Project2_Spring2021

April 27, 2021

1 Project 2

1.1 GENERAL INSTRUCTIONS:

this is NOT a group project - CLEARLY mark where you are answering each question (all written questions must be answered in Markdown cells, NOT as comments in code cells) - Show all code necessary for the analysis, but remove superfluous code

---

2 DONUTS

Using the dataset *krispykreme.csv*,

- **a)** make 3 scatterplots using ggplot to show:
  - `Sodium_100g` vs `Total_Fat_100g`
  - `Sodium_100g` vs. `Sugar_100g`
  - `Sugar_100g` vs `Total_Fat_100g`
- **b)** Using the scatterplots from part **a** as well as the donuts dataset, **thouroughly discuss which clustering method** (KMeans, Gaussian Mixture Models (EM), Hierarchical Clustering, or DBSCAN) **you think would be best for this data and WHY**. Be sure to include discussions of assumptions each algorithm does/does not make, and what types of data they are good/bad for (**mention each of the 4 algorithms at least once**). (*IN A MARKDOWN CELL*)

Please note that for this assignment, "It's easier to code" or "it's comuptationally efficient" does not count as a valid reason. The reasons should be based on the algorithms/data.

(Please use "**" to make any mention of one of the algorithms bold in your discussion. For example "I think **DBSCAN** is the best algorithm ever!" will make the word "DBSCAN" bold in a Markdown cell).

- **c) Implement the algorithm** you think will work best here using the 3 variables `Sodium_100g`, `Total_Fat_100g` and `Sugar_100g`, and describe **how you chose any hyperparameters** (such as distance, # of clusters, min_samples, eps, linkage…etc). Make sure to z-score your variables. (*IN A MARKDOWN CELL*)
- **d) Thouroughly discuss the performance** of your clustering model.
  - which metric did you use to asses your model? (*IN A MARKDOWN CELL*)
  - how did your model perform? (*IN A MARKDOWN CELL*)

  - remake the 3 graphs from part a, but color by cluster assignment. Describe what characterizes each cluster, and give an example of a label for that cluster (e.g. "these donuts are low fat, and low sugar so I would call these healthy donuts") (*IN A MARKDOWN CELL*)

- **e)** Choose ONE other of the `_100g` variables from the data set to **add to your clustering model** to improve it.
  - explain why you chose this variable. (*IN A MARKDOWN CELL*)
  - make a new model, identical to the model in part c, but also including your new variable.
  - did this variable improve the fit of your clustering model? How can you tell? (*IN A MARKDOWN CELL*)

Note: The columns with `_100g` at the end represent the amount of that nutrient per 100 grams of the food. For example, `Total_Fat` tells you the total amount of fat in that food, whereas `Total_Fat_100g` tells you how much fat there is per 100 grams of that food.

[1]:
```
# import necessary packages
# import necessary packages

import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
from plotnine import *

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors

from sklearn.cluster import DBSCAN


from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette_score

from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch

%matplotlib inline
```

[2]:
```
data = pd.read_csv('https://raw.githubusercontent.com/cmparlettpelleriti/
↪CPSC392ParlettPelleriti/master/Data/KrispyKreme.csv')
```

[4]:
```
data.describe()
```

[4]:

	Restaurant_ID	Serving_Size	Serving_Size_text	Calories	Total_Fat	\
count	205.0	205.000000	0.0	205.000000	205.000000	
mean	49.0	34.375610	NaN	257.170732	7.204878	
std	0.0	33.837854	NaN	98.567162	5.938366	
min	49.0	12.000000	NaN	60.000000	0.000000	
25%	49.0	16.000000	NaN	190.000000	2.000000	
50%	49.0	20.000000	NaN	250.000000	6.000000	
75%	49.0	54.000000	NaN	330.000000	11.000000	
max	49.0	199.000000	NaN	500.000000	24.000000	

	Saturated_Fat	Trans_Fat	Cholesterol	Sodium	Potassium	…	\
count	205.000000	205.000000	205.000000	205.000000	39.000000	…	
mean	3.900000	0.002439	14.414634	149.317073	48.461538	…	
std	2.959034	0.034922	12.461658	73.091884	24.712927	…	
min	0.000000	0.000000	0.000000	10.000000	15.000000	…	
25%	1.000000	0.000000	5.000000	105.000000	35.000000	…	
50%	4.000000	0.000000	15.000000	140.000000	50.000000	…	
75%	6.000000	0.000000	25.000000	180.000000	50.000000	…	
max	10.000000	0.500000	45.000000	390.000000	125.000000	…	

	Total_Fat_100g	Saturated_Fat_100g	Trans_Fat_100g	Cholesterol_100g	\
count	205.000000	205.000000	205.000000	205.000000	
mean	6.102439	3.000000	0.004878	5.034146	
std	8.641796	3.852679	0.069843	7.799812	
min	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	0.000000	1.000000	
50%	1.000000	1.000000	0.000000	4.000000	
75%	14.000000	6.000000	0.000000	6.000000	
max	30.000000	13.000000	1.000000	44.000000	

	Sodium_100g	Potassium_100g	Carbohydrates_100g	Protein_100g	\
count	205.000000	39.000000	205.000000	205.000000	
mean	84.292683	67.461538	20.278049	2.834146	
std	104.113636	37.132654	17.794738	1.657368	
min	2.000000	29.000000	3.000000	0.000000	
25%	28.000000	50.000000	9.000000	2.000000	
50%	35.000000	57.000000	11.000000	2.000000	
75%	121.000000	66.000000	39.000000	4.000000	
max	557.000000	176.000000	57.000000	10.000000	

	Sugar_100g	Dietary_Fiber_100g
count	205.000000	174.00000
mean	13.507317	0.41954
std	9.249070	0.73072
min	0.000000	0.00000
25%	8.000000	0.00000
50%	11.000000	0.00000

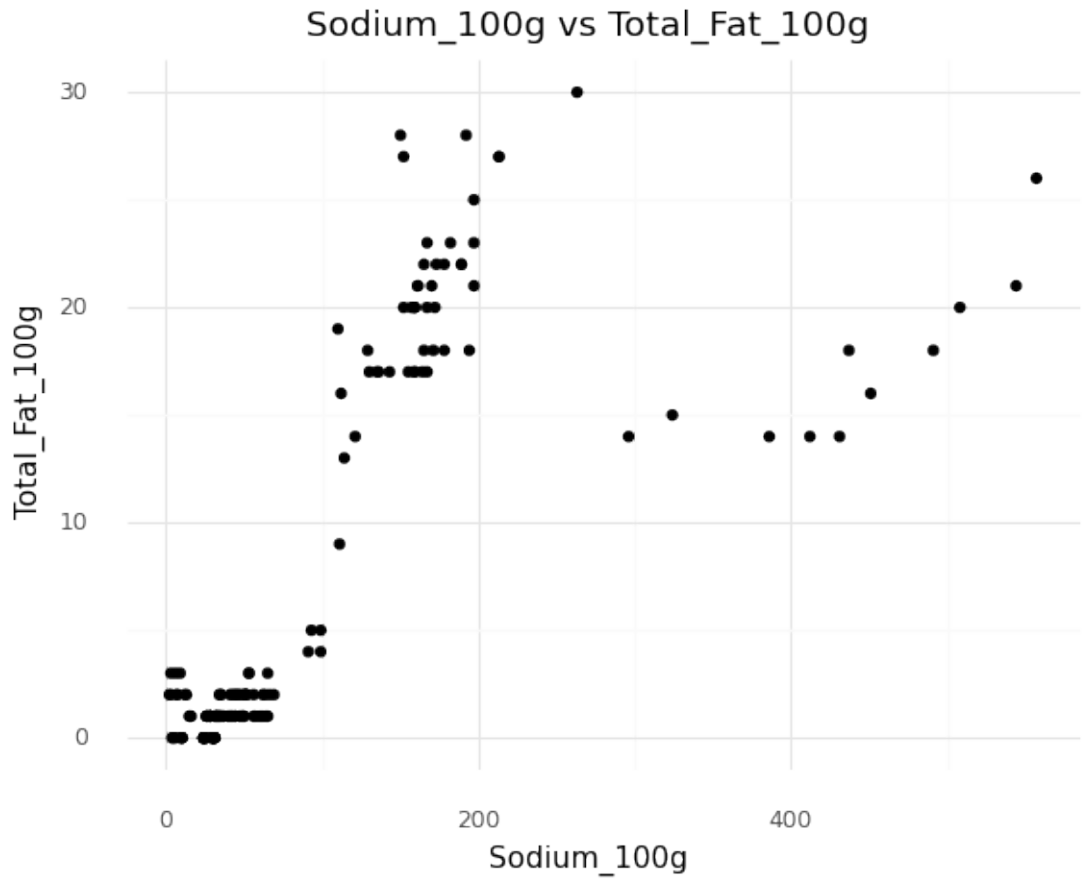
```
75%     18.000000              1.00000
max     39.000000              4.00000

[8 rows x 25 columns]
```

[5]:
```
data.isnull().sum(axis=0) # checked to make sure there is no missing data
```

[5]:
```
Restaurant_Item_Name        0
restaurant                  0
Restaurant_ID               0
Item_Name                   0
Item_Description            0
Food_Category               0
Serving_Size                0
Serving_Size_text         205
Serving_Size_Unit           0
Serving_Size_household    198
Calories                    0
Total_Fat                   0
Saturated_Fat               0
Trans_Fat                   0
Cholesterol                 0
Sodium                      0
Potassium                 166
Carbohydrates               0
Protein                     0
Sugar                       0
Dietary_Fiber              31
Calories_100g               0
Total_Fat_100g              0
Saturated_Fat_100g          0
Trans_Fat_100g              0
Cholesterol_100g            0
Sodium_100g                 0
Potassium_100g            166
Carbohydrates_100g          0
Protein_100g                0
Sugar_100g                  0
Dietary_Fiber_100g         31
dtype: int64
```

3 A)

[15]:
```
# A)
# Sodium_100g vs Total_Fat_100g
(ggplot(data, aes(x = "Sodium_100g", y = "Total_Fat_100g")) + geom_point() +␣
 ↪theme_minimal() + ggtitle("Sodium_100g vs Total_Fat_100g") + labs(x =␣
 ↪"Sodium_100g", y = "Total_Fat_100g"))
```



[15]: <ggplot: (312629736)>

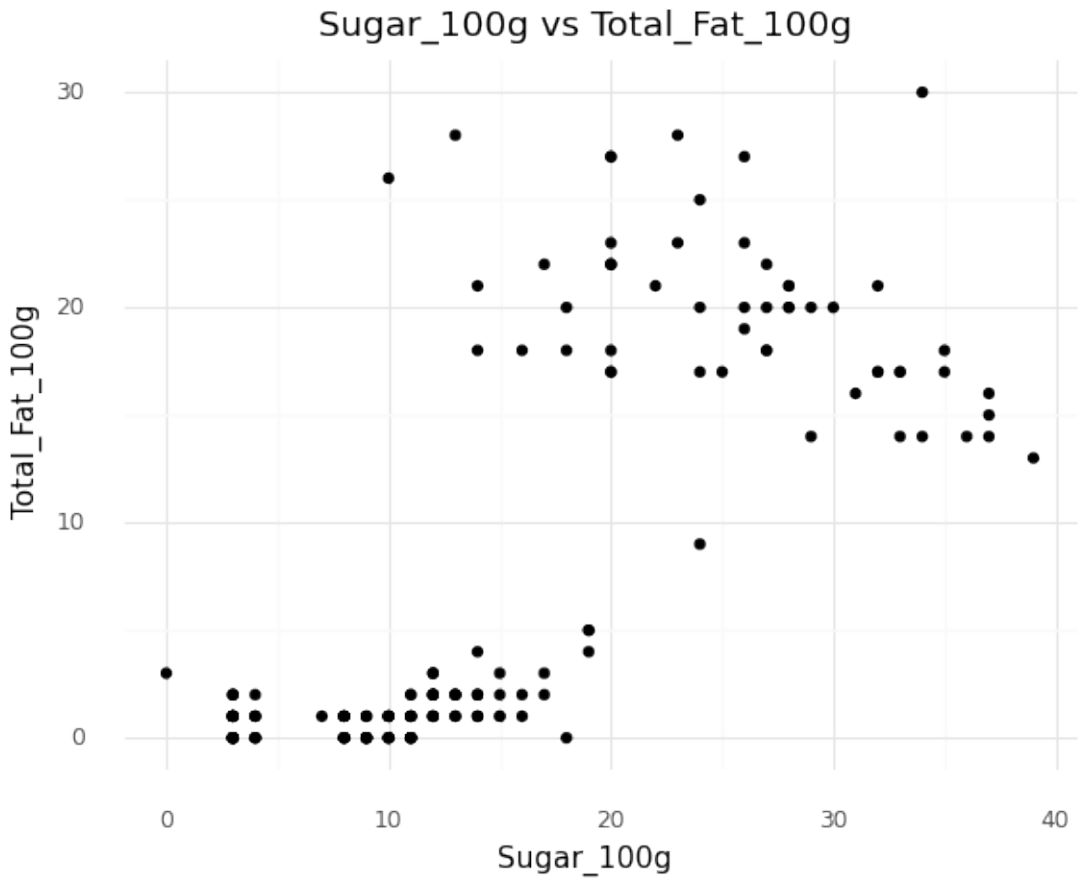
3.1 A cont)

- **Description of Sodium_100g vs Total_Fat_100g Graph:**
  - The distribution of the data points on this graph suggest that there are 2 or 3 clusters. The first cluster (very left) appears to be in the bottom left of the graph. The region (spherical shape) is roughly: Sodium [0, 100] & Total Fat [0, 5]. This is the most dense cluster and has a lot of overlap within the cluster. The next cluster appears to be in the middle of the graph in the region (elliptical shape) roughly of: Sodium [105, 210] & Total Fat [9, 30.5]. This cluster is the second most dense of the clusters and also has

overlap of points within itself. Lastly, the third cluster is in the region (elliptical shape ~ non-dense) of about: Sodium [240, 575] & Total Fat [12, 26]. However, this cluster is the least dense and most likely consists of outliers.

- The overlaps that are observed are often in concentrated areas, which may suggest there is a hierarchical relationship in data. For example, in the first cluster there is a high concentration of overlap in different segments of y-values. There appears to be concentrated segments at: Fat = 3, Sodium [40, 60] and Fat = 4, Sodium [45, 55], and so on.
- Shapes Observed: [sphere, ellipse]
- Densities moderately vary
- Has overlap within clusters
- Possibly has sub-groups/sub-clusters

[7]:
```
# A cont)
# Sugar_100g vs Total_Fat_100g
(ggplot(data, aes(x = "Sugar_100g", y = "Total_Fat_100g")) + geom_point() +␣
↪theme_minimal() + ggtitle("Sugar_100g vs Total_Fat_100g") + labs(x =␣
↪"Sugar_100g", y = "Total_Fat_100g"))
```

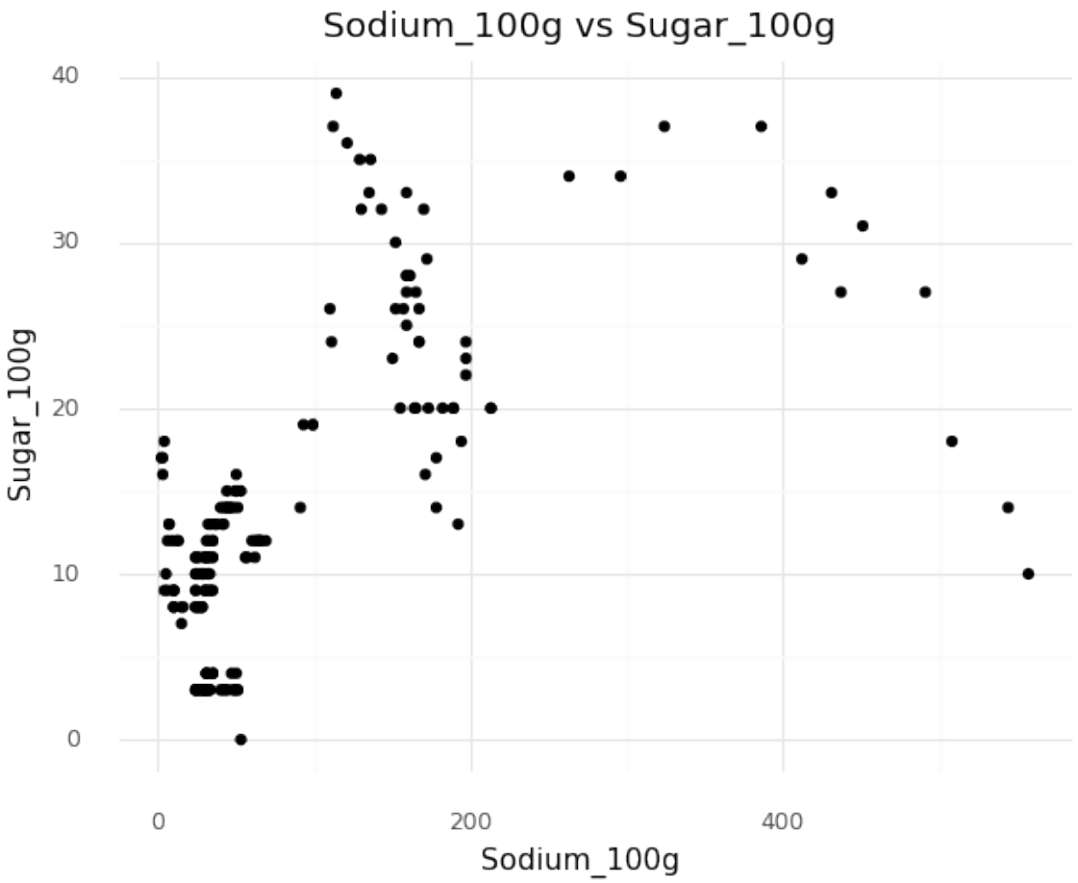


[7]: <ggplot: (312602950)>

3.2 A cont)

- **Description of Sugar_100g vs Total_Fat_100g Graph:**
  - The distribution of the data points on this graph suggest that there are 2 clusters. The first cluster (left one) appears to be in the region (elliptical shaped) of: Sugar [0, 19] & Total Fat [0, 5.5]. This cluster is the more denser one of the two clusters and there is no overlap observed. The second cluster (right side) appears to be in the region (elliptical shaped) of: Sugar [10, 39] & Total Fat [13, 30]. The second cluster is less dense compared to the first cluster. It also does not have overlap of data points within itself.
  - There does not seem to be many sub-groups within these clusters except for left cluster. There is a group of points in the region of Sugar [3, 4] & Total Fat [0, 3] and another group of points observed in the region of Sugar [7, 17] & Total Fat [0, 10]. These group of clusters suggest a possible hierarchical relationship exists within the left cluster.
  - Shapes Observed: [ellipse]
  - Densities greatly vary
  - Does not have overlap within clusters
  - Possibly has sub-groups/sub-clusters

[8]:
```
# A cont)
# Sodium_100g vs Sugar_100g
(ggplot(data, aes(x = "Sodium_100g", y = "Sugar_100g")) + geom_point() +␣
 ↪theme_minimal() + ggtitle("Sodium_100g vs Sugar_100g") + labs(x =␣
 ↪"Sodium_100g", y = "Sugar_100g"))
```



[8]: <ggplot: (312629212)>

3.3 A cont)

- **Description of Sodium_100g vs Sugar_100g Graph:**
  - The distribution of the of the data points on this graph suggest that are 2 or 3 clusters. The first cluster (very left) appears to be in the bottom left of the graph. The region (elliptical shape) is roughly: Sodium [0, 75] & Sugar [0, 18]. This first cluster is the most dense of the three and it has a lot of overlapping points within itself. The second cluster (middle cluster) is roughly in the region (elliptical shape) of: Sodium [110, 220] & Sugar [13, 39]. This cluster is the second most dense of the three and also has some overlap of its datapoints. Lastly, the third cluster (most right) is roughly in the region (elliptical shape) of: Sodium [260, 560] & Sugar [10, 36]. This last cluster is a lot less dense compared to the other clusters and so it should most likely not be considered a cluster.
  - Similar to the previous two graphs, there seems to be potential sub-groups observed in the left cluster. For example, there seems to be a group in the region of Sodium [30, 50] & Sugar [3, 4.5] and another group in the region of Sodium [20, 40] & Sugar [8, 16.5], and so on. These group of clusters suggest a possible hierarchical relationship exists

within the left cluster.
  - Shapes Observed: [ellipse]
  - Densities moderately vary
  - Has overlap within clusters
  - Possibly has sub-groups/sub-clusters

4 B)

4.0.1 Question)

- Using the scatterplots from part a as well as the donuts dataset, thouroughly discuss which clustering method (KMeans, Gaussian Mixture Models (EM), Hierarchical Clustering, or DBSCAN) you think would be best for this data and WHY. Be sure to include discussions of assumptions each algorithm does/does not make, and what types of data they are good/bad for (mention each of the 4 algorithms at least once). (IN A MARKDOWN CELL)

4.0.2 Answer

- **Gaussian Mixture Models (EM)** is the best choice algorithm to use with this dataset (explanation below).

4.0.3 Clustering Models Discussion

- **KMeans**
  - KMeans is a clustering algorithm that takes an iterative approach to clustering data.
- The algorithm can be summarized as follows:
  1. Choose k random points to be cluster centers (k represents the number of clusters it will create)
  2. Assign each data point to the cluster whose center is closest
  3. Using the assignments, recalculate the centers of the clusters
  4. Repeat steps 2 and 3 until the clusters are stable (cluster membership does not change or centers change only a little amount)
- The algorithm is easier to understand (as shown above) than the other clustering algorithms. K-Means has an assumption that the clusters of data are of spherical shapes and that the clusters have the same amount of variance (roughly the same number of data points in each cluster) compared to one another. Those assumptions imply that K-Means is not the ideal clustering algorithm to use when clusters are of different variances or of non-spherical shapes. K-Means employs hard assignment of data points into clusters, meaning that a data point is either considered in a cluster or not.
- K-Means would be a poor choice clustering algorithm to use for the scatter plots above. One reason why is because the majority of the clusters are of elliptical shapes, not spheres. The clusters with each graph have significant differences in densities as well which means K-Means will not perform optimally with this data. The first and third graphs seem to each have outliers (discussed above), which is another reason to not use K-Means. K-Means does not have a concept of noise, and so it will group outliers into clusters (instead of classifying them as noise like DBSCAN would).

- Data good for K-Means: Low-dimensional data with clusters of the same variance and of sphereical shapes. Scales well with large datasets.
- Data bad for K-Means: High-dimensional data with clusters of varying variances, data containing overlapping and/or touching clusters, data with a lot of noise or outliers.
- **Gaussian Mixture Models (EM)**
  - The Gaussian Mixture Models with Expectation Maximization (GMM w/ EM) is a clustering algorithm that also takes an iterative approach to clustering data but is more complex than K-Means.
- The algorithm can be summarized as follows:
  1. Choose k random points to be cluster centers (k represents the number of clusters it will create)
  2. For each data point, calculate its probability of being in each cluster. Note - The higher the probability a data point has for a cluster, the more influence it will have on the cluster in the next step.
  3. Using the probabilities, recalculate the means and variances for each of the clusters.
  4. Repeat steps 2 and 3 until the clusters are stable (cluster membership does not change or centers change only a little amount)
- The GMM w/ EM algorithm is more complicated than K-Means, but has less assumptions. First, the variances of the clusters are not assumed to be the same in GMM w/ EM. And so GMM w/ EM make less assumptions about the shapes of the clusters. Instead of limiting the shape of a cluster to be of a spherical shape only, it recognizes clusters that are of elliptical shapes. By being able to recognize elliptical cluster shapes, GMM w/ EM is a lot more flexible in discovering clusters. Often times there are patterns or clusters that are close to each other and so K-Means would often miss in differentiating patterns between two clusters in those situations because it would group patterns together in one sphere. GMM w/ EM would more likely be able to recognize that there different patterns observed between the two groups of data points. GMM w/ EM is great in working with data that has overlapping clusters or data points, this is partially because GMM w/ EM employs soft assignment. In other words, the probabilities that a data point is in each cluster is calculated instead of just grouping the data point in a cluster. The downside of GMM w/ EM is that like K-Means, it does not have a concept of noise. And so it does not properly assign outliers to a group of noise. Another downside is that GMM w/ EM is generally more computationally expensive because the math behind the scenes of the algorithm is more complicated and involved than the math performed in the K-Means algorithm. Despite these downsides, **GMM w/ EM would be the best choice clustering algorithm to use with the three graphs above**. This is because the majority of the clusters in all of the graphs are of elliptical shapes, the clusters in graph have varying densities, and some of the clusters have a lot of overlapping of points within themselves.
- Data good for GMM w/ EM: elliptical-shaped cluster data, low-dimensional data, overlapping and/or touching clusters, overlapping data points within clusters. Better to use with smaller datasets because computationally expensive.
- Data bad for GMM w/ EM: High-dimensional data with clusters of non-elliptical and non-convex shapes, data containing overlapping and/or touching clusters, data with a lot of noise or outliers, large datasets ~ computationally expensive.

- **Hierarchical Agglomerative Clustering (HAC)**
  - The Hierarchical Agglomerative Clustering algorithm takes an iterative bottom-up approach to cluster data points. The algorithm first classifies each data point as its own separate cluster. Then the similar clusters are iteratively combined. The result of this is a hierarchical relationship which can be visualized via a dendrogram graph. The algorithm can be summarized as follows:

1. Classify each data point as a cluster.
2. Determine the similarities between all of the clusters with each other.
3. Combine the similar clusters.
4. Repeat steps 2 and 3 until the clusters are merged.

- The HAC algorithm is a complicated, but flexible algorithm. It has an assumption in which it assumes each data point is its own cluster in the first iteration. The clusters are then merged iteratively until all clusters are merged, and so overall it assumes that the data points have a hierarchical relationship among each other. With these assumptions in mind, HAC is ideal for data that is hierarchical by nature. For example, HAC would be good at clustering a dataset that consists of different animal or plant groups if given the appropriate type of data. A data scientist can gain insight on whether they are working with hierarchical data or not by simply looking at a scatter plot of the data. If the data scientist observes sub-groups or sub-clusters within clusters, then that is most likely a sign the data being worked with is hierarchical by nature. HAC has some disadvantages. First, it is often considered very computationally expensive and this is because its bottom up approach of making every data point its own cluster and iteratively merging clusters. Another disadvantage of HAC is that it does not have a concept noise, and so outliers are not properly assigned to a group of noise like they would be in DBSCAN.
- HAC would be a great choice algorithm to cluster the data of the three graphs above, but not the best choice. It would be a great choice because there are potential sub-groups observed in at least one of the clusters for each graph. HAC is not sensitive to different cluster variances like K-Means which is another reason to use HAC with this data considering that the clusters of each graph seem to have varying densities. Although HAC is a great choice to use, I believe GMM w/ EM is still a better choice because the main clusters that I have observed have elliptical shapes, which is ideal for GMM w/ EM. The downside of computation efficiency for HAC is not a reason why GMM w/ EM is better because GMM w/ EM also is computationally expensive.
- Data good for HAC: data that is hierarchical by nature and high-dimensional datasets. Better to use with smaller datasets because of computationally efficiency.
- Data bad for HAC: Bad for large datasets ~ computationally expensive. Not good when there is a lot of noise or outliers with the data.
- **Density Based Spatial Clustering of Applications with Noise (DBSCAN)**
  - The DBSCAN algorithm approaches clustering by focusing on densities, being sensitive to its hyperparameters, and by using the concept of noise. The algorithms of DBSCANS focus on clustering data together that are densely packed or near each other. The main three parameters for DBSCAN are the following:

1. Distance Metric (Euclidean, Manhattan, etc)

2. Epsilon (can be thought of as the radius length for the clusters)
3. Minimum points (min points that a cluster is to have)

- DBSCAN is highly sensitive to these parameters above and so the data scientist must carefully choose them to effectively use DBSCAN. The concept of noise means that DBSCAN attempts to detect or recognize outliers as noise. This concept of noise is unique in that it does not exist in the other clustering algorithms discussed. DBSCAN does not make any assumptions about the shapes of clusters like in K-Means or GMM w/ EM. DBSCAN has disadvantages. First, it not very effective when the data provided is high dimensional. Second, it is not great when there are clusters overlapping and/or touching. Lastly, it is not optimal when the clusters are of different densities.
- DBSCAN is not a good choice algorithm to cluster the data from the above scatter plots. This is mainly because there is a lot of overlap of data points within each of the three graphs above. DBSCAN is also not great for this dataset because the clusters seem to be of varying densities in each of the graphs. Although DBSCAN has a concept of noise, it is not enough reason for DBSCAN to be considered a great or the best choice for the data provided.
- Data good for DBSCAN: low-dimensional data, smaller datasets (not large ones), data/graphs with a lot of noise/outliers
- Data bad for DBSCAN: high-dimensional data, very large datasets, data in which clusters are overlapping and/or touching, data containing clusters of varying densities

5 C)

[57]:
```
# c
# features of interest from data frame
features = ['Sodium_100g', 'Total_Fat_100g', 'Sugar_100g']

# create a data series of relevant data
X = data[features]

# z-score data before passing it to the model
z = StandardScaler()
X[features] = z.fit_transform(X)
```

[55]:
```
def calculate_best_n(data_frame, number_tests, n_components):

    for i in n_components:
        curr_test_sils = [] # list that stores the diff sil scores for each j␣
↪number_tests
        for j in range(0, number_tests):
                gmm = GaussianMixture(n_components = i) # same number of␣
↪clusters for each j number_tests
                gmm.fit(data_frame)

                clusters = gmm.predict(data_frame)
```

```
                curr_test_sils.append(silhouette_score(data_frame, clusters))

                # calculate the mean of the sil scores in the last loop and␣
↪then print it
                if(j == number_tests - 1):
                    avg_sil_score = np.mean(curr_test_sils)
                    print("The average score for " + str(i) + " is: " +␣
↪str(avg_sil_score))
```

[67]:
```
calculate_best_n(X, 100, [2,3,4,5,6,7]) # pass relevant data series, 100 tests,␣
↪pass the diff n_components to use
```

```
The average score for 2 is: 0.7481864593538067
The average score for 3 is: 0.7067081278737639
The average score for 4 is: 0.6342002692420872
The average score for 5 is: 0.5639719747069547
The average score for 6 is: 0.5213366908048002
The average score for 7 is: 0.4902627244744071
```

5.1 C)

5.2 Question: Describe how you chose any hyperparameters

5.3 Answer:

- The GaussianMixture Class from sklearn has only one required parameter, n_components, which tell the model how many clusters to form with the provided data. To determine the best n_components value to use, I used a custom homemade function, calculate_best_n, and my observations of the original scatter plots. My custom function runs a n - number_tests on a each component value provided in which it calculates each silhouette score. The function then calculates the average silhouette score for that particular n_component value. The function repeats these steps until the average silhouette scores for each n_component value has been calculated. I decided to try out the different n_components: [2 through 7] because it is very unlikely that there are more than 7 clusters considering the segmentation of data observed in the original scatterplots. From looking at the original scatterplots, I though that the best number of clusters to use must either be 2 or 3 because it looked like there were 2 or 3 clusters in each of the graphs. I supplied 100 to my function to make it run the tests 100 times to ensure that the results I get are reliable and consistent. As shown above, the highest average silhouette score is 2 and I am confident in this answer because it is based on an average of 100 iterations.

[58]:
```
EM = GaussianMixture(n_components = 2)

EM.fit(X)
```

[58]:
```
GaussianMixture(n_components=2)
```

[59]:
```
cluster = EM.predict(X)
X["cluster"] = cluster
```

[60]:
```
EM_silhouette_score = silhouette_score(X, cluster)
print("GMM w/ EM Model Silhouette Score: " + str(EM_silhouette_score))
```

```
GMM w/ EM Model Silhouette Score: 0.7481864593538068
```

6 D)

6.1 Question: Which metric did you use to asses your model?

6.2 Answer

- I used silhouette scores to assess the effectiness of my model because it measures how effective a clustering algorithm segmented its data. The silhouette score is calculated based on the cohesion and separation of the clusters. Cohesion is a measure of how similar the cluster members are to each other. Separation is a measure how different the clusters are from each other. The more cohesion and separation will mean the higher the silhouette score will be. Silhouette scores have a range of [-1, 1] where a value closer to +1 is better.

6.3 Question: How did your model perform?

6.4 Answer

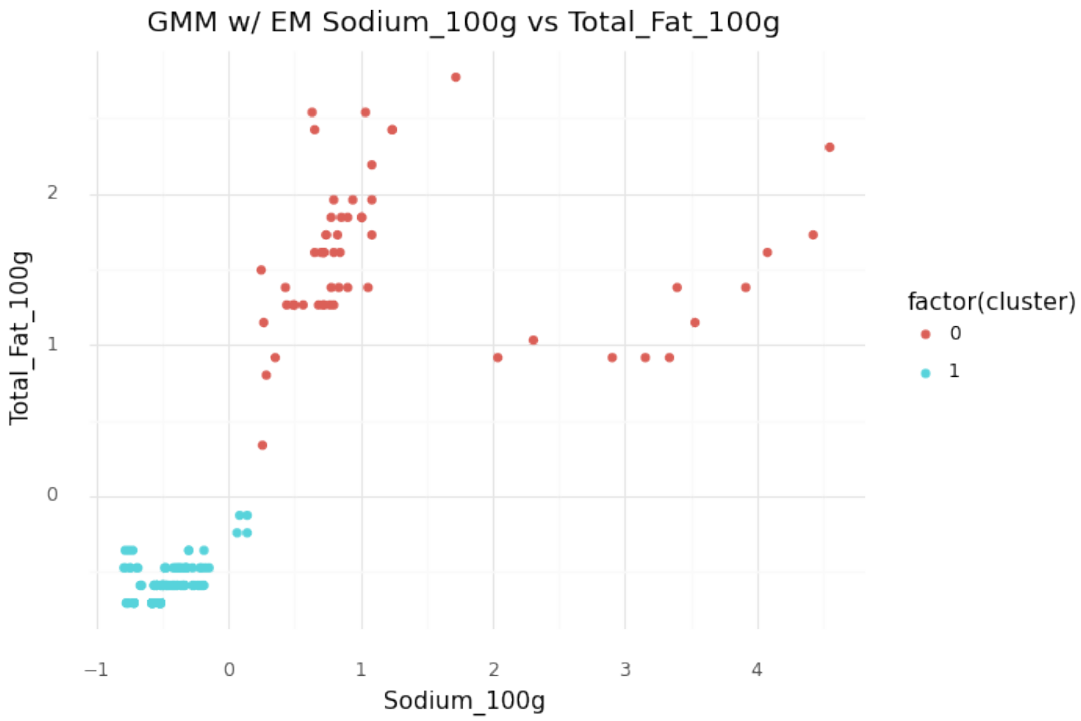
- After determining the best n_components to use for my model, I calculated its silhouette score (shown above) and got a score of about 0.748. This is a good score because it is pretty close to +1, and so the model performed moderately well.

7 D cont)

7.1 Question: Remake the 3 graphs from part a, but color by cluster assignment. Describe what characterizes each cluster, and give an example of a label for that cluster

7.2 Answer:

[68]:
```
# Sodium_100g vs Total_Fat_100g (Part C Model)
(ggplot(X, aes(x = "Sodium_100g", y = "Total_Fat_100g", color =␣
 ↪"factor(cluster)")) + geom_point() + theme_minimal() + ggtitle("GMM w/ EM␣
 ↪Sodium_100g vs Total_Fat_100g") + labs(x = "Sodium_100g", y =␣
 ↪"Total_Fat_100g"))
```



[68]: <ggplot: (315033149)>

7.3 GMM w/ EM Sodium_100g vs Total_Fat_100g Discussion

- The GMM w/ EM model separated the clusters as shown above. The blue-colored cluster located in the bottom left of the graph is characterized by low sodium and low fat, and so I would label this cluster as "Guilt-Free Doughnuts" because low sodium and low fat is generally healthier. The Guilt-Free Doughnuts cluter is fairly dense, tells us its members are very similar to each other. The other cluster, red-colored, is characterized by high fat and a varying sodium levels. Because higher levels of fats and sodiums is generally considered unhealthy I will label it as the "Cheat Day Doughnuts". The Cheat Day Doughnuts cluster is a lot more spread out compared to the other cluster, which tells us that their members are not always so similar. It almost seems like there is 2 groups within Cheat Day Doughnuts, the first being high fat, lower sodium and the second being high fat, higher sodium. If I had to guess, I would assume that the first sub-group is of sweeter tasting unhealthy choices and the second is of saltier tasting unhealthy choices.

[71]:
```
# Sugar_100g vs Total_Fat_100g (Part C Model)
(ggplot(X, aes(x = "Sugar_100g", y = "Total_Fat_100g", color =␣
↪"factor(cluster)")) + geom_point() + theme_minimal() + ggtitle("GMM w/ EM␣
↪Sugar_100g vs Total_Fat_100g") + labs(x = "Sugar_100g", y =␣
↪"Total_Fat_100g"))
```

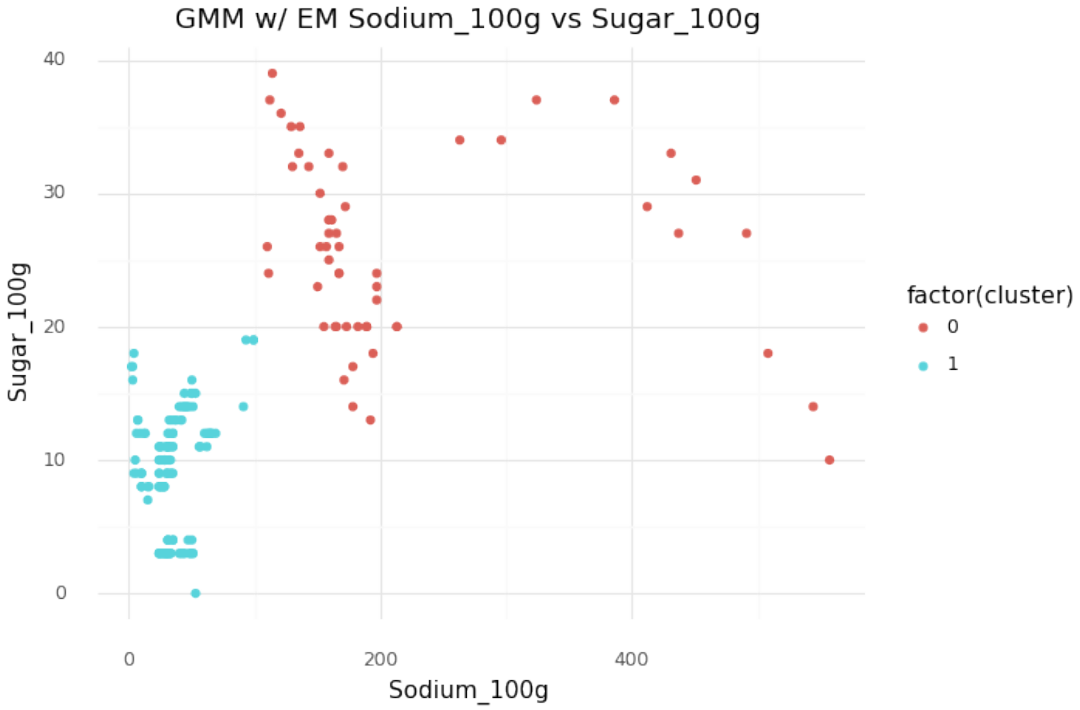
[71]: <ggplot: (312631065)>

7.4 GMM w/ EM Sugar_100g vs Total_Fat_100g Discussion

- The GMM w/ EM model separated the clusters as shown above. The blue-colored cluster located in the bottom left of the graph is characterized by lower sugar and low fat, and so I would label this cluster as "Nearly Guilt-Free Doughnuts" because lower sugar and low fat is generally healthier. The Nearly Guilt-Free Doughnuts cluster is fairly dense, which tells us its members are very similar to each other. Some of the data points in this cluster almost have no sugar, and so it would not be suprising if this cluster is not even composed of doughnuts (or are doughnut holes). The other cluster, red-colored, is characterized by varying higher sugar and varying higher sugar levels. Because higher levels of sugars and fats is generally considered unhealthy I will label it as the "Do Not Eat too many of these Doughnuts". The Do Not Eat too many of these Doughnuts cluster is a lot more spread out compared to the other cluster, which tells us that their members are not always so similar. Although there is a significant amount of spread observed in this cluster, there does not seem to be any sub-groups observed within this cluster. Since there does not seem to be sub-groups in this cluster, I think that the type of doughnuts here are varying based on the type of doughnuts made. Some doughnuts have more sugar because Krispy Kreme intends for them to be very sweet and others have less because Krispy Kreme intends for them to be not super sweet.

[70]:
```
# Sodium_100g vs Sugar_100g (Part C GMM w/ EM Model)
```

```
(ggplot(data, aes(x = "Sodium_100g", y = "Sugar_100g", color =␣
↪"factor(cluster)")) + geom_point() + theme_minimal() + ggtitle("GMM w/ EM␣
↪Sodium_100g vs Sugar_100g") + labs(x = "Sodium_100g", y = "Sugar_100g"))
```



[70]: <ggplot: (315117677)>

7.5 GMM w/ EM Sodium_100g vs Sugar_100g Discussion

- The GMM w/ EM model separated the clusters as shown above. The blue-colored cluster located in the bottom left of the graph is characterized by low sodium and varying medium sugar, and so I would label this cluster as "Healthier Doughnuts" because low sodium and low sugar is generally healthier. The Healthier Doughnuts cluter is fairly dense, which tells us that its members are very similar to each other. The other cluster, red-colored, is characterized by high sugar and a varying sodium levels. Because higher levels of sugar and sodium is generally considered unhealthy I will label it as "Dangerous Doughnuts". The Dangerous Doughnuts cluster is a lot more spread out compared to the other cluster, which tells us that their members are not always so similar. It almost seems like there is 2 groups within Dangerous Doughnuts, the first being high sugar, lower sodium and the second being high sugar, higher sodium. Considering that both of these observed sub-groups have very high sugar, I think the "Dangerous Doughnuts" name is appropriate. If I had to guess, I would assume that the first sub-group is of sweeter tasting unhealthy choices and the second is of sweet & salty tasting unhealthy choices.

8 E

8.1 Question: Choose ONE other of the _100g variables from the data set to add to your clustering model to improve it

8.2 Answer:

[81]:
```
mod_features = ['Sodium_100g', 'Total_Fat_100g', 'Sugar_100g' ,"Protein_100g"]

# create a data series of relevant data
mod_X = data[mod_features]

# z-score data before passing it to the model
z = StandardScaler()
mod_X[mod_features] = z.fit_transform(mod_X)
```

[82]:
```
mod_X.head()
```

[82]:

	Sodium_100g	Total_Fat_100g	Sugar_100g	Protein_100g
0	0.247520	1.496117	1.354003	0.705159
1	3.396008	1.380117	1.462386	0.705159
2	0.777083	1.380117	0.703701	1.310002
3	0.565258	1.264117	2.004305	1.310002
4	2.904959	0.916117	2.546223	0.705159

9 E cont)

9.1 Question: explain why you chose this variable.

9.2 Answer:

- I added Protein_100g as a feature of interest for my modified model. I chose to add protein as a feature because I observed sub-groups when there was low sugar and low fat as well as when there was low sugar and low sodium. The common denominator in those sub-groups is low sugar, and generally foods of higher proteins such as meat or eggs have very low sugar. I think by adding protein, I will be able to gain further insight on these observed sub-groups. I suspect that these groups have higher protein levels and so the model will cluster them separately.

10 E cont)

10.1 Question: Make a new model, identical to the model in part c, but also including your new variable.

10.2 Answer:

[83]:
```
calculate_best_n(mod_X, 100, [2,3,4,5,6,7]) # pass relevant data series, 100␣
↪tests, pass the diff n_components to use
```

```
The average score for 2 is: 0.7170330965661261
The average score for 3 is: 0.6185476289802686
The average score for 4 is: 0.5366619101014246
The average score for 5 is: 0.4770979894309064
The average score for 6 is: 0.45662315446377655
The average score for 7 is: 0.4662482203443175
```

[84]:
```
mod_EM = GaussianMixture(n_components = 2)

mod_EM.fit(mod_X)
```

[84]:
```
GaussianMixture(n_components=2)
```

[85]:
```
mod_cluster = mod_EM.predict(mod_X)
mod_X["mod_cluster"] = mod_cluster
```

[86]:
```
mod_EM_silhouette_score = silhouette_score(mod_X, mod_cluster)
print("MOD GMM w/ EM Model Silhouette Score: " + str(mod_EM_silhouette_score))
```

```
MOD GMM w/ EM Model Silhouette Score: 0.7261613744946628
```

11 E cont)

11.1 Question: Did this variable improve the fit of your clustering model? How can you tell?

11.2 Answer:

- The variable did not improve the fit of the clustering model, it actually gave worse results. I can tell this because I calculated the silhouette scores for the new model by using my custom function. The best average silhouette score I got is about 0.717 which is nearly 0.03 points worse than the original model's average silhouette score. Based on these results, it seems like protein is not an important feature to consider when clustering food items from KrispyKreme. But this may be the case because I am using a GMM w/ EM model. The result of adding a feature to the model will be different depending on the type of model being used. And so perhaps if I was using a HAC model, the clustering would maybe improve especially considering that HAC is good recognizing hierarchical patterns. With this in mind, I am curious as to how my results would differ if I had chosen HAC instead of GMM w/ EM.