipy.cluster.hierarchy as shc
- x
  - data = pd.read_csv('file.csv')
  - data1 = X.drop('column', axis = 1)
  - data1.dropna(inplace = True)
- scaling
  - scaler = StandardScaler()
  - X_scaled = scaler.fit_transform(X)
  - X_normalized = normalize(X_scaled)
- Dendrogram (to find the optimal no of clusters and also view the points)
  - x1
    - import scipy.cluster.hierarchy as shc
    - plt.figure(figsize=(10, 7))
    - plt.title("Dendrograms")
    - dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
  - The x-axis contains the samples and y-axis represents the distance between these samples. The vertical line with maximum distance is the blue line and hence we can decide a threshold of y1 and cut the dendrogram
  - x
    - plt.figure(figsize=(10, 7))
    - plt.title("Dendrograms")
    - dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
    - plt.axhline(y=6, color='r', linestyle='--')
  - We have Cn clusters as this line cuts the dendrogram at n points. Let's now apply hierarchical clustering for n clusters:
- Clusters
  - x
    - from sklearn.cluster import AgglomerativeClustering
    - cluster = AgglomerativeClustering(n_clusters=Cn, affinity='euclidean', linkage='ward')
    - cluster.fit_predict(data_scaled)
- Plot
  - plt.figure(figsize=(10, 7))
  - plt.scatter(data_scaled['column1'], data_scaled['column2'], c=cluster.labels_)

Python - Clustering

mtcars dataset

KMeans

- #libraries
  - x
    - import numpy as np
    - import pandas as pd
    - import matplotlib.pyplot as plt
    - from pydataset import data
    - import seaborn as sns
  - c
    - from sklearn.cluster import KMeans
    - from kneed import KneeLocator
    - from sklearn.preprocessing import MinMaxScaler
    - from sklearn.preprocessing import StandardScaler
  - cn
    - from kneed import KneeLocator
    - from sklearn.metrics import silhouette_score
- Data
  - data = read.csv(....)
  - Select only numeric columns
  - data1 = data1.select_dtypes(exclude=['object'])
- Scaling
  - from skleam.preprocessing import StandardScaler
  - data2_scaled = StandardScaler().fit_transform(data2)
- Model
  - kmeans = KMeans(n_clusters=3).fit(data2_scaled)
- Output
  - kmeans.inertia_ — Error : 0 to ..; As less as possible
  - kmeans.cluster_centers_ — Mean values of values of rows of clusters
  - kmeans.labels_ — which row - which cluster no
  - kmeans.n_iter — Number of iterations run.
- Results
  - clusterNos = kmeans.labels_
  - data1.groupby([clusterNos]).mean() — Mean of each column in each cluster (Original units)
- No of Clusters
  - x