USL_Faculty_Notebook-Day1

March 11, 2020

1 Faculty_Notebook-Day01

1.0.1 K - Means

K-means algorithm is the most popular and yet simplest of all the clustering algorithms.

- Select the number of clusters k that you think is the optimal number.
- Initialize k points as “centroids” randomly within the space of our data.
- Attribute each observation to its closest centroid.
- Update the centroids to the center of all the attributed set of observations.
- Repeat steps 3 and 4 a fixed number of times or until all of the centroids are stable (i.e. no longer change in step 4).
- This algorithm is easy to describe and visualize. Let’s take a look.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from scipy.stats import zscore
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
df = pd.read_csv("abalone.
↪csv",header=None,names=['Gender','Length','Diameter','Height','Weight','MeatWeight','GutWei
```

[3]:
```
df.head()
```

[3]:
```
  Gender  Length  Diameter  Height  Weight  MeatWeight  GutWeight  \
0      M   0.455     0.365   0.095  0.5140      0.2245     0.1010
1      M   0.350     0.265   0.090  0.2255      0.0995     0.0485
2      F   0.530     0.420   0.135  0.6770      0.2565     0.1415
3      M   0.440     0.365   0.125  0.5160      0.2155     0.1140
4      I   0.330     0.255   0.080  0.2050      0.0895     0.0395

   ShellWeight  Rings
0        0.150     15
1        0.070      7
2        0.210      9
3        0.155     10
4        0.055      7
```

[4]:
```
df['Gender'].value_counts()
```

[4]:
```
M    1528
I    1342
F    1307
Name: Gender, dtype: int64
```

[5]:
```
df_rows , df_cols = df.shape
print(df_rows)
print(df_cols)
```

```
4177
9
```

[6]:
```
df2 = df.loc[:, 'Length':'Rings']
```

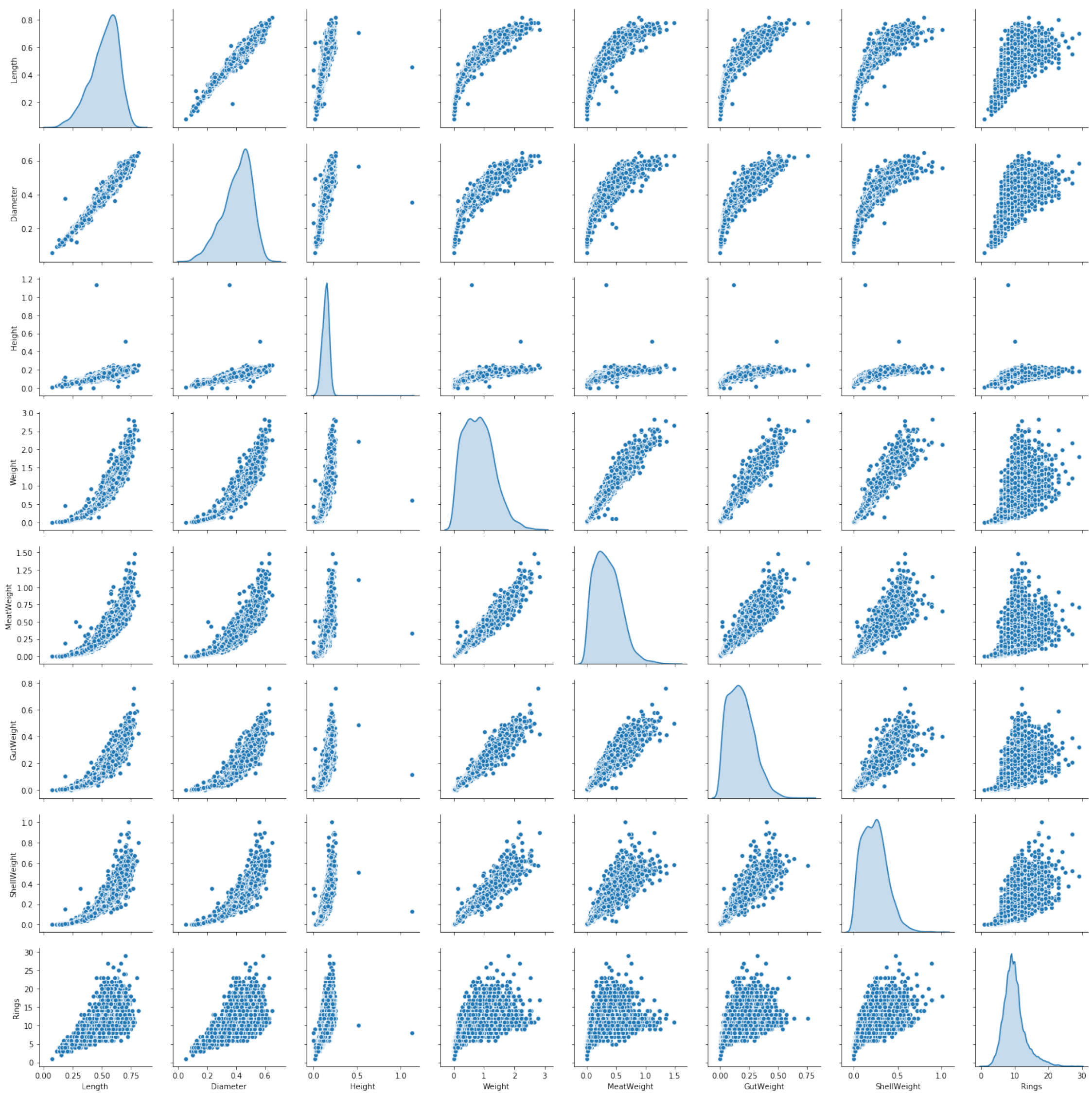
[7]:
```
df2.head()
```

[7]:

	Length	Diameter	Height	Weight	MeatWeight	GutWeight	ShellWeight	Rings
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

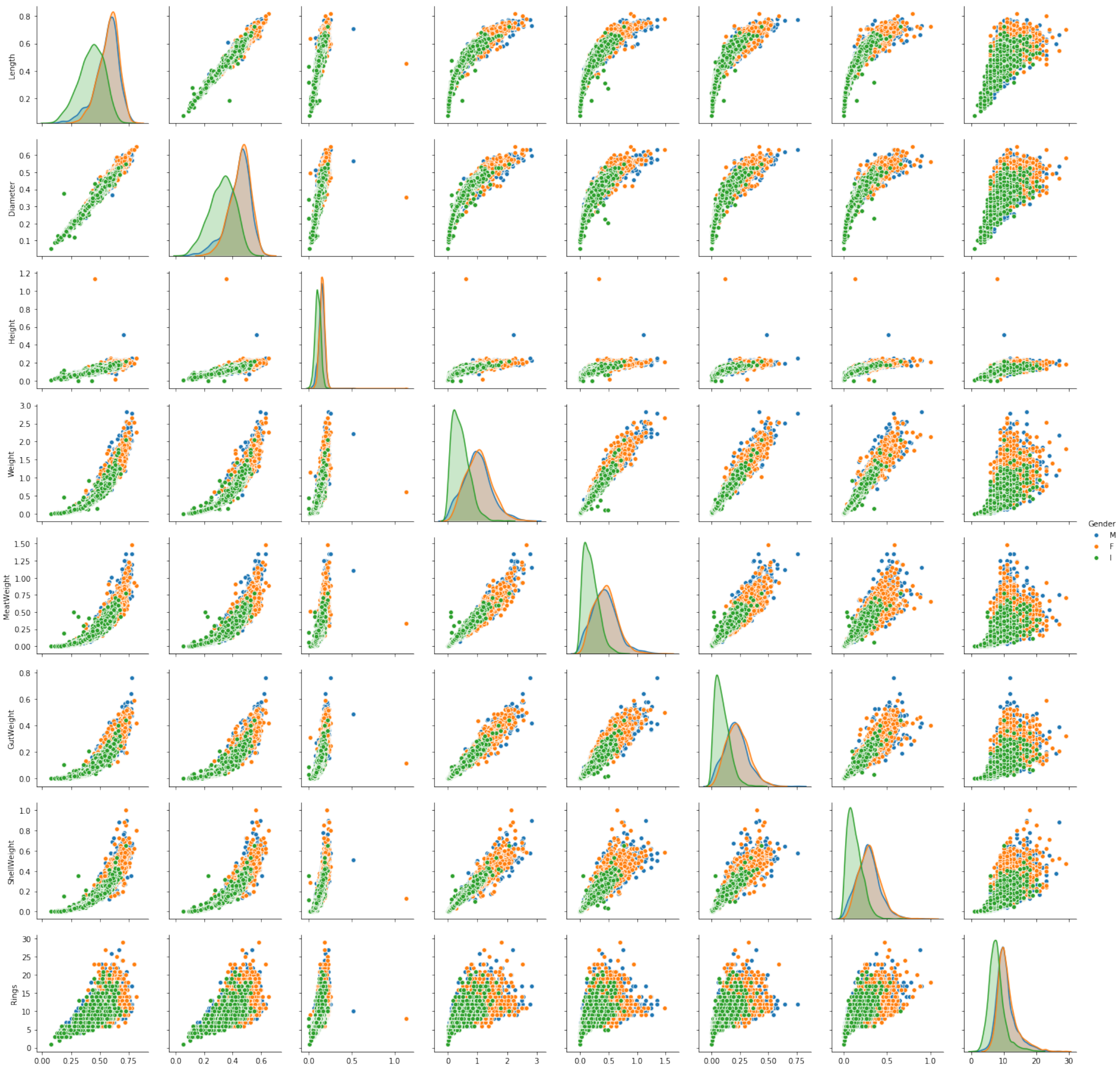
[8]:
```
import seaborn as sns
sns.pairplot(df2,diag_kind='kde')
```

[8]:
```
<seaborn.axisgrid.PairGrid at 0x1f8f0e9e4c8>
```




```
[9]: sns.pairplot(df,diag_kind='kde', hue='Gender')
```

```
[9]: <seaborn.axisgrid.PairGrid at 0x1f8f41c4a88>
```



[11]:
```
df_scaled = df2.apply(zscore)
```

[12]:
```
df_scaled.head()
```

[12]:

	Length	Diameter	Height	Weight	MeatWeight	GutWeight	ShellWeight	\
0	-0.574558	-0.432149	-1.064424	-0.641898	-0.607685	-0.726212	-0.638217	
1	-1.448986	-1.439929	-1.183978	-1.230277	-1.170910	-1.205221	-1.212987	
2	0.050033	0.122130	-0.107991	-0.309469	-0.463500	-0.356690	-0.207139	
3	-0.699476	-0.432149	-0.347099	-0.637819	-0.648238	-0.607600	-0.602294	
4	-1.615544	-1.540707	-1.423087	-1.272086	-1.215968	-1.287337	-1.320757	

	Rings
0	1.571544
1	-0.910013

```
2 -0.289624
3  0.020571
4 -0.910013
```

[0]:
```python
model = KMeans(n_clusters = 3)
```

[0]:
```python
model
```

[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

[13]:
```python
cluster_range = range( 1, 15 )
cluster_errors = []
for num_clusters in cluster_range:
  clusters = KMeans( num_clusters, n_init = 10 )
  clusters.fit(df_scaled)
 # labels = clusters.labels_
 # centroids = clusters.cluster_centers_
  cluster_errors.append( clusters.inertia_ )
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors":␣
 ↪cluster_errors } )
clusters_df[0:15]
```

[13]:

	num_clusters	cluster_errors
0	1	33416.000000
1	2	14612.656454
2	3	9922.798048
3	4	7867.534463
4	5	6799.301810
5	6	5837.314584
6	7	5282.154783
7	8	4752.474837
8	9	4379.562577
9	10	3928.794722
10	11	3650.603366
11	12	3429.190887
12	13	3263.715194
13	14	3141.955177

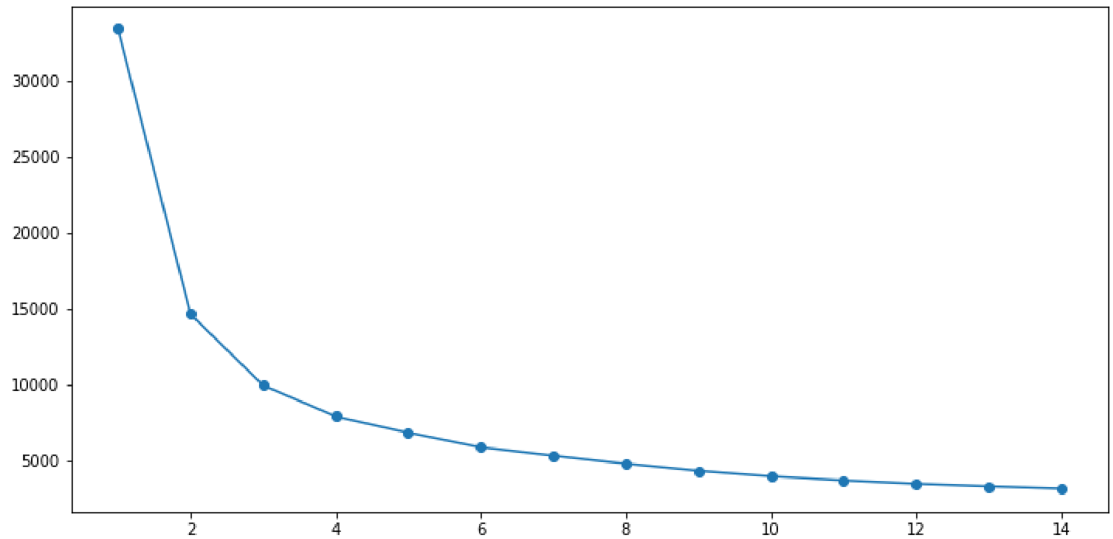
The total sum of squared distances of every data point from respective centroid is also called inertia. Let us print the inertia value for all K values. That K at which the inertia stop to drop significantly (elbow method) will be the best K.

[0]:
```python
# Elbow plot

plt.figure(figsize=(12,6))
```

```
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

[0]: [<matplotlib.lines.Line2D at 0x7f5d50d691d0>]



[0]:
```
kmeans = KMeans(n_clusters=3, n_init = 15, random_state=2345)
```

[0]:
```
kmeans.fit(df_scaled)
```

[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=15, n_jobs=1, precompute_distances='auto',
    random_state=2345, tol=0.0001, verbose=0)
```

[0]:
```
centroids = kmeans.cluster_centers_
```

[0]:
```
centroids
```

[0]:
```
array([[-1.27709975, -1.27787282, -1.12085776, -1.13750326, -1.08408354,
        -1.11591049, -1.1283335 , -0.85717476],
       [ 1.03608988,  1.04187408,  0.98130211,  1.22176997,  1.17823769,
         1.19808871,  1.17676949,  0.62065252],
       [ 0.13570472,  0.13223651,  0.06955548, -0.08492905, -0.09047204,
        -0.0829803 , -0.06004439,  0.14255146]])
```

[0]:
```
centroid_df = pd.DataFrame(centroids, columns = list(df_scaled) )
```

[0]:
```
centroid_df
```

```
[0]:      Length  Diameter    Height    Weight  MeatWeight  GutWeight  ShellWeight  \
     0 -1.277100 -1.277873 -1.120858 -1.137503   -1.084084  -1.115910    -1.128333
     1  1.036090  1.041874  0.981302  1.221770    1.178238   1.198089     1.176769
     2  0.135705  0.132237  0.069555 -0.084929   -0.090472  -0.082980    -0.060044

          Rings
     0 -0.857175
     1  0.620653
     2  0.142551
```

kmeans.labels_ pd.DataFrame(kmeans.labels_, columns = "label")

```
[0]: ## creating a new dataframe only for labels and converting it into categorical␣
      ↪variable
     df_labels = pd.DataFrame(kmeans.labels_ , columns = list(['labels']))

     df_labels['labels'] = df_labels['labels'].astype('category')
```

```
[0]: # Joining the label dataframe with the Wine data frame to create␣
      ↪wine_df_labeled. Note: it could be appended to original dataframe
     snail_df_labeled = df2.join(df_labels)
```

```
[0]: df_analysis = (snail_df_labeled.groupby(['labels'] , axis=0)).head(4177)  # the␣
      ↪groupby creates a groupeddataframe that needs
     # to be converted back to dataframe. I am using .head(30000) for that
     df_analysis
```

```
[0]:       Length  Diameter  Height  Weight  MeatWeight  GutWeight  ShellWeight  \
     0      0.455     0.365   0.095  0.5140      0.2245     0.1010       0.1500
     1      0.350     0.265   0.090  0.2255      0.0995     0.0485       0.0700
     2      0.530     0.420   0.135  0.6770      0.2565     0.1415       0.2100
     3      0.440     0.365   0.125  0.5160      0.2155     0.1140       0.1550
     4      0.330     0.255   0.080  0.2050      0.0895     0.0395       0.0550
     5      0.425     0.300   0.095  0.3515      0.1410     0.0775       0.1200
     6      0.530     0.415   0.150  0.7775      0.2370     0.1415       0.3300
     7      0.545     0.425   0.125  0.7680      0.2940     0.1495       0.2600
     8      0.475     0.370   0.125  0.5095      0.2165     0.1125       0.1650
     9      0.550     0.440   0.150  0.8945      0.3145     0.1510       0.3200
     10     0.525     0.380   0.140  0.6065      0.1940     0.1475       0.2100
     11     0.430     0.350   0.110  0.4060      0.1675     0.0810       0.1350
     12     0.490     0.380   0.135  0.5415      0.2175     0.0950       0.1900
     13     0.535     0.405   0.145  0.6845      0.2725     0.1710       0.2050
     14     0.470     0.355   0.100  0.4755      0.1675     0.0805       0.1850
     15     0.500     0.400   0.130  0.6645      0.2580     0.1330       0.2400
     16     0.355     0.280   0.085  0.2905      0.0950     0.0395       0.1150
     17     0.440     0.340   0.100  0.4510      0.1880     0.0870       0.1300
     18     0.365     0.295   0.080  0.2555      0.0970     0.0430       0.1000
```

19	0.450	0.320	0.100	0.3810	0.1705	0.0750	0.1150
20	0.355	0.280	0.095	0.2455	0.0955	0.0620	0.0750
21	0.380	0.275	0.100	0.2255	0.0800	0.0490	0.0850
22	0.565	0.440	0.155	0.9395	0.4275	0.2140	0.2700
23	0.550	0.415	0.135	0.7635	0.3180	0.2100	0.2000
24	0.615	0.480	0.165	1.1615	0.5130	0.3010	0.3050
25	0.560	0.440	0.140	0.9285	0.3825	0.1880	0.3000
26	0.580	0.450	0.185	0.9955	0.3945	0.2720	0.2850
27	0.590	0.445	0.140	0.9310	0.3560	0.2340	0.2800
28	0.605	0.475	0.180	0.9365	0.3940	0.2190	0.2950
29	0.575	0.425	0.140	0.8635	0.3930	0.2270	0.2000
…	…	… …	…	…	…	…	…
4147	0.695	0.550	0.195	1.6645	0.7270	0.3600	0.4450
4148	0.770	0.605	0.175	2.0505	0.8005	0.5260	0.3550
4149	0.280	0.215	0.070	0.1240	0.0630	0.0215	0.0300
4150	0.330	0.230	0.080	0.1400	0.0565	0.0365	0.0460
4151	0.350	0.250	0.075	0.1695	0.0835	0.0355	0.0410
4152	0.370	0.280	0.090	0.2180	0.0995	0.0545	0.0615
4153	0.430	0.315	0.115	0.3840	0.1885	0.0715	0.1100
4154	0.435	0.330	0.095	0.3930	0.2190	0.0750	0.0885
4155	0.440	0.350	0.110	0.3805	0.1575	0.0895	0.1150
4156	0.475	0.370	0.110	0.4895	0.2185	0.1070	0.1460
4157	0.475	0.360	0.140	0.5135	0.2410	0.1045	0.1550
4158	0.480	0.355	0.110	0.4495	0.2010	0.0890	0.1400
4159	0.560	0.440	0.135	0.8025	0.3500	0.1615	0.2590
4160	0.585	0.475	0.165	1.0530	0.4580	0.2170	0.3000
4161	0.585	0.455	0.170	0.9945	0.4255	0.2630	0.2845
4162	0.385	0.255	0.100	0.3175	0.1370	0.0680	0.0920
4163	0.390	0.310	0.085	0.3440	0.1810	0.0695	0.0790
4164	0.390	0.290	0.100	0.2845	0.1255	0.0635	0.0810
4165	0.405	0.300	0.085	0.3035	0.1500	0.0505	0.0880
4166	0.475	0.365	0.115	0.4990	0.2320	0.0885	0.1560
4167	0.500	0.380	0.125	0.5770	0.2690	0.1265	0.1535
4168	0.515	0.400	0.125	0.6150	0.2865	0.1230	0.1765
4169	0.520	0.385	0.165	0.7910	0.3750	0.1800	0.1815
4170	0.550	0.430	0.130	0.8395	0.3155	0.1955	0.2405
4171	0.560	0.430	0.155	0.8675	0.4000	0.1720	0.2290
4172	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490
4173	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605
4174	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080
4175	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960
4176	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950

	Rings	labels
0	15	2
1	7	0
2	9	2

3	10	2
4	7	0
5	8	0
6	20	2
7	16	2
8	9	2
9	19	2
10	14	2
11	10	0
12	11	2
13	10	2
14	10	0
15	12	2
16	7	0
17	10	0
18	7	0
19	9	0
20	11	0
21	10	0
22	12	2
23	9	2
24	10	1
25	11	2
26	11	2
27	12	2
28	15	2
29	11	2
…	…	…
4147	11	1
4148	11	1
4149	6	0
4150	7	0
4151	6	0
4152	7	0
4153	8	0
4154	6	0
4155	6	0
4156	8	0
4157	8	2
4158	8	0
4159	9	2
4160	11	2
4161	11	2
4162	8	0
4163	7	0
4164	7	0
4165	7	0

```
4166      10       2
4167       9       2
4168       8       2
4169      10       2
4170      10       2
4171       8       2
4172      11       2
4173      10       2
4174       9       1
4175      10       1
4176      12       1

[4177 rows x 9 columns]
```

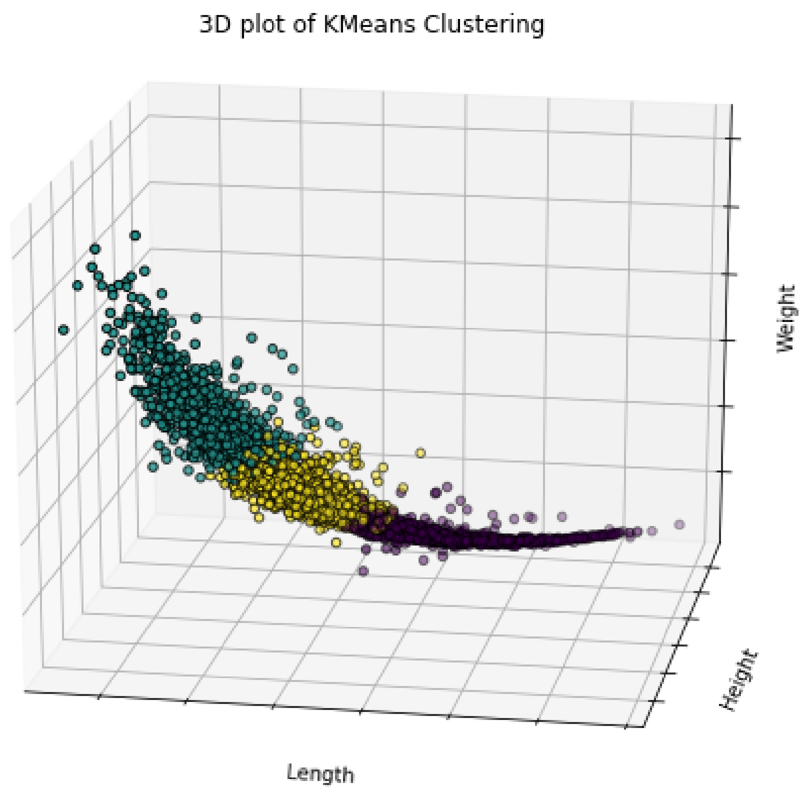
[0]:
```
snail_df_labeled['labels'].value_counts()   #0-Infant, 1-Female, 2-Male
```

[0]:
```
2    1773
1    1224
0    1180
Name: labels, dtype: int64
```

[0]:
```
from mpl_toolkits.mplot3d import Axes3D
```

[0]:
```
fig = plt.figure(figsize=(8, 6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=20, azim=100)
kmeans.fit(df_scaled)
labels = kmeans.labels_
ax.scatter(df_scaled.iloc[:, 0], df_scaled.iloc[:, 1], df_scaled.iloc[:,␣
 ↪3],c=labels.astype(np.float), edgecolor='k')
ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel('Length')
ax.set_ylabel('Height')
ax.set_zlabel('Weight')
ax.set_title('3D plot of KMeans Clustering')
```

[0]:
```
Text(0.5,0.92,'3D plot of KMeans Clustering')
```


3D plot of KMeans Clustering
Weight
Height
Length