# # Clustering

# ##Index:

#

# 1. K-means clustering implement

# 2. Visualization

# 3. Repeat with scaled data

# 4. Clustering evaluation

# 5. DBSCAN clustering

# 6. Evaluation as explained by Andrew NJ.

# Load beer dataset

import pandas as pd

url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/beer.txt'

beer = pd.read\_csv(url, sep=' ')

beer

#drop undesired columns and build training set

X = beer.drop('name', axis=1)

#1. K-means clustering implement (with 3 clusters)

from sklearn.cluster import KMeans

km = KMeans(n\_clusters=3, random\_state=1)

km.fit(X)

# review the cluster labels

# review the cluster centers

km.cluster\_centers\_

km.labels\_

# save the cluster labels and sort by cluster

beer['cluster'] = km.labels\_

beer.sort('cluster')

# calculate the mean of each feature for each cluster

beer.groupby('cluster').mean()

centers = beer.groupby('cluster').mean()

#2. Visualization

# create a "colors" array for plotting

import numpy as np

colors = np.array(['red', 'green', 'blue', 'yellow'])

# scatter plot of calories versus alcohol, colored by cluster (0=red, 1=green, 2=blue)

plt.scatter(beer.calories, beer.alcohol, c=colors[beer.cluster], s=50)

# cluster centers, marked by "+"

plt.scatter(centers.calories, centers.alcohol, linewidths=3, marker='+', s=300, c='black')

# add labels

plt.xlabel('calories')

plt.ylabel('alcohol')

# scatter plot matrix (0=red, 1=green, 2=blue)

pd.scatter\_matrix(X, c=colors[beer.cluster], figsize=(10,10), s=100)

#3.Repeat with scaled data

# Standardize features by removing the mean and scaling to unit variance

# center and scale the data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# K-means with 3 clusters on scaled data

km = KMeans(n\_clusters=3, random\_state=1)

km.fit(X\_scaled)

# save the cluster labels and sort by cluster

beer['cluster'] = km.labels\_

beer.sort('cluster')

# What are the "characteristics" of each cluster?

# review the cluster centers

beer.groupby('cluster').mean()

# scatter plot matrix of new cluster assignments (0=red, 1=green, 2=blue)

pd.scatter\_matrix(X, c=colors[beer.cluster], figsize=(10,10), s=100)

#4. Clustering evaluation.

# A Silhouette Coefficient is calculated for \*\*each observation\*\*:

#

# $$SC = \frac{b-a} {max(a, b)}$$

#

# - a = mean distance to all other points in \*\*its cluster\*\*

# - b = mean distance to all other points in \*\*the next nearest cluster\*\*

#

# It ranges from -1 (worst) to 1 (best). A \*\*global score\*\* is calculated by taking the mean score for all observations.

# calculate SC for K=3

from sklearn import metrics

metrics.silhouette\_score(X\_scaled, km.labels\_)

# calculate SC for K=2 through K=19

k\_range = range(2, 20)

scores = []

for k in k\_range:

km = KMeans(n\_clusters=k, random\_state=1)

km.fit(X\_scaled)

scores.append(metrics.silhouette\_score(X\_scaled, km.labels\_))

# plot the results

plt.plot(k\_range, scores)

plt.xlabel('Number of clusters')

plt.ylabel('Silhouette Coefficient')

plt.grid(True)

# K-means with 4 clusters on scaled data

km = KMeans(n\_clusters=4, random\_state=1)

km.fit(X\_scaled)

beer['cluster'] = km.labels\_

#5. DBSCAN clustering

# DBSCAN with eps=1 and min\_samples=3

from sklearn.cluster import DBSCAN

db = DBSCAN(eps=1, min\_samples=3)

db.fit(X\_scaled)

# review the cluster labels

db.labels\_

# save the cluster labels and sort by cluster

beer['cluster'] = db.labels\_

beer.sort('cluster')

# review the cluster centers

beer.groupby('cluster').mean()

# scatter plot matrix of DBSCAN cluster assignments (0=red, 1=green, 2=blue, -1=yellow)

pd.scatter\_matrix(X, c=colors[beer.cluster], figsize=(10,10), s=100)

#6. Evaluation as explained by Andrew NJ.

#for each feature, find the variance

#while calculating variance the mean will be the mean distance of all examples belongs to a #cluster

X\_test=pd.DataFrame(X\_scaled)

X\_test ['cluster'] = km.labels\_

X=X\_test

p=X.groupby(‘cluster’).var()

q=X.cluster.value\_counts()

p=np.matrix(p)

q=np.matrix(q)

m= X.shape[0]

variance=(q\*p)/m