Unsupervised Machine Learning: Clustering Techniques on Dry Bean Dataset

Nariman Pashayev

Contents

Main Objectives

- There is a wide range of genetic diversity of dry bean which is the most produced one among the edible legume crops in the world. Seed quality is definitely influential in crop production. Therefore, seed classification is essential for both marketing and production to provide the principles of sustainable agricultural systems. The primary objective of this study is to provide a method for obtaining uniform seed varieties from crop production, which is in the form of population, so the seeds are not certified as a sole variety. Thus, a computer vision system was developed to distinguish seven different registered varieties of dry beans with similar features in order to obtain uniform seed classification. For the classification model, images of 13,611 grains of 7 different registered dry beans were taken with a high-resolution camera. A total of 16 features, 12 dimensions and 4 shape forms were obtained from the grains.

- Originally this dataset was obtained for predicting Classification problem. But, in my analysis I used this as Clustering analysis and for this purpose I assumed I don’t know the Dry Bean Classes, hence Unsupervised Learning.

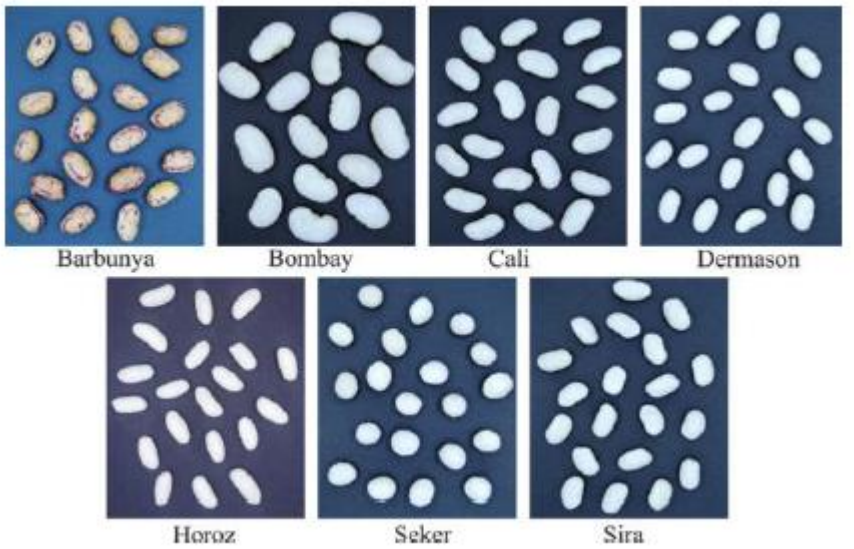
- Then, I compared my obtained results with the experimentally obtained Classes to measure my models’ performance, but in real cases it is not usually possible, since this is Unsupervised Learning

- Link to Dataset: https://www.muratkoklu.com/datasets/

1. DATA DESCRIPTION

Dataset Description

- Dataset contains 13611 samples and 16 features plus target variable “Class”
- Target variable contains 7 different classes : Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira



	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactn
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.988856	0.958027	0.913
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272750	0.783968	0.984986	0.887034	0.953
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.989559	0.947849	0.908
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.976696	0.903936	0.928
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.990893	0.984877	0.970
...	...	...	...	...	...	...	...	...	...	...	...	
13606	42097	759.696	288.721612	185.944705	1.552728	0.765002	42508	231.515799	0.714574	0.990331	0.916603	0.801
13607	42101	757.499	281.576392	190.713136	1.476439	0.735702	42494	231.526798	0.799943	0.990752	0.922015	0.822
13608	42139	759.321	281.539928	191.187979	1.472582	0.734065	42569	231.631261	0.729932	0.989899	0.918424	0.822
13609	42147	763.779	283.382636	190.275731	1.489326	0.741055	42667	231.653248	0.705389	0.987813	0.907906	0.817
13610	42159	772.237	295.142741	182.204716	1.619841	0.786693	42600	231.686223	0.788962	0.989648	0.888380	0.784

Dataset Description-Attribute Information

	Feature Name	Feature Description
1	Area (A)	The area of a bean zone and the number of pixels within its boundaries.
2	Perimeter (P)	Bean circumference is defined as the length of its border.
3	Major axis length (L)	The distance between the ends of the longest line that can be drawn from a bean.
4	Minor axis length (l)	The longest line that can be drawn from the bean while standing perpendicular to the main axis.
5	Aspect ratio (K)	Defines the relationship between L and l.
6	Eccentricity (Ec)	Eccentricity of the ellipse having the same moments as the region.
7	Convex area (C)	Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
8	Equivalent diameter (Ed)	The diameter of a circle having the same area as a bean seed area.
9	Extent (Ex)	The ratio of the pixels in the bounding box to the bean area.
10	Solidity (S)	Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11	Roundness (R)	Calculated with the following formula: (4piA)/(P^2)
12	Compactness (CO):	Measures the roundness of an object: Ed/L
13	ShapeFactor1 (SF1)	
14	ShapeFactor2 (SF2)	
15	ShapeFactor3 (SF3)	
16	ShapeFactor4 (SF4)	
17	Class	Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira

Dataset Description-Basic Statistics

	count	mean	std	min	25%	50%	75%	max
Area	13611.0	53048.284549	29324.095717	20420.000000	36328.000000	44652.000000	61332.000000	254616.000000
Perimeter	13611.0	855.283459	214.289696	524.736000	703.523500	794.941000	977.213000	1985.370000
MajorAxisLength	13611.0	320.141867	85.694186	183.601165	253.303633	296.883367	376.495012	738.860153
MinorAxisLength	13611.0	202.270714	44.970091	122.512653	175.848170	192.431733	217.031741	460.198497
AspectRation	13611.0	1.583242	0.246678	1.024868	1.432307	1.551124	1.707109	2.430306
Eccentricity	13611.0	0.750895	0.092002	0.218951	0.715928	0.764441	0.810466	0.911423
ConvexArea	13611.0	53768.200206	29774.915817	20684.000000	36714.500000	45178.000000	62294.000000	263261.000000
EquivDiameter	13611.0	253.064220	59.177120	161.243764	215.068003	238.438026	279.446467	569.374358
Extent	13611.0	0.749733	0.049086	0.555315	0.718634	0.759859	0.786851	0.866195
Solidity	13611.0	0.987143	0.004660	0.919246	0.985670	0.988283	0.990013	0.994677
roundness	13611.0	0.873282	0.059520	0.489618	0.832096	0.883157	0.916869	0.990685
Compactness	13611.0	0.799864	0.061713	0.640577	0.762469	0.801277	0.834270	0.987303
ShapeFactor1	13611.0	0.006564	0.001128	0.002778	0.005900	0.006645	0.007271	0.010451
ShapeFactor2	13611.0	0.001716	0.000596	0.000564	0.001154	0.001694	0.002170	0.003665
ShapeFactor3	13611.0	0.643590	0.098996	0.410339	0.581359	0.642044	0.696006	0.974767
ShapeFactor4	13611.0	0.995063	0.004366	0.947687	0.993703	0.996386	0.997883	0.999733

2. Exploratory Data Analysis

and Feature Engineering

EDA-Checking for Data Types and Missing Values

- All features hold numeric values (both int and float type), and the last column is "object" type.
- There are not missing values in the dataset.

```
# Lets check for null (missing) values
df.isna().sum()
```

```
Area                0
Perimeter           0
MajorAxisLength     0
MinorAxisLength     0
AspectRation        0
Eccentricity        0
ConvexArea          0
EquivDiameter       0
Extent              0
Solidity            0
roundness           0
Compactness         0
ShapeFactor1        0
ShapeFactor2        0
ShapeFactor3        0
ShapeFactor4        0
Class               0
dtype: int64
```

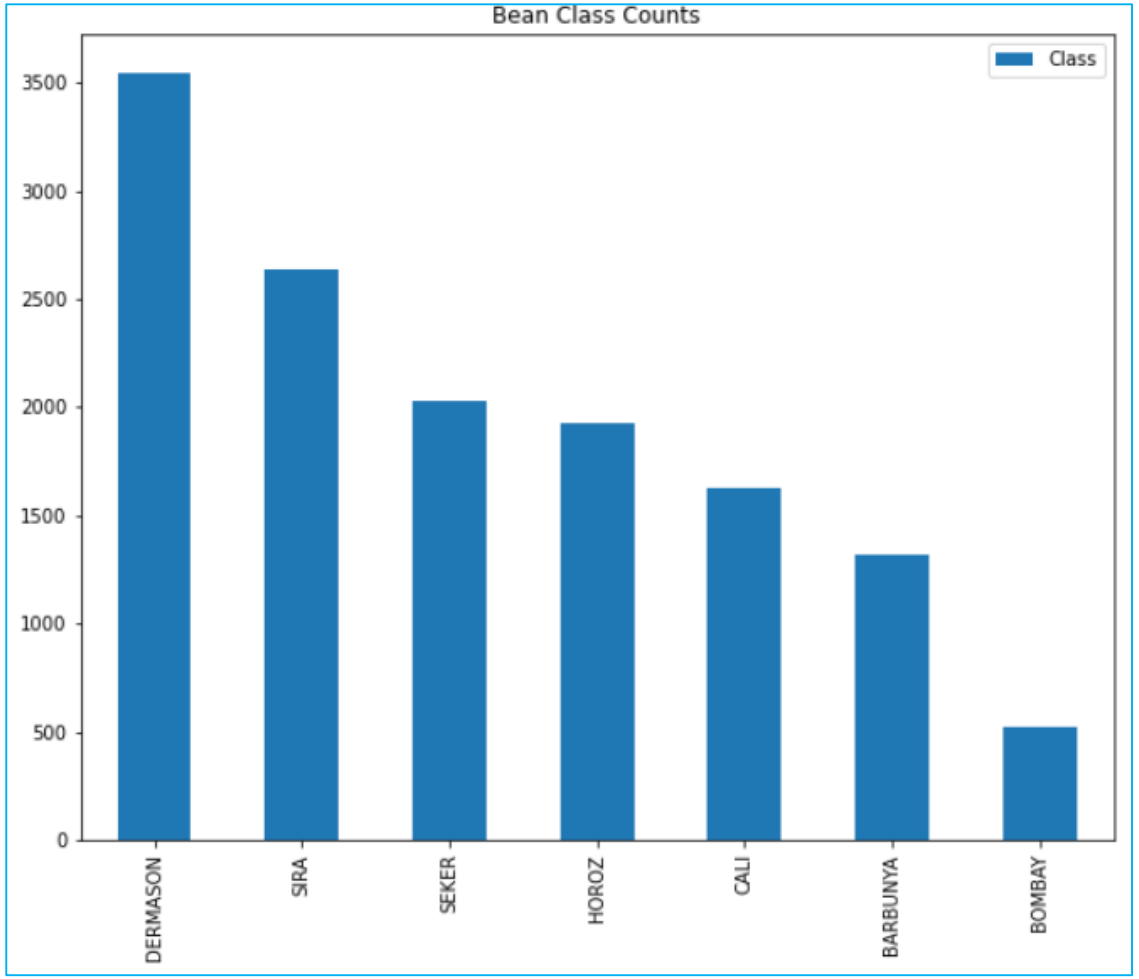
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13611 entries, 0 to 13610
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Area             13611 non-null  int64
 1   Perimeter        13611 non-null  float64
 2   MajorAxisLength  13611 non-null  float64
 3   MinorAxisLength  13611 non-null  float64
 4   AspectRation     13611 non-null  float64
 5   Eccentricity     13611 non-null  float64
 6   ConvexArea       13611 non-null  int64
 7   EquivDiameter    13611 non-null  float64
 8   Extent           13611 non-null  float64
 9   Solidity         13611 non-null  float64
 10  roundness        13611 non-null  float64
 11  Compactness      13611 non-null  float64
 12  ShapeFactor1     13611 non-null  float64
 13  ShapeFactor2     13611 non-null  float64
 14  ShapeFactor3     13611 non-null  float64
 15  ShapeFactor4     13611 non-null  float64
 16  Class            13611 non-null  object
dtypes: float64(14), int64(2), object(1)
memory usage: 1.8+ MB
```

EDA-Checking for Target Variable

- Since this is Clustering example, we will not need y-variable (Class variable) as an input in my clustering analysis.
- However, lets check how many classes we have and how many samples of each class we have
- As you can see the data is not evenly split between target values, quite imbalanced.

```
class_counts=df['Class'].value_counts().to_frame()
class_counts
```

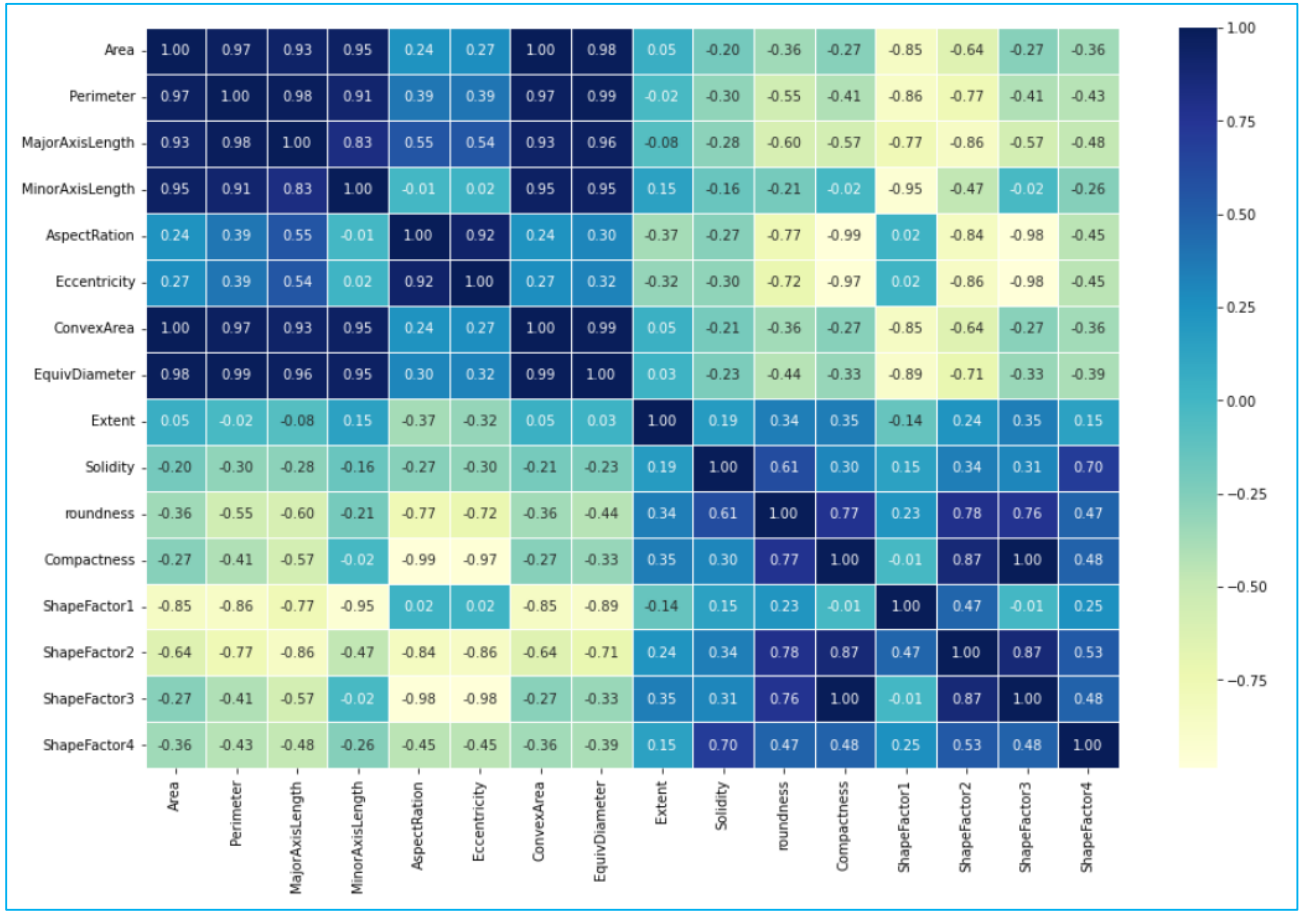
	Class
DERMASON	3546
SIRA	2636
SEKER	2027
HOROZ	1928
CALI	1630
BARBUNYA	1322
BOMBAY	522



EDA-Checking Feature Correlations

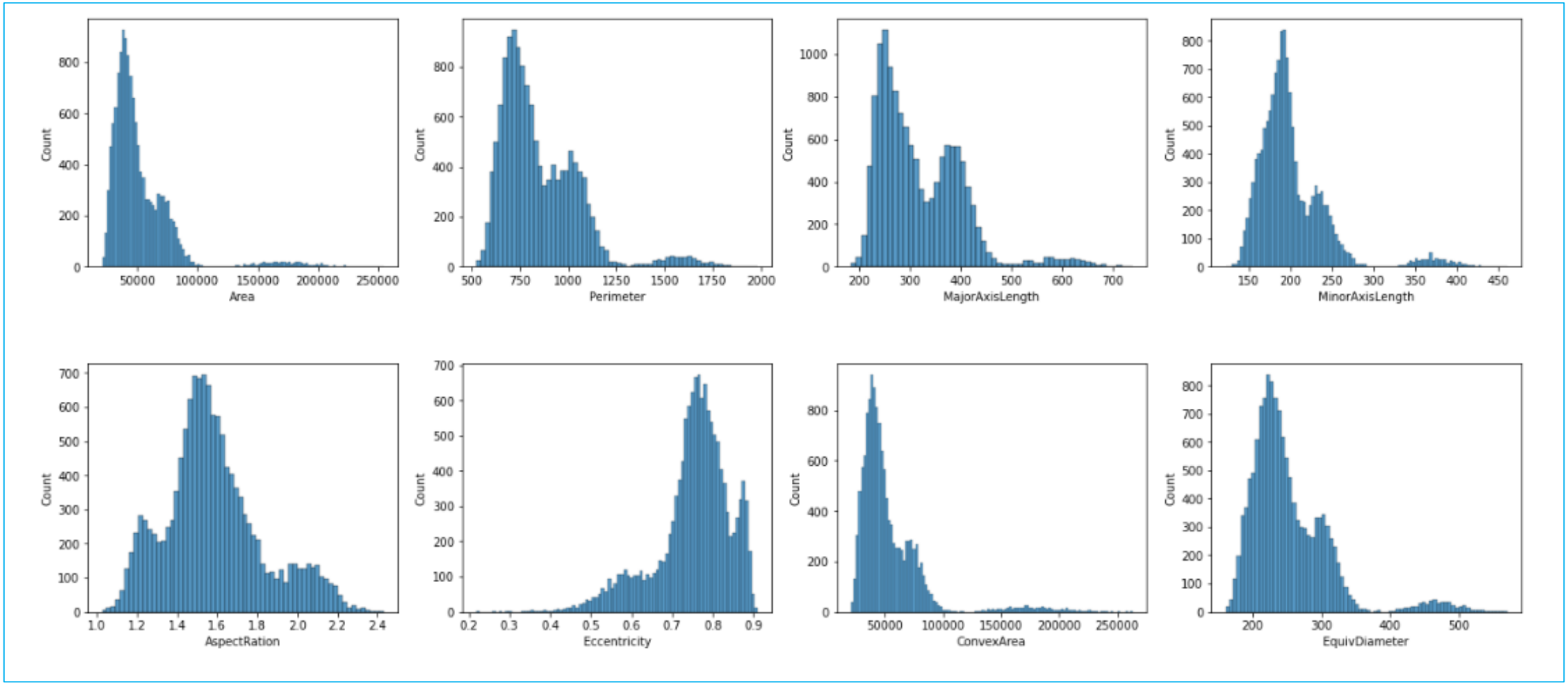
- Some features are strongly correlated to each other, like 'Area-ConvexArea', 'compactness-shape_factor_3', 'Perimeter-EquivDiameter', 'AspectRation-Compactness', 'Eccentricity-ShapeFactor3', 'MajorAxisLength-Perimeter', 'MinorAxisLength-Area', etc.

	Feature_one	Feature_two	correlation
0	Area	ConvexArea	0.999939
1	ConvexArea	Area	0.999939
2	Compactness	ShapeFactor3	0.998686
3	ShapeFactor3	Compactness	0.998686
4	Perimeter	EquivDiameter	0.991380
5	EquivDiameter	Perimeter	0.991380
6	AspectRation	Compactness	0.987687
7	Eccentricity	ShapeFactor3	0.981058
8	MajorAxisLength	Perimeter	0.977338
9	MinorAxisLength	Area	0.951602
10	ShapeFactor1	MinorAxisLength	0.947204
11	ShapeFactor2	ShapeFactor3	0.872971
12	roundness	ShapeFactor2	0.782824
13	Solidity	ShapeFactor4	0.702163
14	ShapeFactor4	Solidity	0.702163
15	Extent	AspectRation	0.370184

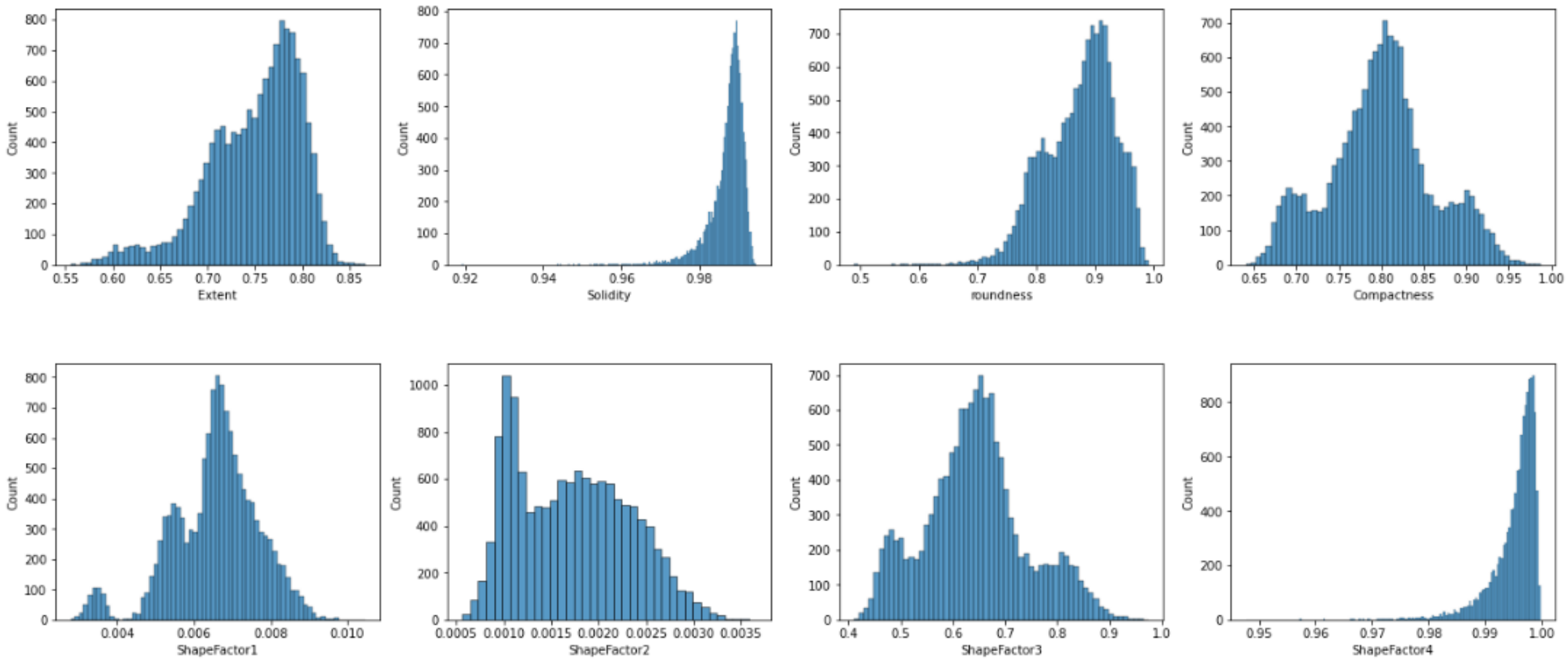


EDA-Features Distributions 1

- The plots show that values in some features have similar distribution (e.g., "area", "perimeter", "major" and "minor" axis length), whereas others are completely different.
- Some of the features are skewed
- Distribution shapes of some features suggest existence of outliers.

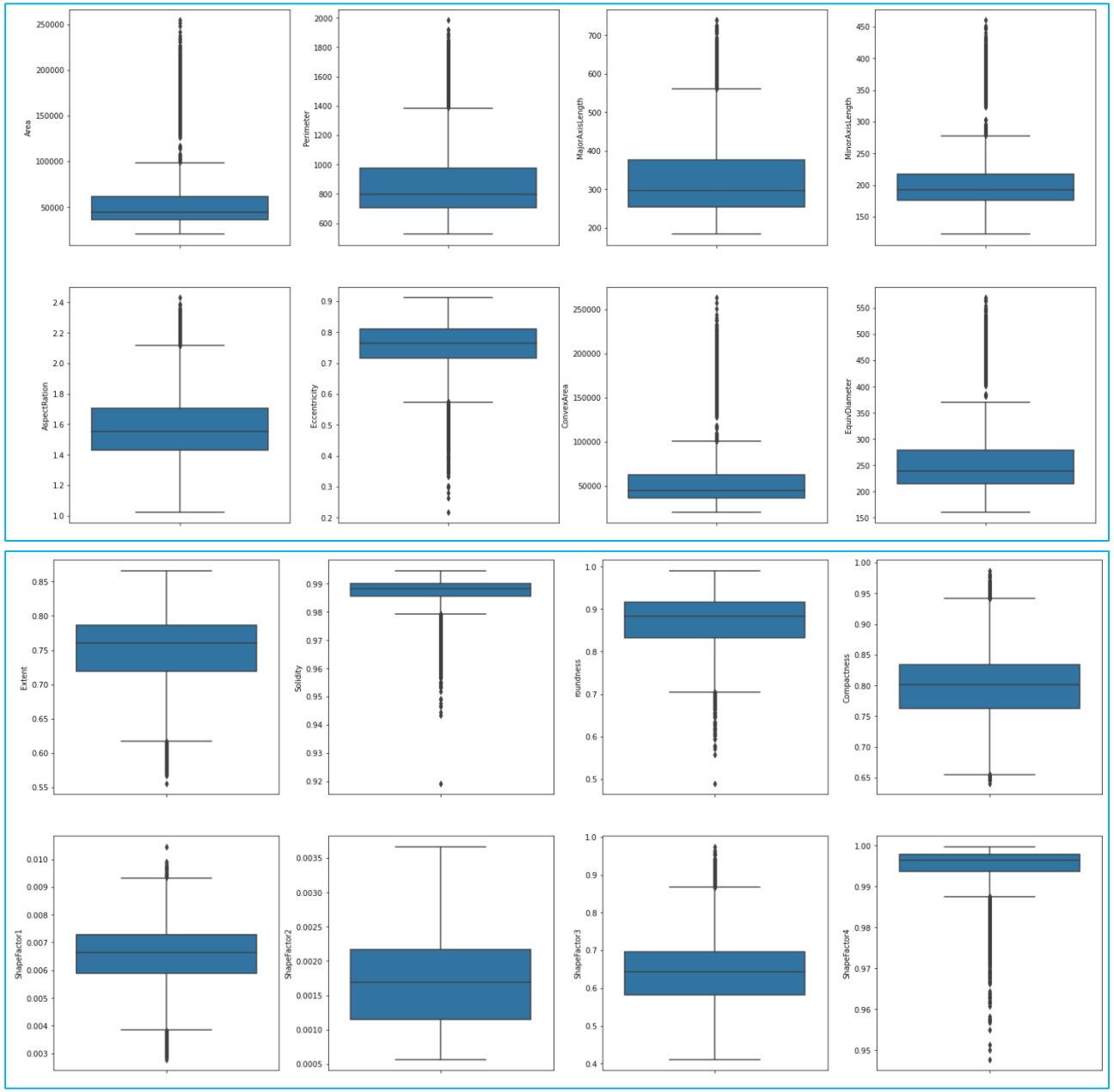


EDA-Features Distributions 2



EDA-Checking for Outliers

There are a lot of outliers in most of the features



EDA-Checking for Skewness

- Log transformation applied to Skewed variables greater than 0.75

Before Log Transformation

	Skewness
Area	2.952931
ConvexArea	2.941821
MinorAxisLength	2.238211
EquivDiameter	1.948958
Perimeter	1.626124
MajorAxisLength	1.357815

Log Transformation

After Log Transformation

	Skewness
MinorAxisLength	1.312460
EquivDiameter	1.074117
Area	1.071399
ConvexArea	1.066074
Perimeter	0.843593

EDA-Feature Scaling

- For models which use distance as metric in its calculations, feature scaling plays vital role.
- All unsupervised models I used in my analysis uses distance metrics, so we need feature scaling before beginning Machine learning section
- Sklearn's StandardScaler applied

```
]: # Lets apply Standard Scaler to bring all the values into same scale.

from sklearn.preprocessing import StandardScaler

float_columns = [x for x in df.columns if x not in ['Class']]

ss = StandardScaler()
df[float_columns] = ss.fit_transform(df[float_columns])
df
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compa
0	10.254004	6.415573	5.343186	5.164150	1.197191	0.549812	10.265210	5.253012	0.763923	0.988856	0.958027	0.
1	10.265871	6.459933	5.305912	5.213491	1.097356	0.411785	10.280999	5.258915	0.783968	0.984986	0.887034	0.
2	10.288103	6.437928	5.365163	5.175761	1.209713	0.562727	10.298599	5.269974	0.778113	0.989559	0.947849	0.
3	10.309253	6.472167	5.354499	5.212305	1.153638	0.498616	10.332832	5.280495	0.782681	0.976696	0.903936	0.
4	10.313642	6.431547	5.312456	5.253735	1.060798	0.333680	10.322790	5.282678	0.773098	0.990893	0.984877	0.
...	...	...	...	...	...	...	...	...	...	...	...	
13606	10.647756	6.634234	5.668921	5.230813	1.552728	0.765002	10.657471	5.448958	0.714574	0.990331	0.916603	0.
13607	10.647851	6.631341	5.643949	5.256000	1.476439	0.735702	10.657142	5.449005	0.799943	0.990752	0.922015	0.
13608	10.648753	6.633741	5.643820	5.258474	1.472582	0.734065	10.658905	5.449455	0.729932	0.989899	0.918424	0.
13609	10.648943	6.639587	5.650321	5.253716	1.489326	0.741055	10.661205	5.449549	0.705389	0.987813	0.907906	0.
13610	10.649227	6.650586	5.690842	5.210604	1.619841	0.786693	10.659633	5.449691	0.788962	0.989648	0.888380	0.

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Comp
0	-1.281157	-1.383199	-1.626115	-0.683706	-1.565053	-2.185720	-1.281561	-1.280395	0.289087	0.367613	1.423867	
1	-1.252264	-1.185410	-1.778667	-0.425194	-1.969784	-3.686040	-1.243234	-1.251545	0.697477	-0.462907	0.231054	2
2	-1.198136	-1.283525	-1.536168	-0.622875	-1.514291	-2.045336	-1.200512	-1.197496	0.578195	0.518417	1.252865	1
3	-1.146644	-1.130860	-1.579813	-0.431411	-1.741618	-2.742211	-1.117416	-1.146077	0.671260	-2.241767	0.515049	2
4	-1.135959	-1.311976	-1.751884	-0.214349	-2.117993	-4.535028	-1.141792	-1.135406	0.476020	0.804772	1.874992	2
...	...	...	...	...	...	...	...	...	...	...	...	
13606	-0.322503	-0.408241	-0.292960	-0.334442	-0.123703	0.153343	-0.329392	-0.322740	-0.716284	0.684173	0.727872	0
13607	-0.322271	-0.421137	-0.395163	-0.202479	-0.432979	-0.165141	-0.330192	-0.322509	1.022933	0.774384	0.818807	0
13608	-0.320075	-0.410439	-0.395691	-0.189518	-0.448618	-0.182940	-0.325912	-0.320314	-0.403392	0.591370	0.758468	0
13609	-0.319613	-0.384373	-0.369085	-0.214446	-0.380735	-0.106960	-0.320330	-0.319852	-0.903414	0.143717	0.581753	0
13610	-0.318920	-0.335332	-0.203243	-0.440321	0.148374	0.389116	-0.324145	-0.319159	0.799227	0.537539	0.253681	-0

13611 rows × 17 columns

3. Clustering Techniques and Findings

Clustering Introduction

- In the following slides, I will show 4 different clustering techniques applied on this dataset and discuss the key findings
  - K-Means Clustering
  - Hierarchical Agglomerative Clustering
  - DBSCAN Algorithm
  - MeanShift Algorithm

Clustering 1: K-Means Algorithm 1

- In order to find the optimal number of cluster, I run the model with clusters ranging from 1 to 20
- Then, for each model, I stored the number of clusters and respective inertia value
- Finally, plotted cluster number vs inertia to find the optimal cluster size

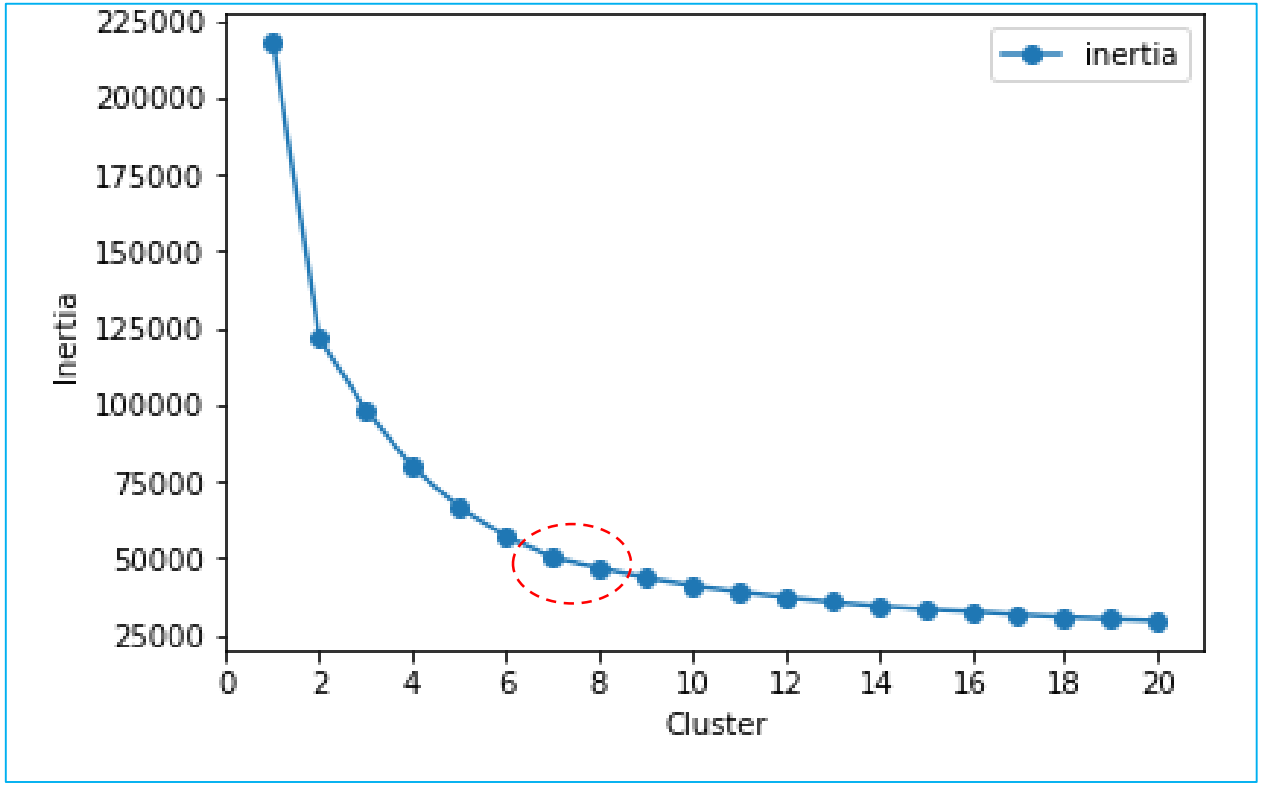
```
from sklearn.cluster import KMeans

# Create and fit a range of models
km_list = list()

for clust in range(1,21):
    km = KMeans(n_clusters=clust, random_state=42)
    km = km.fit(df[float_columns])

    km_list.append(pd.Series({'clusters': clust,
                              'inertia': km.inertia_,
                              'model': km}))
```

Inertia plot shows 7-8 clusters as an optimal size



Clustering 1: K-Means Algorithm 2

- Here, I applied Kmeans algorithm to the dataset with 7 clusters

```
km = KMeans(n_clusters=7, random_state=42)
km = km.fit(df[float_columns])

df['kmeans'] = km.predict(df[float_columns])
```

Predicted Classes

kmeans	Class	number
0	BARBUNYA	1176
	CALI	1321
	HOROZ	30
	SIRA	13
1	DERMASON	2735
	HOROZ	4
	SEKER	14
	SIRA	71
2	BARBUNYA	8
	CALI	24
	DERMASON	5
	HOROZ	1660
	SIRA	83
3	BARBUNYA	40
	BOMBAY	2
	CALI	268
	DERMASON	7
	HOROZ	188
	SEKER	3
	SIRA	25
4	BARBUNYA	1
	BOMBAY	520
	CALI	2
5	BARBUNYA	84
	CALI	13
	DERMASON	685
	HOROZ	46
	SEKER	133
	SIRA	2422
6	BARBUNYA	13
	CALI	2
	DERMASON	114
	SEKER	1877
	SIRA	22

Actual Classes

Class	
DERMASON	3546
SIRA	2636
SEKER	2027
HOROZ	1928
CALI	1630
BARBUNYA	1322
BOMBAY	522

Index	Class Name	Predicted:Actual	Prediction Accuracy (%)
0	Barbunya	1176:1322	88.96
1	Dermason	2735:3546	77.13
2	Horzo	1660:1928	86.10
3	Cali	268:1630	16.44
4	Bombay	520:522	99.17
5	Sira	2422:2636	91.88
6	Seker	1877:2027	92.60

- I can say that, K-Means outcomes were satisfactory for 6 clusters
- Only Cali was predicted poorly-16%, because of similar characteristics with Cluster Barbunya

Clustering 2: Hierarchical Agglomerative Clustering

- Here, I applied Agglomerative clustering algorithm to the dataset with 7 clusters

```
from sklearn.cluster import AgglomerativeClustering

ag = AgglomerativeClustering(n_clusters=7, linkage='ward', compute_full_tree=True)
ag = ag.fit(df[float_columns])
df['agglom'] = ag.fit_predict(df[float_columns])
```

Predicted Classes

agglom	Class	number
0	BARBUNYA	1273
	CALI	1572
	DERMASON	1
	HOROZ	52
	SEKER	7
	SIRA	22
1	BARBUNYA	2
	CALI	32
	DERMASON	1
	HOROZ	1602
	SIRA	33
2	BARBUNYA	32
	CALI	18
	DERMASON	494
	HOROZ	167
	SEKER	60
	SIRA	2252
3	BARBUNYA	6
	CALI	2
	DERMASON	80
	SEKER	1879
	SIRA	57
4	BOMBAY	522
5	BARBUNYA	8
	DERMASON	2969
	HOROZ	12
	SEKER	80
	SIRA	266
6	BARBUNYA	1
	CALI	6
	DERMASON	1
	HOROZ	95
	SEKER	1
	SIRA	6

Actual Classes

Class	
DERMASON	3546
SIRA	2636
SEKER	2027
HOROZ	1928
CALI	1630
BARBUNYA	1322
BOMBAY	522

Index	Class Name	Predicted:Actual	Prediction Accuracy (%)
0	Barbunya	1273:1322	96.30
1	Horzo	1602:1928	83.73
2	Sira	2252:2636	83.09
3	Seker	1879:2027	0.37
4	Bombay	522:522	100
5	Dermason	2969:3546	85.43
6	Cali	6:1630	92.70

- Again, I can say that, outcomes were satisfactory for 6 clusters
- Only Cali was predicted poorly-0.37%, because of similar characteristics with Cluster Barbunya

Clustering 3: DBSCAN Algorithm 1

Minimum Samples ("MinPts")

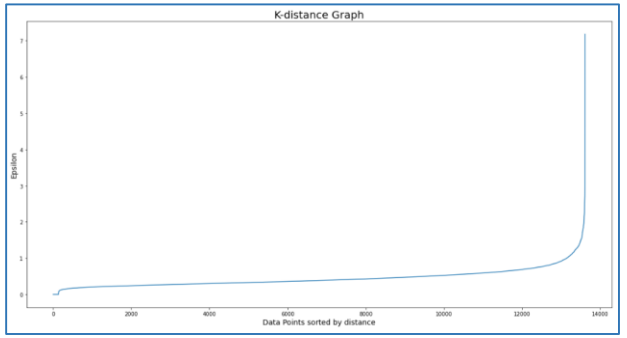
- There is no automatic way to determine the MinPts value for DBSCAN.
- Here are a few rules of thumb for selecting the MinPts value:
  - Generally, MinPts should be greater than or equal to the dimensionality of the data set
  - If your data has more than 2 dimensions, choose MinPts = 2*dim, where dim= the dimensions of your data set (Sander et al., 1998).

In this example, my data contains 16 features, 16 dimensional data then. So, MinSamples=16 is chosen

Epsilon (ε)

- After I selected my MinPts value, I used NearestNeighbors from Scikit-learn, to calculate the average distance between each point and its n_neighbors.
- The one parameter you need to define is n_neighbors, which in this case is the value you choose for MinPts=16.
- Lets find the epsilon now. For epsilon, I am using the K-distance graph. For plotting a K-distance Graph, we need the distance between a point and its nearest data point for all data points in the dataset. We obtain this using NearestNeighbors from sklearn.neighbors.

```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=16)
nbrs = neigh.fit(df[float_columns])
distances, indices = nbrs.kneighbors(df[float_columns])
```



- The optimum value of epsilon is at the point of maximum curvature in the K-Distance Graph, which is 0.65 in this case.

Clustering 3: DBSCAN Algorithm 2

```
from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=0.65, min_samples=16, metric='euclidean')
dbs = dbs.fit(df[float_columns])
np.unique(dbs.labels_)
```

```
array([-1,  0,  1,  2,  3,  4,  5,  6], dtype=int64)
```

```
df['dbscan'] = dbs.fit_predict(df[float_columns])
```

```
(df[['Class','dbscan']].groupby(['Class','dbscan']).size().to_frame().rename(columns={0:'number'}))
```

- As it is seen, DBSCAN gave unrealistic results here, since it predicted most of the clusters as outliers

Class	dbscan	number
BARBUNYA	-1	1280
	0	13
	1	19
	2	10
BOMBAY	-1	522
CALI	-1	1142
	0	7
	2	467
	3	14
DERMASON	-1	588
	0	2958
HOROZ	-1	1417
	0	16
	2	3
	4	167
	5	258
	6	67
SEKER	-1	211
	0	1816
SIRA	-1	496
	0	2139
	6	1

Clustering 4: MeanShift Algorithm

```
|: from sklearn.cluster import MeanShift
   ms  = MeanShift(bandwidth=2.7, n_jobs=-1)
   ms = ms.fit(df[float_columns])
   np.unique(ms.labels_)

|: array([0, 1, 2, 3, 4, 5, 6], dtype=int64)
```

Predicted Classes

MeanShift	Class	number
0	BARBUNYA	973
	CALI	901
	DERMASON	3475
	HOROZ	1865
	SEKER	1882
	SIRA	2634
1	BARBUNYA	340
	BOMBAY	508
	CALI	701
	HOROZ	2
2	BARBUNYA	1
	DERMASON	67
	SEKER	63
	SIRA	2
3	BARBUNYA	1
	BOMBAY	14
	CALI	28
	HOROZ	22
4	BARBUNYA	6
	SEKER	81
5	HOROZ	4
	SEKER	1
6	BARBUNYA	1
	DERMASON	4
	HOROZ	35

Actual Classes

	Class
DERMASON	3546
SIRA	2636
SEKER	2027
HOROZ	1928
CALI	1630
BARBUNYA	1322
BOMBAY	522

- MeanShift algorithm also performed poorly at clustering the Bean classes.

Results and Key Findings

- There are some features which highly correlated to each other that can affect the model performance
- Also, there are quite amount of outliers for most of the features, which also can affect the model performance badly if not excluded
- Some of the features are skewed and log transformed before applying different clustering techniques
- Out of 4 algorithms tested, only 2 of them gave good results: K-Means and Agglomerative Clustering
- Only Class Cali was predicted poorly, because of similar characteristics with Cluster Barbunya
- DBSCAN and MeanShift performed poorly at clustering the Beans into classes
- My Github for Jupyter file: **https://github.com/NARIMANPASHA/Unsupervised-Learning-Clustering-on-Dry-Bean-Dataset.git**

Suggestions

- To predict the class CALI more accurately, there should be “Color” column in the dataset, so to avoid the similarity with class Barbunya.

Class Name	K-Means Accuracy (%)	Agglomerative Accuracy (%)
Barbunya	88.96	96.3
Dermason	77.13	83.73
Horzo	86.10	83.09
Cali	16.44	0.37
Bombay	99.17	100
Sira	91.88	85.43
Seker	92.6	92.70