BİTİRME PROJESİ

May 19, 2018

Contents

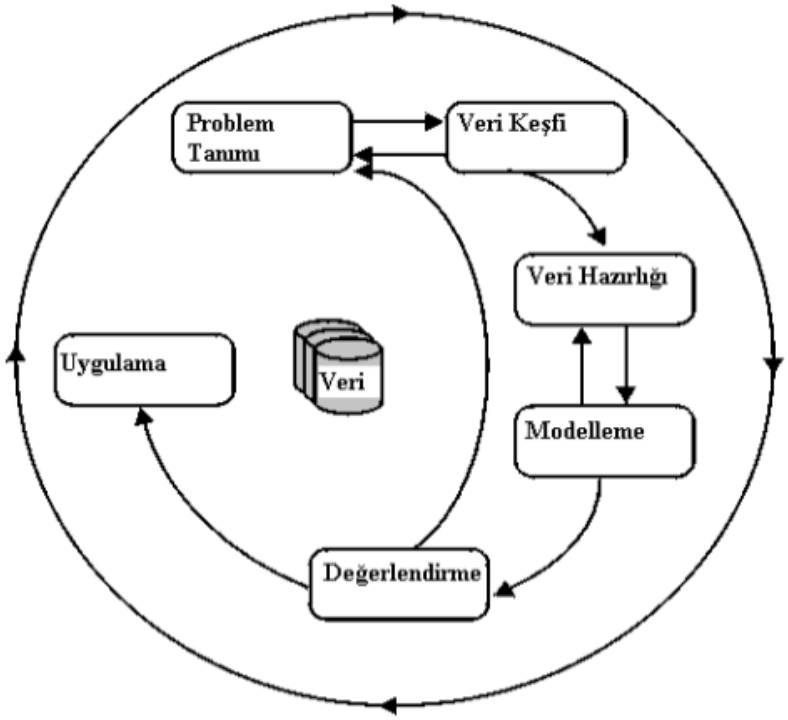
0.1 VERİ MADENCİLİĞİ

Veri madenciliği için birçok farklı tanım bulunmaktadır. Amerikan Pazarlama Birliği (AMA) veri madenciliğini şu şekilde tanımlamıştır "Verilerin, yeni ve potansiyel olarak yararlı bilgi bulma amaçlı analiz süreci. Bu süreç bulunması zor örüntülerin ortaya çıkarılması için matematiksel araçların kullanımını içerir." Gartner tarafından verilen bir diğer tanım ise şöyledir "Veri depolarında saklanan büyük miktarda veri üzerinden eleme ile anlamlı korelasyonlar, örüntüler ve trendler keşfetme süreci. Veri madenciliği, örüntü tanıma teknolojilerinin yanı sıra matematiksel ve istatistiksel teknikleri kullanmaktadır."

Bir diğer deyişle veri madenciliğiyle elde bulunan verilere çeşitli algoritmalar uygulanıp çıkan sonuçlarla birlikte veri hakkında yorum yapılabilmektedir ve istatiksel hesaplamalara dayanır. Çoğunlukla büyük hacimli gözlemsel veri kümelerinin analizini yapıp, beklenmedik ilişkileri bulmak, yeni yöntemler kullanarak anlaşılabilir ve faydalı bir biçime sokmak olarak da tanımlanabilir.

Veri madenciliğinin geçmişi onlarca yıl ötesine dayanmasına karşın, veri miktarının çok arttığı ve hemen her işlemin dijital ortamda kayıt edilebildiği son on beş yılda kullanılırlığı daha da artmıştır. Hem istatistik biliminin kullandığı algoritma ve yöntemleri hem de yapay zekânın bir bileşeni olan makine öğrenmesi algoritmalarını kullanır. Veri madenciliğinin temelde üç ayrı yöntemi vardır. Bunlar, sınıflandırma, kümeleme ve birliktelik kuralları çıkarımıdır. Bu üç yöntem farklı alanlarda kullanıldığı gibi birlikte de kullanılabilir. Bunun dışında vereceği sonuçlara göreyse tahminleyici, tanımlayıcı ve önerici olacak şekilde üç ayrı kategori söylenebilir.

```
In [19]: from IPython.display import Image, display
         display(Image(filename='Desktop/MyPicture.png', embed=True))
```



0.1.1 Adımlarla Veri Madenciliği

Veri madenciliği için endüstriler arası bir standart model 1996 yalında CRISP-DM adıyla bir modelde sunuldu. Buna göre bir veri madenciliği projesinin adımları aşağıdaki gibi olmalıdır.

1) Problem Tanımı: İşin farkında varılma evresidir. Projenin amacı, hedefleri ve gereksinimleri belirlenir.

2) Veri Keşfi: Verinin toplanması, incelenmesi ve tarif evresidir. Verinin kalitesi veya sorunları belirlenir.

3) Veri Hazırlama: Modelleme süreci için veri modeli oluşturulur. Veri kullanılabilir formata dönüştürülür. Yeni değişkenler de oluşturulabilir. Veri bu aşamada birden fazla değişime uğrayabilir. Modelleme aracı için veri hazır hale getirilir.

4) Modelleme: Aynı veri üzerinde farklı matematiksel modeller kullanılarak çözümleme yapılma evresidir. Bir problemi çözmek veya veri hakkında analiz yapabilmek için birden fazla yöntem kullanılabilir. Hangi modelin veriye uygun olup olmadığının kararını doğru verebilmek için her bir model değerlendirilmelidir. En iyi değerler elde edilene kadar farklı modeller kullanmak bir çözümken aynı modeli farklı parametreler kullanarak uygulamak da modelin kalitesi için bir seçenektir. Tüm bu aşamalar uygulandığında ise yüksek kaliteli bir model ortaya çıkmış olur.

5) Değerlendirme: Kurulan model değerlendirilir. Beklentilere ne derece karşılık verip vermediği üzerinde yorumlar yapıldığı evredir. Model beklentileri karşılamadığı taktirde modelleme aşamasına geri dönüp yöntemler ve parametreler tekrar incelenir. Modeli yeniden kurmak bile gerekebilir. Eğer model iş hedeflerine ulaşıyorsa, veri madenciliği sonuçlarının nasıl kullanılacağına karar verilir.

6) Uygulama: Sonuçlar diğer uygulamalara aktarılır.

0.1.2 Denetimli ve Denetşmsiz Öğrenme Nedir?

Makine Öğrenmesi önceden yapılan gözlemlere dayanarak doğru tahminler yapabilmeyi öğrenebilmek amacıyla sistemlerin geliştirilmesidir. Makine Öğrenmesi kullanılarak oluşturulan sistemler genelde iki farklı öğrenme modeli kullanmaktadır. Bu modeller denetimli (supervised) ve denetimsiz (unsupervised) öğrenme modelidir.

Denetimsiz Öğrenme: Denetimsiz öğrenme modelinde sistem eğitilirken ön bilgisiz veri kullanılarak öğrenmesi sağlanır. Denetimsiz öğrenmede amaç veri setindeki örneklerin çıkışları bilinmediği için tanıma veya sınıflandırma değildir. Genellikle kümeleme, olasılık yoğunluk tahmininde kullanılır. Ek olarak denetimsiz öğrenme algoritması ile elde edilen sonuçlar denetimli öğrenme için de kullanılabilmektedir.

Denetimli Öğrenme: Denetimli öğrenme modelinde sistemin ön bilgili veriler kullanılarak eğitilmesi ile öğrenmenin sağlanmasıdır. Sistem eğitilirken veri setinde bulunan her bir örneğe ait giriş ve çıkışlar verilir. Test veri seti ise sistemin doğrulanması amacıyla kullanılır. Sistemin doğrulanması aşamasında öğrenme algoritması kategorisi bilinmeyen bir test verisine, eğitim verisinde bulunan çıkışlardan herhangi birini atar. Denetimli öğrenme modelinde problem, sınıflandırma problemi olarak ele alınır ve eğitilmiş sistem test setine yönelik tahmin ve tanıma amacıyla kullanılır. Karar ağaçları denetimli öğrenme modellerindendir.

0.1.3 K- means nedir?

En eski kümeleme algoritmalarından olan k-means, 1967 yılında J.B. MacQueen tarafından geliştirilmiştir. En yaygın kullanılan denetimsiz öğrenme yöntemlerinden birisi olan K-means'in atama mekanizması, her verinin sadece bir kümeye ait olabilmesine izin verir. Bu nedenle keskin bir kümeleme algoritmasıdır. Merkez noktanın kümeyi temsil etmesi ana fikrine dayalı bir yöntemdir. Eşit büyüklükte küresel kümeleri bulmaya çalşır. Yöntemi uygulayabilmek için öncelikle istenilen sayıda K kümesi berlitmemiz gerekmektedir. Küme sayısını bulabilmek için ise Elbow Metodunu kullandım aşağıda ayrıntılı açıklamasını anlatacağım.

Her verinin sadece bir kümeye ait olmasına izin vermektedir. Temel amaç, gerçekleştirilen bölümleme işlemi sonunda elde edilen kümelerin, küme içi benzerliklerinin maksimum ve kümeler arası benzerliklerinin minimum olmasını sağlamaktır.

En düşük WSS değerine sahip kümeleme sonucu en iyi sonucu verir. Nesnelerin bulundukları kümenin merkez noktalarına olan uzaklıklarının karelerinin toplamı aşağıdaki formülle bulunur.

Squared Euclidean Distance

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i \in C_k} \sum_{j=1}^{P} (x_{ij} - x_{ij})^2$$

$|C_k|$: k. kümedeki veri sayısı

Elimizde verilerimiz var ve bu verileri kümelere koyduğumuzu varsayalım. Yalnız iki farklı kümede aynı veri olmasın. $C_1, C_2, ...C_K$ ise kümelerimiz olsun. Wss ne kadar küçükse kümelememiz o kadar iyidir.C_k kümesi için kümelerin birbirlerinden farkını gö $W(C_k)$ ile gösterelim. Veri setimizi K kümeye ayıracağız. Within-cluster toplamı mümkün olabilidğince küçük olacaktır.

$$\underset{C_1, C_2, ..., C_k}{minimize} \sum_{k=1}^{K} W(C_k)$$

k. küme için within-cluster değişkeni, k. kümedeki verilerin tüm ikili squared Euclidean distances toplamının k. kümedeki toplam veri sayısına bölümüdür.

Formüller birleştirilirse

$$\underset{C_1, C_2, ..., C_k}{minimize} \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i \in C_k} \sum_{j=1}^{P} (x_{ij} - x_{ij})^2$$

0.2 K means Pseudo Code

In [20]:
```
display(Image(filename='Desktop/code.png', embed=True))
```

Algorithm 1: K-Means Algorithm

Input: $E = \{e_1, e_2, \ldots, e_n\}$ (set of entities to be clustered)
k (number of clusters)
$MaxIters$ (limit of iterations)

Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
$L = \{l(e) \mid e = 1, 2, \ldots, n\}$ (set of cluster labels of E)

foreach $c_i \in C$ **do**
  $c_i \leftarrow e_j \in E$ (e.g. random selection)
end
foreach $e_i \in E$ **do**
  $l(e_i) \leftarrow argminDistance(e_i, c_j) j \in \{1 \ldots k\}$
end

$changed \leftarrow false$;
$iter \leftarrow 0$;
repeat
  foreach $c_i \in C$ **do**
    $UpdateCluster(c_i)$;
  end
  foreach $e_i \in E$ **do**
    $minDist \leftarrow argminDistance(e_i, c_j)\ j \in \{1 \ldots k\}$;
    if $minDist \neq l(e_i)$ **then**
      $l(e_i) \leftarrow minDist$;
      $changed \leftarrow true$;
    end
  end
  $iter++$;
until $changed = true$ *and* $iter \leq MaxIters$;

In [21]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sns
```

Numpy nedir? Numpy, pyhtonda kullanılan temel pakettir. Bilimsel hesaplamalarda kullanılır. Matematiksel ve mantıksal fonksiyonları hesaplamada yardımcı olduğu gibi çok boyutlu diziler ve matrislerde de hızlı işlemler yapılabilmesini sağlar. Bunların yanı sıra rastgele veri tipleri tanımlanabilir bu sayede çok çeşitli veri tabanları ile sorunsuz ve hızlı bir şekilde entegre olunabilir.

Matplotlib.pyplot nedir? Hem iki boyutlu hem de üç boyutlu grafikler oluşturulabilen, verileri görselleştirebilen kuvvetli bir paket olan matplotlib.pyplot bilimsel programlamada önemli bir yere sahiptir. Grafik ekrana show() komutu ile getirilir.

Pandas nedir? Açık kaynak kodlu bir kütüphane olup kullanım kolaylığıyla birlikte veri yapıları ve veri analiz araçları sunmaktadır.

```
In [22]: df=pd.read_csv(r'C:\Users\asus\Desktop\menu1.csv')
         df.head()
```

Out[22]:

	Category	Item	Serving Size	Calories	\
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	
4	Breakfast	Sausage McMuffin with Egg Whites	5.7 oz (161 g)	400	

	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	\
0	120	13.0	20	5.0	
1	70	8.0	12	3.0	
2	200	23.0	35	8.0	
3	250	28.0	43	10.0	
4	210	23.0	35	8.0	

	Saturated Fat (% Daily Value)	Trans Fat	...	\
0	25	0.0	...	
1	15	0.0	...	
2	42	0.0	...	
3	52	0.0	...	
4	42	0.0	...	

	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	\
0	31	10	4	
1	30	10	4	
2	29	10	4	
3	30	10	4	
4	30	10	4	

	Dietary Fiber (% Daily Value)	Sugars	Protein	Vitamin A (% Daily Value)	\
0	17	3	17	10	
1	17	3	18	6	
2	17	2	14	8	

```
3                              17       2       21                          15
4                              17       2       21                           6

   Vitamin C (% Daily Value)  Calcium (% Daily Value)  Iron (% Daily Value)
0                          0                       25                    15
1                          0                       25                     8
2                          0                       25                    10
3                          0                       30                    15
4                          0                       25                    10

[5 rows x 24 columns]
```

Pandas sayesinde daha önceden elimizde bulunan csv uzantılı veri setini read_csv ile okuyup bir değişkene atadı. head() metodu ile veri setinin ilk 5 kaydı görülür.

```
In [23]: df.shape

Out[23]: (260, 24)
```

shape sayesinde veri setinin kaç satır ve sütundan oluştuğunu görebiliriz. Uygulanacak işlemler için bunları bilmek önemlidir.

```
In [24]: df.columns

Out[24]: Index(['Category', 'Item', 'Serving Size', 'Calories', 'Calories from Fat',
                'Total Fat', 'Total Fat (% Daily Value)', 'Saturated Fat',
                'Saturated Fat (% Daily Value)', 'Trans Fat', 'Cholesterol',
                'Cholesterol (% Daily Value)', 'Sodium', 'Sodium (% Daily Value)',
                'Carbohydrates', 'Carbohydrates (% Daily Value)', 'Dietary Fiber',
                'Dietary Fiber (% Daily Value)', 'Sugars', 'Protein',
                'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
                'Calcium (% Daily Value)', 'Iron (% Daily Value)'],
               dtype='object')
```

Colums veri setinde olan sütünların isimlerini tek tek görmemezi sağlar.

```
In [25]: print (df.dtypes)

Category                         object
Item                             object
Serving Size                     object
Calories                          int64
Calories from Fat                 int64
Total Fat                       float64
Total Fat (% Daily Value)         int64
Saturated Fat                   float64
Saturated Fat (% Daily Value)     int64
Trans Fat                       float64
Cholesterol                       int64
```

```
Cholesterol (% Daily Value)        int64
Sodium                             int64
Sodium (% Daily Value)             int64
Carbohydrates                      int64
Carbohydrates (% Daily Value)      int64
Dietary Fiber                      int64
Dietary Fiber (% Daily Value)      int64
Sugars                             int64
Protein                            int64
Vitamin A (% Daily Value)          int64
Vitamin C (% Daily Value)          int64
Calcium (% Daily Value)            int64
Iron (% Daily Value)               int64
dtype: object
```

dtypes sayesinde her bir sütunun hangi veri tipinde olduğunu bulabiliriz.

```
In [26]: df.describe()
```

```
Out[26]:           Calories  Calories from Fat   Total Fat  Total Fat (% Daily Value)  \
         count   260.000000         260.000000  260.000000                 260.000000
         mean    368.269231         127.096154   14.165385                  21.815385
         std     240.269886         127.875914   14.205998                  21.885199
         min       0.000000           0.000000    0.000000                   0.000000
         25%     210.000000          20.000000    2.375000                   3.750000
         50%     340.000000         100.000000   11.000000                  17.000000
         75%     500.000000         200.000000   22.250000                  35.000000
         max    1880.000000        1060.000000  118.000000                 182.000000

                Saturated Fat  Saturated Fat (% Daily Value)   Trans Fat  Cholesterol  \
         count     260.000000                     260.000000  260.000000   260.000000
         mean        6.007692                      29.965385    0.203846    54.942308
         std         5.321873                      26.639209    0.429133    87.269257
         min         0.000000                       0.000000    0.000000     0.000000
         25%         1.000000                       4.750000    0.000000     5.000000
         50%         5.000000                      24.000000    0.000000    35.000000
         75%        10.000000                      48.000000    0.000000    65.000000
         max        20.000000                     102.000000    2.500000   575.000000

                Cholesterol (% Daily Value)       Sodium          ...          \
         count                   260.000000   260.000000          ...
         mean                     18.392308   495.750000          ...
         std                      29.091653   577.026323          ...
         min                       0.000000     0.000000          ...
         25%                       2.000000   107.500000          ...
         50%                      11.000000   190.000000          ...
         75%                      21.250000   865.000000          ...
```

```
max                    192.000000  3600.000000          ...

       Carbohydrates  Carbohydrates (% Daily Value)  Dietary Fiber  \
count     260.000000                     260.000000     260.000000
mean       47.346154                      15.780769       1.630769
std        28.252232                       9.419544       1.567717
min         0.000000                       0.000000       0.000000
25%        30.000000                      10.000000       0.000000
50%        44.000000                      15.000000       1.000000
75%        60.000000                      20.000000       3.000000
max       141.000000                      47.000000       7.000000

       Dietary Fiber (% Daily Value)      Sugars     Protein  \
count                     260.000000  260.000000  260.000000
mean                        6.530769   29.423077   13.338462
std                         6.307057   28.679797   11.426146
min                         0.000000    0.000000    0.000000
25%                         0.000000    5.750000    4.000000
50%                         5.000000   17.500000   12.000000
75%                        10.000000   48.000000   19.000000
max                        28.000000  128.000000   87.000000

       Vitamin A (% Daily Value)  Vitamin C (% Daily Value)  \
count                 260.000000                 260.000000
mean                   13.426923                   8.534615
std                    24.366381                  26.345542
min                     0.000000                   0.000000
25%                     2.000000                   0.000000
50%                     8.000000                   0.000000
75%                    15.000000                   4.000000
max                   170.000000                 240.000000

       Calcium (% Daily Value)  Iron (% Daily Value)
count               260.000000            260.000000
mean                 20.973077              7.734615
std                  17.019953              8.723263
min                   0.000000              0.000000
25%                   6.000000              0.000000
50%                  20.000000              4.000000
75%                  30.000000             15.000000
max                  70.000000             40.000000

[8 rows x 21 columns]
```

Describe fonksiyonu sayesinde her bir verimizin kaç satırdan oluştuğunu, ortalama değerini, sapmasını, max ve min değerlerini görebiliyoruz. Bu bilgiler veriler hakkında daha somut yorum yapabilme şansı doğuyor. Calories kolonunu ele aldığımızda 260 satırdan oluştuğunu, ortalamasının 368 olduğunu ve standart sapmasının 240 olduğunu görebiliyoruz. Eğer veriler ortalamaya yakın ise standart sapmanın düşük olacağını, veriler ortalamadan uzaklaştıkça sapmanın da arttığını göz önüne alırsak; elimizdeki veri setinin kolonlarındaki sapmalar verilerin ortalamadan uzaklaştığını, min max arasındaki farkın fazla olduğunu gösteriyor.

Veri setimizdeki verilerin dağılımları hakkında fikir sahibi olduktan sonra veri setimizi kullanmak sitediğimiz hale getirmemiz gerekmektedir. Bunun için daha önce tanımladığım df veri setindeki kullanmak istemediğim kolonları droplamam gerekmektedir. df.drop sayesinde kullanmak istemediğim kolonları dropladım ve yeni oluşan veri setini df1 diye tanımladım. Bundan sonraki analizlerimde artık df1 veri setini kullanacağım.

```
In [27]: df1=df.drop(columns=['Category', 'Item','Serving Size','Total Fat', 'Saturated Fat',
                 'Trans Fat','Total Fat (% Daily Value)',
                 'Saturated Fat (% Daily Value)',
                 'Cholesterol (% Daily Value)', 'Sodium (% Daily Value)', 'Carbohydrates (% Daily
                 'Dietary Fiber (% Daily Value)'])
```

```
In [28]: df1.head()
```

Out[28]:

	Calories	Calories from Fat	Cholesterol	Sodium	Carbohydrates	\
0	300	120	260	750	31	
1	250	70	25	770	30	
2	370	200	45	780	29	
3	450	250	285	860	30	
4	400	210	50	880	30	

	Dietary Fiber	Sugars	Protein	Vitamin A (% Daily Value)	\
0	4	3	17	10	
1	4	3	18	6	
2	4	2	14	8	
3	4	2	21	15	
4	4	2	21	6	

	Vitamin C (% Daily Value)	Calcium (% Daily Value)	Iron (% Daily Value)
0	0	25	15
1	0	25	8
2	0	25	10
3	0	30	15
4	0	25	10

```
In [29]: df1.columns
```

```
Out[29]: Index(['Calories', 'Calories from Fat', 'Cholesterol', 'Sodium',
                'Carbohydrates', 'Dietary Fiber', 'Sugars', 'Protein',
                'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
                'Calcium (% Daily Value)', 'Iron (% Daily Value)'],
               dtype='object')
```

Yeni oluşan veri setimin kolonları artık kullanacağım formata geldi.

```
In [30]: X=df1.values
```

df1 veri setindeki verilerin değerlerini X değişkeninde tutuyoruz.

0.2.1 K Means Modeli Uygulama

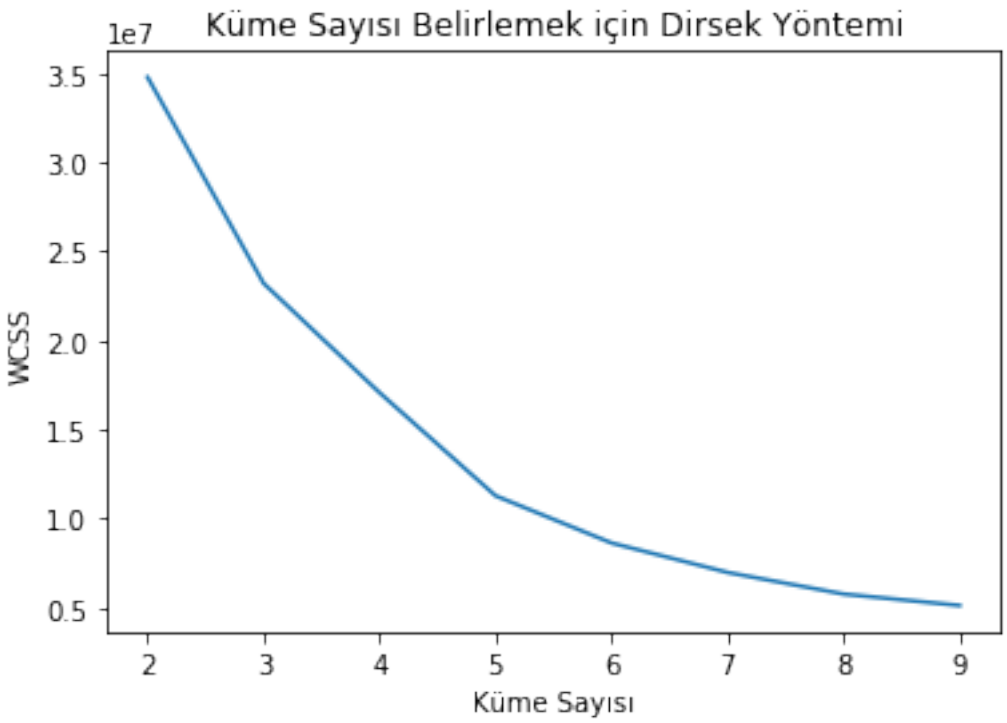
Yukarıda tanımını yaptığım k means modelini uygulayabilmek için öncelike küme sayısının belirlenmesi gerekmektedir. Bu örneğimizde küme sayısını bulmak için Dirsek yöntemini kullanacağım. Öncelikle Dirsek yöntemi nedir ve işleyiş şekli nasıldır bunlara bakmalıyız.

0.2.2 Matematiksel Açıklaması

Elimizde verilerimiz var ve bu verileri kümelere koyduğumuzu varsayalım. Yalnız iki farklı kümede aynı veri olmasın. $C_1, C_2, ...C_K$ ise kümelerimiz olsun. Wss ne kadar küçükse kümelememiz o kadar iyidir.C_k kümesi için kümelerin birbirlerinden farkını gö $W(C_k)$ ile gösterelim. Veri setimizi K kümeye ayıracağız. Within-cluster toplamı mümkün olabilidğince küçük olacaktır.

$$\underset{C_1,C_2,...,C_k}{minimize} \sum_{k=1}^{K} W(C_k)$$

```
In [31]: wcss = []
         kume_sayisi_listesi = range(2,10)
         for i in kume_sayisi_listesi :
             kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, ra
             kmeans.fit(X)
             wcss.append(kmeans.inertia_)
         plt.plot(kume_sayisi_listesi, wcss)
         plt.title('Küme Sayısı Belirlemek için Dirsek Yöntemi')
         plt.xlabel('Küme Sayısı')
         plt.ylabel('WCSS')
         plt.show()
```



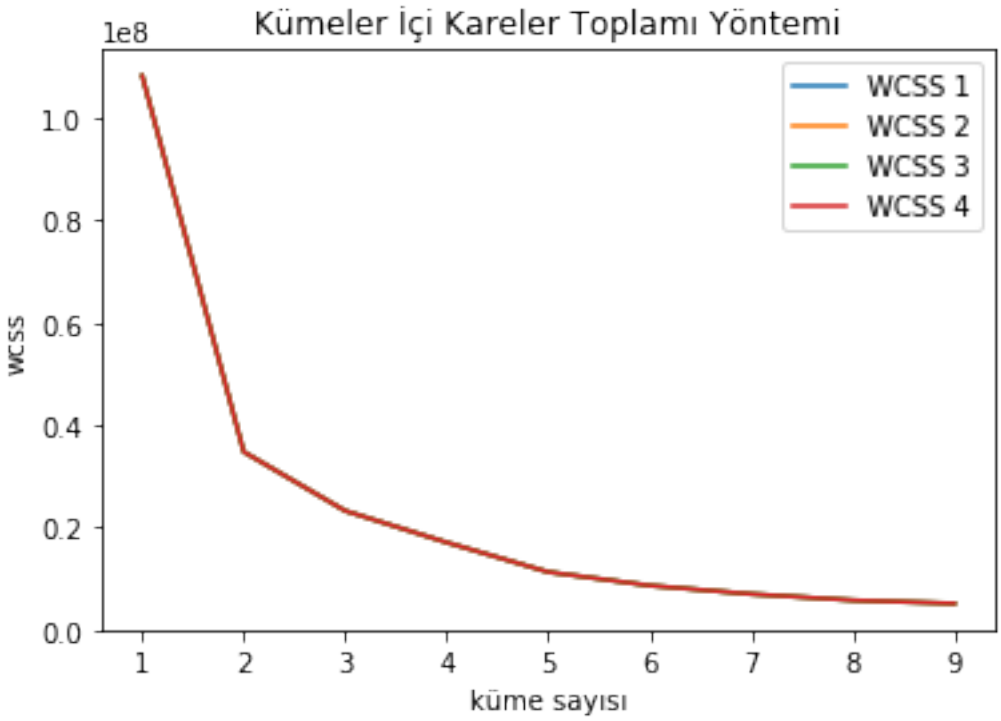
Kod İşleyiş

1. K means adında bir nesne oluşturulur.
2. for döngüsü i değişkeniyle her dönüşünde bir artarak küme sayısını parametre olarak n_clusters'a verir.
3. init parametresi başlangıç noktalarını seçmek için ideal küme merkezlerini belirler.
4. max_iter en fazla kaç iterasyona kadar gideceğini gösterir. Default olarak 300 seçilir. 400 veya 200 seçilmesi bir şeyi değiştirmeyeceğini daha sonraki adımlarda göstereceğim.
5. n_init küme merkezi başlangıç noktasının kaç farklı noktadan başlayabileceğini belirler. Default olarak 10 alınır.

Küme sayısı bize neyi ifade ediyor? Küme sayımızı belirledikten ve algoritmayı uyguladıktan sonra kullandığımız algoritma her bir noktaya bir kümeye yerleştirdi. Daha sonra her bir noktanın küme merkezine olan uzaklığının karesinin toplamını alıyoruz. Bu uygulamayı her küme için tekrarlıyoruz. Toplamda çıkan rakam ne kadar düşükse yapılan kümeleme o kdar iyi çalışmış, merkez nokta ile kümeye dahil noktalar birbirine yakındır demektir.

0.3 Farklı max iter, random state değerleri herhangi bir değişiklik yapar mı?

```
In [32]: wcss1=[]
         wcss2=[]
         wcss3=[]
         wcss4=[]
         for i in range (1,10):
             kmeans = KMeans(n_clusters = i, init='k-means++', n_init=i, random_state=1)
             kmeans.fit(df1)
             wcss1.append(kmeans.inertia_)

         for i in range (1,10):
             kmeans = KMeans(n_clusters = i, init='k-means++', max_iter=150, n_init=i, random_st
             kmeans.fit(df1)
             wcss2.append(0.5+kmeans.inertia_)

         for i in range (1,10):
             kmeans = KMeans(n_clusters = i, init='k-means++', max_iter=250, n_init=i, random_st
             kmeans.fit(df1)
             wcss3.append(1+kmeans.inertia_)

         for i in range (1,10):
             kmeans = KMeans(n_clusters = i, init='k-means++', max_iter=300, n_init=i, random_st
             kmeans.fit(df1)
             wcss4.append(1.5+kmeans.inertia_)

         fig1,=plt.plot(range(1,10),wcss1, label='WCSS 1')
         fig2,=plt.plot(range(1,10),wcss2, label='WCSS 2')
         fig3,=plt.plot(range(1,10),wcss3, label='WCSS 3')
         fig4,=plt.plot(range(1,10),wcss4, label='WCSS 4')

         plt.title('Kümeler İçi Kareler Toplamı Yöntemi')
         plt.xlabel('küme sayısı')
         plt.ylabel('wcss')
         plt.legend(handles=[fig1,fig2,fig3,fig4])
         plt.show()
```



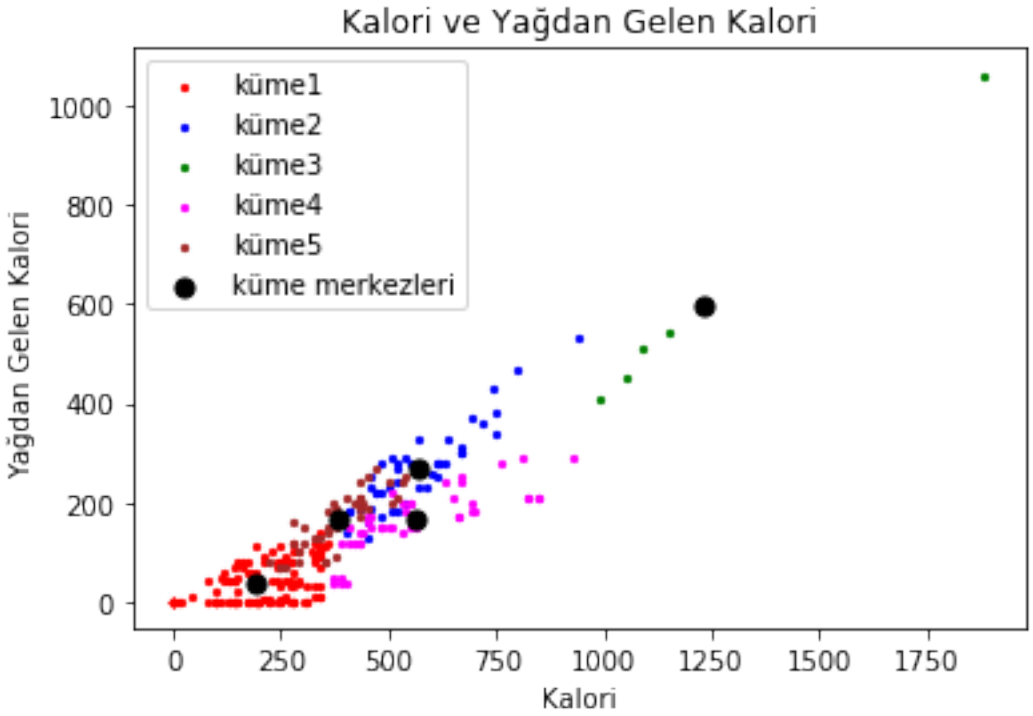
Yukarıdaki grafikte farklı max iter ve random state değerleri kullanarak dört tane WCSS değeri hesaplanmış ve grafikle çizdirilmiştir. Grafikte göründüğü gibi farklı değerlerin alınması hiçbir şeyi değiştirmemiştir ve bu değerlerin küme sayısını seçmeye herhangi bir etkisi bulunmamıştır.

0.3.1 K Means Modeli

Kullandığım veri seti olan df1 veri setinde kullanabileceğim birçok kolon var fakat hepsine k-means uygulamayacağım. Seçtiğim beş farklı ikiliye k-means uygulayarak açıklamaya çalışacağım. İlk olarak K means modeline Kalori ve Yağdan Gelen Kalori kolonlarına uyguluyorum. Dirsek metodunun sonucunda küme sayısı olarak 5 belirlemiştik ve modelde n_clusters=5 olarak yerleştirdik.

```
In [51]: kmeans = KMeans(n_clusters = 5, init='k-means++', max_iter=300, n_init=10, random_state
         y_kmeans = kmeans.fit_predict(X)

         plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 5, c = 'red', label = 'küme1'
         plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 5, c = 'blue', label = 'küme2
         plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 5, c = 'green', label = 'küme
         plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 5, c = 'magenta', label = 'kü
         plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 5, c = 'brown', label = 'küme

         plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=50, c='black'
         plt.title('Kalori ve Yağdan Gelen Kalori')
         plt.xlabel('Kalori')
         plt.ylabel('Yağdan Gelen Kalori ')
```

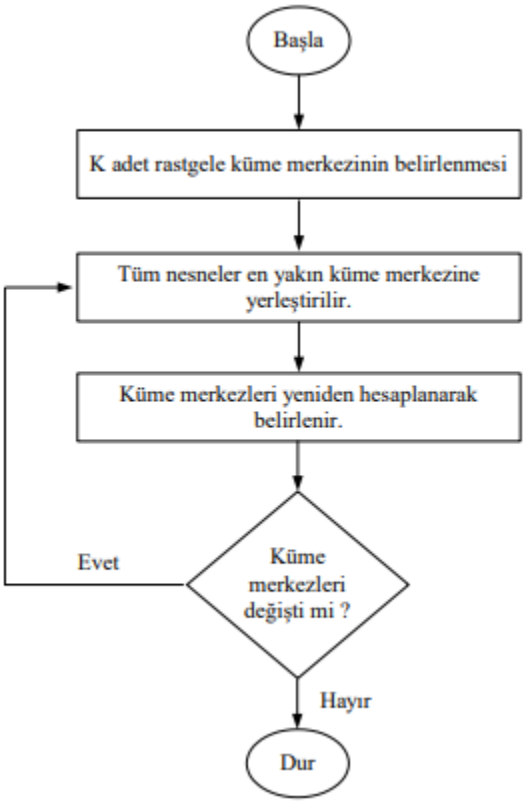
```
plt.legend()
plt.show()
```



Model Açıklaması K means sonuçlarına göre Kalori ve Yağdan Gelen Kalorine göre kümeleme yapıldığında 4 küme arasındaki mesafe birbirinden uzakken pembe renkli kümelemenin olduğu yerdeki oluşan küme, mavi ve kahverengi renkle oluşan kümelerin küme merkezlerine çok yakındır ve bu modelin istenilen kadar doğru sonuç veremeyeceğini gösterir. Grafiğe bakarak kalori miktarının artmasıyla yağdan gelen kalori miktarının artması bekleyebiliriz.

0.3.2 Model Akışı

```
In [34]: display(Image(filename='Desktop/sedaseda.png', embed=True))
```



Ne anlama geliyor? max_iter algoritmanın son haline gelmesi için en fazla kaç iterasyon yapabileceğini belirler.

Varsaylan 300 dür. Fakat İkinci adımda 500 alarak tekrar çalıştrdığımızda aynı sonuca ulaştığımızı görürüz.

n_init ise küme merkezi başlangıç noktasının kaç farklı noktadan başlayabileceğini belirler.

İkinci adımda ise 20 alarak herhangi bir değişikliğin olmadığı görülmüştür.

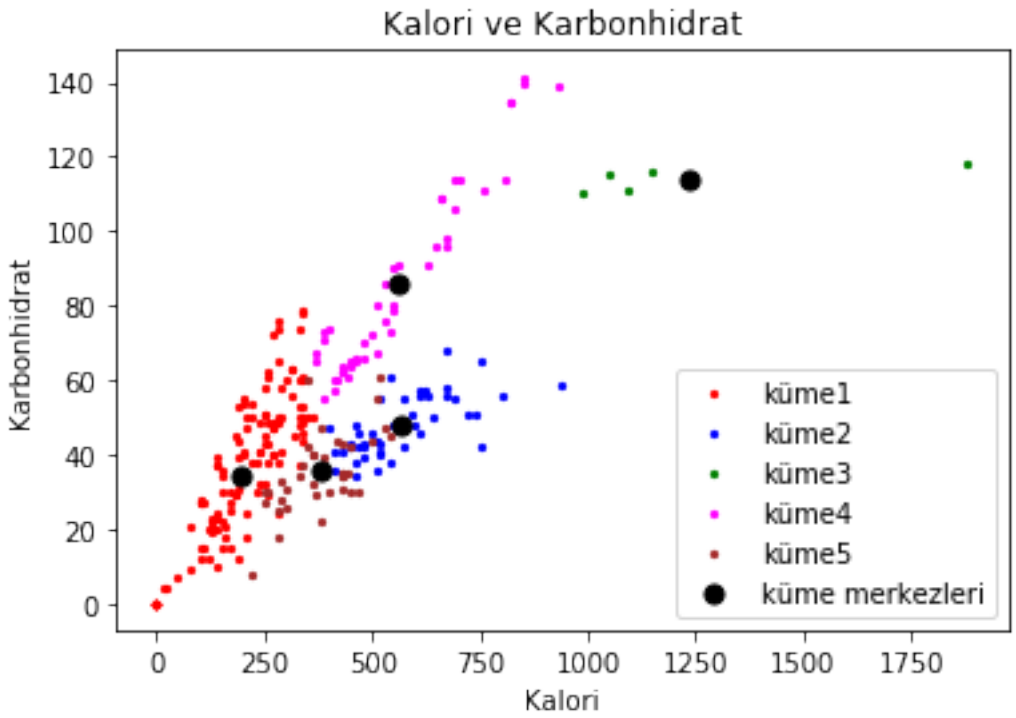
random_state ise bu işlemleri uygulayan herkesin aynı sonuçları elde etmesini sağlar.

kmeans++ parametresi ise rastgele başlangıç noktası tuzağından kurtararak iyi başlangıç noktaları seçilmesini sağlar.

fit_predict: Küme Merkezlerini hesaplar ve her bir veri için küme indeksini tahmin eder.

0.3.3 Kalori ve Karbonhidrat

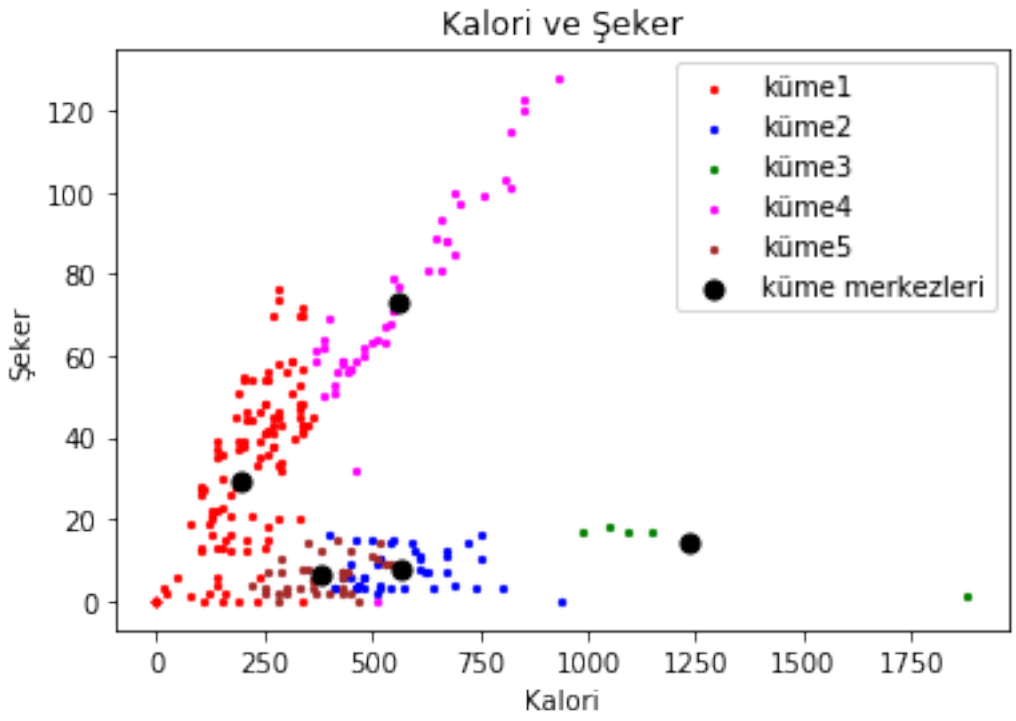
```
In [54]: kmeans = KMeans(n_clusters = 5, init='k-means++', max_iter=300, n_init=10, random_state
         y_kmeans = kmeans.fit_predict(X)
         plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 4], s = 5, c = 'red', label = 'küme1'
         plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 4], s = 5, c = 'blue', label = 'küme2
         plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 4], s = 5, c = 'green', label = 'küme
         plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 4], s = 5, c = 'magenta', label = 'kü
         plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 4], s = 5, c = 'brown', label = 'küme

         plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,4], s=50, c='black'
         plt.title('Kalori ve Karbonhidrat')
         plt.xlabel('Kalori')
         plt.ylabel('Karbonhidrat')
         plt.legend()
         plt.show()
```



Kalori ve Karbonhidrat değerlerime k-means uyguladığımda 5 farklı kümeye veriler dağılmıştır. Küme 2 ve Küme 5'in küme merkezlerinin birbirine çok yakın olması istenilen bir durum değildir. Kalori değeri 500'e yakın yerlerde Karbonhidrat değerlerinin 40 ve 80'e yakın olduğunu iki küme merkezinden görülmektedir. Kalori ve Karbonhidrat arasında tahmin yapıldığında hata payının artacağı gözükmektedir. Elimdeki veri setine Kalori ve Karbonhidrat değerleri için k-means uygulamanın doğru bir yöntem olmadığı gözükmektedir.

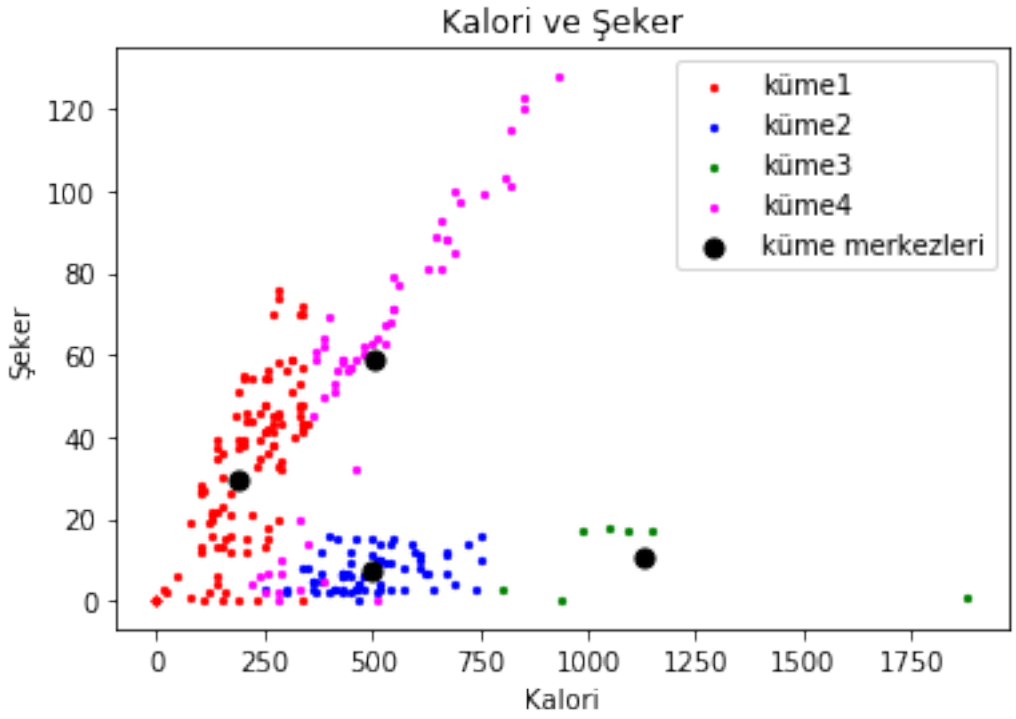
0.3.4 Kalori ve Şeker

```
In [47]: kmeans = KMeans(n_clusters = 5, init='k-means++', max_iter=300, n_init=10, random_state
         y_kmeans = kmeans.fit_predict(X)
         plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 6], s = 5, c = 'red', label = 'küme1'
         plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 6], s = 5, c = 'blue', label = 'küme2
         plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 6], s = 5, c = 'green', label = 'küme
         plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 6], s = 5, c = 'magenta', label = 'kü
         plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 6], s = 5, c = 'brown', label = 'küme

         plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,6], s=50, c='black'
         plt.title('Kalori ve Şeker')
         plt.xlabel('Kalori')
         plt.ylabel('Şeker')
         plt.legend()
         plt.show()
```



Küme 2 ve küme 5 in küme merkezleri birbirlerine çok yakındır. Mavi ve kahverengi noktalar iç içe geçmiştir. Homojen bir dağılım olmadığı ve beş kümenin bu model için uygun olmayacağı gözükmektedir. Bu nedenle bu modeli bir de küme sayısını 4 alıp uyguluyorum.

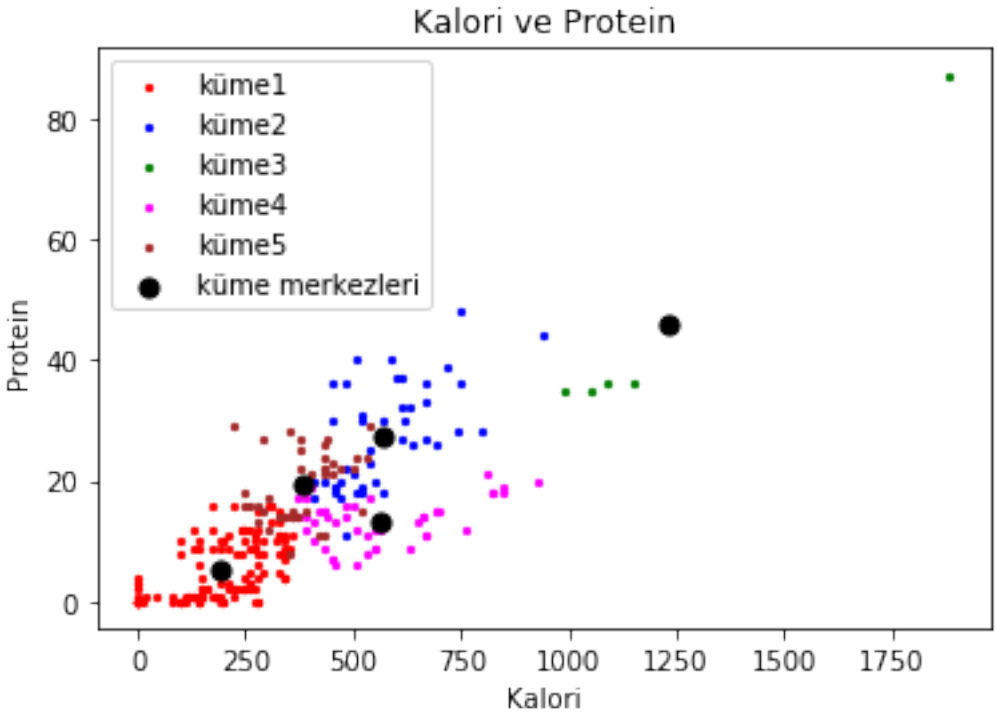
```
In [48]: kmeans = KMeans(n_clusters = 4, init='k-means++', max_iter=300, n_init=10, random_state
         y_kmeans = kmeans.fit_predict(X)
         plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 6], s = 5, c = 'red', label = 'küme1'
         plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 6], s = 5, c = 'blue', label = 'küme2
         plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 6], s = 5, c = 'green', label = 'küme
         plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 6], s = 5, c = 'magenta', label = 'kü


         plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,6], s=50, c='black'
         plt.title('Kalori ve Şeker')
         plt.xlabel('Kalori')
         plt.ylabel('Şeker')
         plt.legend()
         plt.show()
```



Küme sayısı 4 olduğunda kümelemenin daha optimal olduğu gözükmektedir. Küme 3 deki problem ise elimizdeki verilerin maximum ve minimum değerleri arasındaki farktan kaynaklanmaktadır. Verilerin içeriği describe metodunu kullanarak daha önce açıklamıştım.

0.3.5 Kalori ve Protein

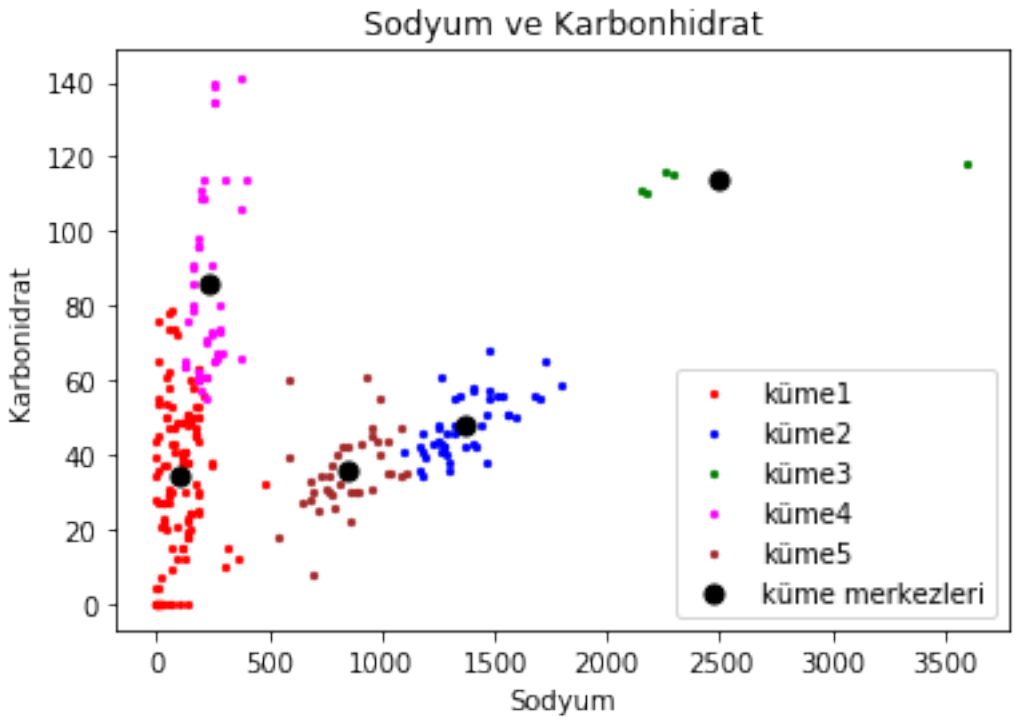
```
In [56]: kmeans = KMeans(n_clusters = 5, init='k-means++', max_iter=300, n_init=10, random_state
         y_kmeans = kmeans.fit_predict(X)
         plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 7], s = 5, c = 'red', label = 'küme1'
         plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 7], s = 5, c = 'blue', label = 'küme2
         plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 7], s = 5, c = 'green', label = 'küme
         plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 7], s = 5, c = 'magenta', label = 'kü
         plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 7], s = 5, c = 'brown', label = 'küme

         plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,7], s=50, c='black'
         plt.title('Kalori ve Protein')
         plt.xlabel('Kalori')
         plt.ylabel('Protein')
         plt.legend()
         plt.show()
```



Kalori ve Protein için incelediğimizde küme2, küme4, küme5 in küme merkezlerinin birbirlerine çok yakın olduğu gözükmektedir. Mavi, kahverengi ve pembe noktaların iç içe girmesinden dolayı bu model için k-meansin doğru bir model olmadığını söyleyebiliriz.

0.3.6 Sodyum ve Karbonhidrat

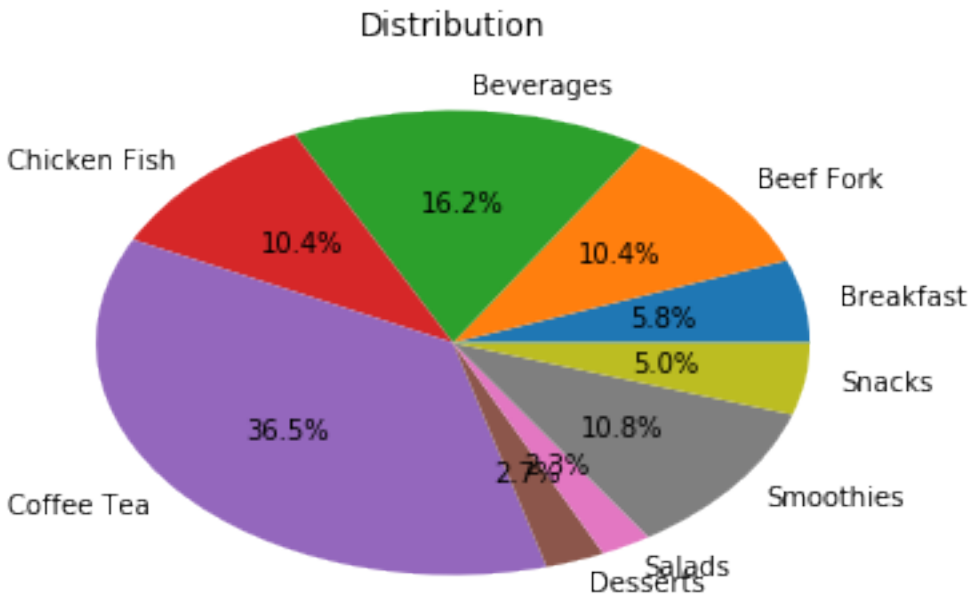
```
In [44]: kmeans = KMeans(n_clusters = 5, init='k-means++', max_iter=500, n_init=10, random_state
         y_kmeans = kmeans.fit_predict(X)
         plt.scatter(X[y_kmeans == 0, 3], X[y_kmeans == 0, 4], s = 5, c = 'red', label = 'küme1'
         plt.scatter(X[y_kmeans == 1, 3], X[y_kmeans == 1, 4], s = 5, c = 'blue', label = 'küme2
         plt.scatter(X[y_kmeans == 2, 3], X[y_kmeans == 2, 4], s = 5, c = 'green', label = 'küme
         plt.scatter(X[y_kmeans == 3, 3], X[y_kmeans == 3, 4], s = 5, c = 'magenta', label = 'kü
         plt.scatter(X[y_kmeans == 4, 3], X[y_kmeans == 4, 4], s = 5, c = 'brown', label = 'küme

         plt.scatter(kmeans.cluster_centers_[:,3], kmeans.cluster_centers_[:,4], s=50, c='black'
         plt.title('Sodyum ve Karbonhidrat')
         plt.xlabel('Sodyum')
         plt.ylabel('Karbonidrat')
         plt.legend()
         plt.show()
```



Sodyum ve Karbonhidrat arasındaki ilişkiye baktığımızda küme 1 ve küme 4 oluşumunda sodyumun pek bir etkisi olmadığı gözükmektedir. Küme 2 ve küme 5 oluşumda ise karbonhidrat miktarının çok bir etkisinin olmadığı görülmüştür. Yeşille gösterilen küme 3 ise elimizdeki veriyle Sodyum ve Karbonhidrat arasındaki ilişkiyi k-means uygulayarak bulmanın doğru bir yöntem olmayacağını göstermektedir.

Kategorik Verilerin Yorumlanması

```
In [36]: f = df.groupby(['Category']).size()
         label = ['Breakfast', 'Beef Fork', 'Beverages', 'Chicken Fish', 'Coffee Tea', 'Desserts
         plt.pie(f, labels = label, autopct = '%1.01f%%')
         plt.title('Distribution')
         plt.show()
```



Kategorik veriler df veri setimde Category kolonunda yer almaktadır. Bu sütunda menü türleri bulunmaktadır. Öncelikle hangi menülerin ne oranda veri setinde bulunduğunu grafikten görmekteyiz. %36.5 ile Coffee Tea veri setimde en fazla bulunan menüdür. Bu dağılımı görmek bizlere veri setini yorumlamakta yardımcı olmaktadır.

```
In [45]: from collections import Counter
         res = df['Category'].groupby(y_kmeans).agg(lambda x: Counter(x))
         for i in range(5):
             print(res[i])
```

```
Counter({'Coffee & Tea': 70, 'Beverages': 27, 'Smoothies & Shakes': 10, 'Desserts': 7, 'Snacks &
Counter({'Breakfast': 23, 'Chicken & Fish': 16, 'Beef & Pork': 5})
Counter({'Breakfast': 4, 'Chicken & Fish': 1})
Counter({'Coffee & Tea': 25, 'Smoothies & Shakes': 18, 'Breakfast': 1, 'Snacks & Sides': 1})
Counter({'Breakfast': 11, 'Beef & Pork': 9, 'Chicken & Fish': 9, 'Snacks & Sides': 6, 'Salads':
```

Bu adımda da her obek icindeki nesnelerin sayilmis hallerini görmekteyiz. Eğer ideal bi kümeleme olsaydı, tek tip menü türlerinin farklı kümelerde olması beklenirdi. İlk kümeye baktığımızda içinde birçok menü tipinin olduğunu görüyoruz. Fakat ilk Coffee & Tea ve Beverages ağırlıklı olduğundan içecekler kümesi olarak da yorumlayabiliriz. İkinci kümede Breakfast ve Chicken & Fish alanlarının daha fazla bulunduğunu görmekteyiz. Üçüncü küme eleman sayısı açısından diğer kümelere göre eşit dağılmamıştır. Dördüncü kümede ise Coffee & Tea ve Smoothies & Shakes ağır basmıştır. Beşinci kümede ise birçok menünün neredeyse eşit derecede bulunduğunu, kümenin içinde farklı tarzda menü tiplerinin bulunduğunu görmekteyiz. Bütün bu veriler ise modelin homojen bir dağılıma sahip olmadığını göstermektedir. Kullanılan veri setine k-means uygulamak doğru sonuçlar vermeyebilir.

0.4 Sınıflandırma

Veri madenciliği temel yöntemlerinden biri olan sınıflandırma bir tahmin modelidir. Veri setindeki değişkenlerden biri tahmin edilmek üzere seçilir. Seçilen bu değişkene sınıf adı verilir ve uygun bir algoritma ile o sınıfın diğer ortamlarda oluşabilecek değeri tahmin edilir. Bunun yanı sıra, farklı algoritmalarla sınıfın tahmin edilmesi için, kurallar, örüntüler ve diğer bilgiler çıkartılarak kullanılabilir özetler halinde kullanıcıya ya da araştırmacıya sunulur. Sınıflandırma modelinde kullanılan algoritmalar çeşitli sınıflara ayrılmaktadır. Bunlar; istatistiğe dayalı algoritmalar, Karar ağaçları, Yapay sinir ağları, Genetik algoritmalar ve mesafeye dayalı sınıflandırma algoritmalarıdır. Ben bu analizimde Sınıflandırma problemlerinde en çok kullanılan yöntemlerden olan Karar Ağacı'nı kulladım.

0.5 Karar Ağacı

Karar ağacı algoritması, veri madenciliği sınıflandırma algoritmalarından biridir. Karar ağaçları, veri madenciliğinde kuruluşlarının ucuz olması, yorumlanmalarının kolay olması, veri tabanı sistemleri ile kolayca entegre edilebilmeleri ve güvenilirliklerinin iyi olması nedenleri ile sınıflama modelleri içerisinde en yaygın kullanıma sahip tekniktir. İçinde çok sayıda kayıt bulunan veri setlerini küçük yapılar yardımıyla karar vermeyi sağlar. Karar ağaçları, adından da anlaşılacağı gibi bir ağaç görünümünde, tahmin edici bir tekniktir. Karar düğümü, gerçekleştirilecek testi belirtir. Bu testin sonucu ağacın veri kaybetmeden dallara ayrılmasına neden olur. Her düğümde test ve dallara ayrılma işlemleri ardışık olarak gerçekleşir ve bu ayrılma işlemi üst seviyedeki ayrımlara bağımlıdır. Ağacın her bir dalı sınıflama işlemini tamamlamayabilir.

Karar Ağacı Matematiksel Yorum Tanım 1:D =(t1,t2,t3,. . . ,tn) bir veri setidir. her ti , ti = den ibarettir ve veri seti (A1, A2 ... An) alan isimlerinden oluşmaktadır.

C = (C1 ... Cn)verilen sınıflardır.

Karar ağacı algoritmaları yukarıdaki tanım çerçevesinde verilen D veri setini i kadar C sınıfına ayırır.

Oluşacak karar ağacı her bir dala ayrılma yeri Aialanıyla isimlendirilmiş ve her yaprağın bir sınıf olduğu bir ağaçtır.

Karar Ağacı Pseudo Code D: Öğrenme veritabanı
T: Kurulacak ağaç
T = 0 // baG̣langıçta ağaç boG̣ küme
Dallara ayırma kriterlerini belirle
T = kök düğümü belirle
T = dallara ayrılma kurallarına göre kök düğümü dallara ayır; herbir dal için
do
Bu düğüme gelecek değişkeni belirle

```
if (durma koşuluna ulaşıldı)
   Yaprak ekle ve dur
           else
           loop
```

0.6 Gini Algoritması

Karar ağaçlarıyla alt düğümler oluiturulur. Alt düğümlerin oluşturulması ise alt düğümlerin homojenliğini arttırır. Bir düğümü iki veya daha fazla alt düğümde bölmeye karar vermek için birden fazla algoritma kullanır. Bu analizde Gini algoritmasını kullanarak karar vermeye çalştık. CART metodu olarak tanımlanır.

Sınıflandırma ve regresyon ağaçları (CART) metodu Breiman vd. (1984) tarafından önerilmiştir. CART algoritması ikili bölünme esasına dayanarak yeni bölünmeler olduğu sürece devam etmektedir. Her bir değişken için, CART algoritması iki ayrı ve birbirini tamamlayan kümeler içerisindeki tüm olası aday bölünmelerle ilgilenmektedir.

İlk hangi nitelikten bölüneceği ve bölünme değeri gini indeks değerine bakılarak hesaplanır. Gini indeks değeri veri setindeki varlıkların oranı olarak tanımlanabilir. İki varlığın gini değeri aynı çıkarsa sonuç dağılımları aynı demektir. Eğer veri setindeki bir nitelikte 3 veya daha fazla seçenek bulunuyorsa ve ikiden fazla bölünmeye izin verilmediği için birbirine yakın seçenekler gruplandırılır.

Gini algoritması bütün veri setlerinde başarılı sonuç vermez ve bazı durumlarda ağaç sonlandırılamayabilir. Bu durumda aşağıdaki durma kurallarından biri veya bir kaçı uygulanabilir.

- Eğer düğüm saf hale gelmişse
- Bütün düğümler saf hale gelmişse
- Ağaç maksimum derinliğe ulaşmışsa
- Minimum düğüm boyutuna ulaşılmışsa

0.6.1 Gini Değeri Hesaplanması

$$GiniSol : Gini_l = 1 - \sum_{k=1}^{k} (\frac{L_i}{T_l})^2$$

$$GiniSa : Gini_r = 1 - \sum_{k=1}^{k} (\frac{R_i}{T_r})^2$$

k: Sınıfların sayısı
T: Bir düğümdeki örnek sayısı
T_l: Sol koldaki örneklerin sayısı
T_r:Sağ koldaki örneklerin sayısı
L_i: Sol kolda i kategorisindeki örneklerin sayısı
R_i: Sağ kolda i kategorisindeki örneklerin sayısı

$$Gini_j = (|T_l|Gini_l + |T_r|Gini_r)\frac{1}{n}$$

Her bir nitelik için hesaplanan Gini değerleri arasından en küçük olanı seçilir ve bölünme bu nitelik üzerinden gerçekleşir. Kalan veri seti üzerinde yukarıdaki bahsedilen adımlar tekrar uygulanır ve diğer bölünme hesaplanır.

Kümelemede kullandığım veri setini kullanarak şimdi de sınıflandırma yapmaya çalışağım. İlk olarak karar ağacında kullanacağom kütüphaneleri import ettim. Karar ağacını grafik şeklinde göstermeme yarayan graphizi kullandım.

```
In [65]: from sklearn import tree
         from sklearn.tree import DecisionTreeClassifier, export_graphviz
         from sklearn.model_selection import train_test_split

         import graphviz
         import pydotplus
         import io
         from scipy import misc

         %matplotlib inline
```

0.6.2 BİRİNCİ BÖLÜM

Hedef değişkeni Sugar seçildi.

Daha önce tanımladığım df1 veri setini kullanıldı.

Karar ağacı algoritmasının kullanıldığı sınıflandırma problemlerinde kullanılacak veri seti iki ana parçaya train ve test olarak ayrılmalıdır. Algoritma, eğitim verilerini(train) kullanarak model oluşturur. Oluşturulan bu model de test verisi üzerinde uygulanarak modelin problem çözümündeki başarısı hesaplanmaktadır.

```
In [66]: train, test = train_test_split(df1, test_size = 0.15)
```

İlk adım öğrenme basamağıdır. Öğrenme basamağında önceden bilinen bir eğitim verisi, model oluşturmak amacıyla sınıflama algoritması tarafından analiz edilir. İkinci adım ise sınıflama basamağıdır. Sınıflama basamağında test verisi, sınıflama kurallarının veya karar ağacının doğruluğunu belirlemek amacıyla kullanılır. Eğer doğruluk kabul edilebilir oranda ise, kurallar yeni verilerin sınıflanması amacıyla kullanılır.

Adımları uygulamak için öncelikle train ve test adında iki nesne oluşturuyoruz. Kullanmak istesiğimiz kolonların yer aldığı veri seti olan df1'in %15'ini test size olarak alıyoruz. Kalan %85 lik kısmı ise modelimizi kurmak için kullanıyoruz. 260 olan veri setinin 221'ini train için 39'unu ise test için ayrmış oldum.

```
In [67]: print("Training size: {}; Test size: {}".format(len(train), len(test)))

Training size: 221; Test size: 39
```

Scikit-learn kütüphanesi tree modülü DecisionTreeClassifier sınıfıyla c nesnesi ile modelimiz oluşturalım.

```
In [68]: c = DecisionTreeClassifier(min_samples_split=100)

In [69]: features =['Calories', 'Calories from Fat', 'Cholesterol', 'Sodium',
                    'Carbohydrates', 'Dietary Fiber', 'Protein',
                    'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
                    'Calcium (% Daily Value)', 'Iron (% Daily Value)']
```

Bağımsız değişkenler Sugar kolonu dışında df1 veri setindeki diğer kolonlardır. Hedef değişken olarak Sugar seçtiğim için, Sugar hariç diğer kolanları bir araya getiriyorum ve features adlı değişkende tutuyorum.

```
In [70]: x_train = train[features]
         y_train = train['Sugars']
```

Oluşturduğumuz train nesnesini eğitip, x_train ve y_train adında iki nesne oluşturuyoruz. Böylece eğitim verilerimiz oluşmuş oldu.

```
In [71]: x_test = test[features]
         y_test = test['Sugars']
```

Oluşturduğumuz test nesnesi de hedef değişken ve bağımsız değişkenlerle iki ayrı nesneye atıyoruz.

```
In [104]: x_test.head()

Out[104]:      Calories  Calories from Fat  Cholesterol  Sodium  Carbohydrates  \
           10        430                240           30    1080             34
           53        440                200           90    1110             35
           51        720                360          115    1470             51
           208        80                 40           15      65              9
           186       480                150           50     270             66
```

```
     Dietary Fiber  Protein  Vitamin A (% Daily Value)  \
10               2       11                          0
53               2       27                          6
51               4       39                          8
208              0        1                          4
186              1       16                         15

     Vitamin C (% Daily Value)  Calcium (% Daily Value)  Iron (% Daily Value)
10                           0                        6                    15
53                          10                       20                    20
51                          25                       30                    25
208                          0                        4                     0
186                          0                       50                     4
```

```
In [73]: dt = c.fit(x_train,y_train)
```

Karar ağacı modeli oluşturuldu ve eğitildi.

```
In [74]: y_pred = c.predict(x_test)
```

Test verileriyle tahmin yapıldı.

```
In [75]: y_pred
```

```
Out[75]: array([ 3,  3,  3,  0, 59, 59,  0,  0,  3,  0,  0,  3,  3,  0, 59,  3,  3,
                59,  0, 59,  0, 59,  3,  3,  0, 59,  3, 59,  0,  3, 59,  3,  0, 59,
                 0,  3, 59,  0, 59], dtype=int64)
```

```
In [76]: deneme=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
         deneme.head()
```

```
Out[76]:      Actual  Predicted
         10        2          3
         53        7          3
         51       14          3
         208       1          0
         186      60         59
```

Test ve tahmin verilerini gösterdik.

```
In [77]: from sklearn.metrics import accuracy_score
```

```
In [78]: score = accuracy_score(y_test, y_pred) * 100
```

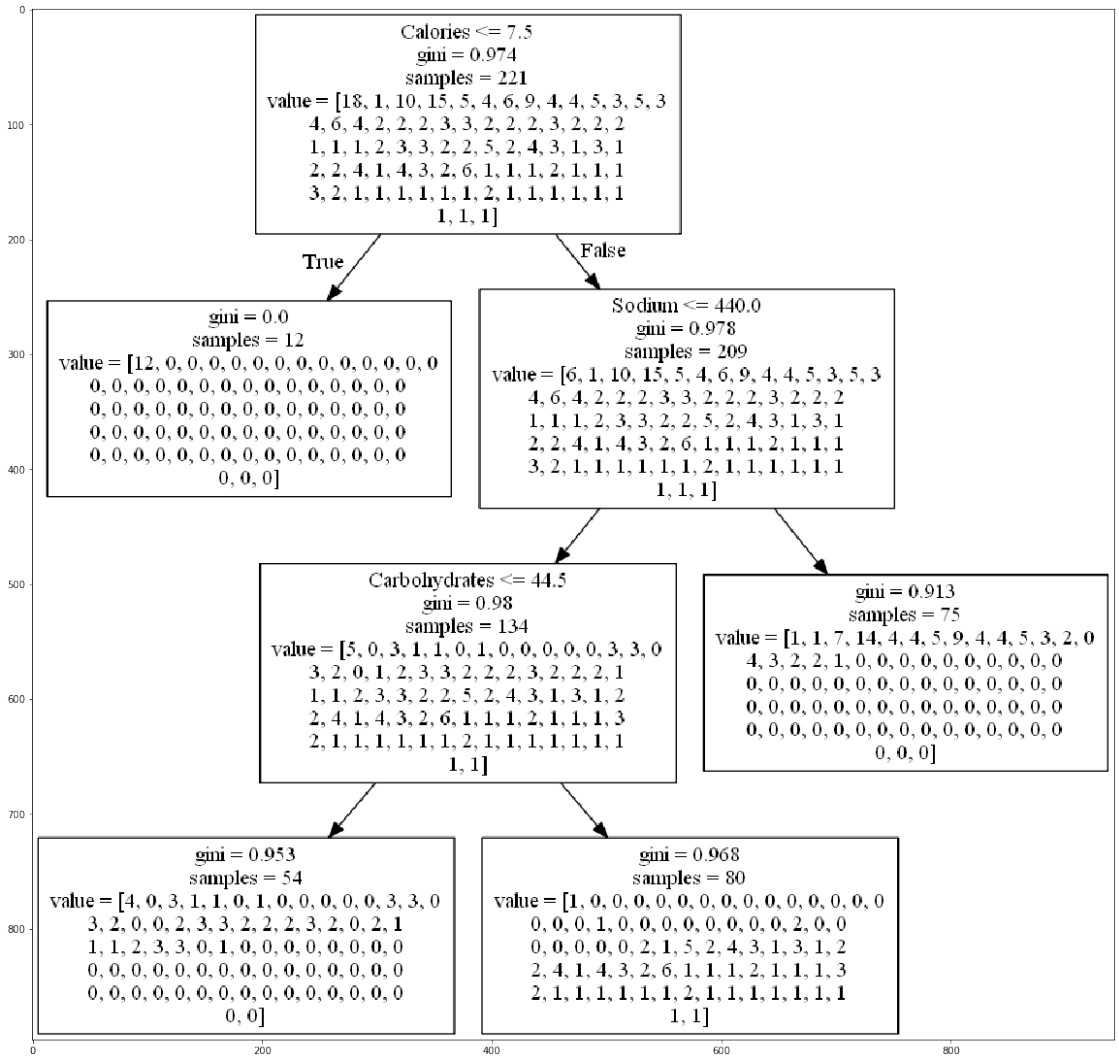
```
In [79]: print("Accuracy using decision tree: ", round(score,1), "%" )
```

```
Accuracy using decision tree:  17.9 %
```

```
In [80]: def show_tree(tree, features, path):
             f = io.StringIO()
             export_graphviz(tree, out_file = f, feature_names = features)
             pydotplus.graph_from_dot_data(f.getvalue()).write_png(path)
             img = misc.imread(path)
             plt.rcParams["figure.figsize"] = (20,20)
             plt.imshow(img)
```

Ağacı yorumlarsak eğer, ilk olarak nitelik Calories seçilmiş. Bu da bize Gini Calories değerininin en küçük gini değerine sahip olduğunu göstermektedir. Bu durumda kök düğümden itibaren bölünme Calories değeri üzerinden yapılmıştır. Birinci bölünme sonucu oluşan dallanmadan sonra gini adımları tekrar uygulanır ve ikinci bölünmenin hangi niteliğe göre olacağı hesaplanır. Bunun için öncelikle eğitim verisinden 12 tane örnek ve bunun kayıtları çıkarılır daha sonra kalan kayıt ve örneklerle ikinci bölünmenin hangi niteliğe göre dallanacağının kararı verilir. Gini adımları uygulandıktan sonra en küçük giniye sahip olan Sodium niteliği seçilmiştir. İkinci bölünme ise Sodium üzerinden yapılmıştır. Üçüncü bölünme için 75 örnek ve bunların kayıtları eğitim setinden çıkarılarak tekrar gini uygulanır. Üçüncü bölünmenin hangi nitelik üzerinde olacağına karar vermek için gini değeri en düşük olan seçilmelidir. Carbonhydrates ise gini değeri en küçük olan değerdir. Üçüncü bölünme bu nitelik üzerinden yapılmıştır. En son dallanma da nihai karara ulaşılmıştır.

```
In [81]: show_tree(dt, features, 'dec_tree_01.png')
```



0.6.3 İKİNCİ BÖLÜM

Bu bölümde hedef değişken ilk olarak Vitamin A'yı daha sonra Vitamin C'yi seçeceğiz. Karar ağacı adımlarını ve gini algoritmasını uyguladıktan sonra çıkan sonuçlarla karşılaştırma yapıp bu modelin hangi vitamin için daha doğru tahmin yaptığını karşılaştıracağız. Her iki adım için de test size 0.25 olarak alınmıştır.

Hedef değişkeni Vitamin A (% Daily Value) seçildi.

```
In [82]: train2, test2 = train_test_split(df1, test_size = 0.25)

         c2 = DecisionTreeClassifier(min_samples_split=100)

         features2 =['Calories', 'Calories from Fat', 'Cholesterol', 'Sodium',
                'Carbohydrates', 'Dietary Fiber', 'Sugars', 'Protein', 'Vitamin C (% Daily Value
                'Calcium (% Daily Value)', 'Iron (% Daily Value)']
```

```
x_train2 = train2[features2]
y_train2 = train2['Vitamin A (% Daily Value)']

x_test2 = test2[features2]
y_test2 = test2['Vitamin A (% Daily Value)']

dt2 = c2.fit(x_train2,y_train2)    # model kuruldu

y_pred2 = c2.predict(x_test2)      # test ile tahmin yapıldı

from sklearn.metrics import accuracy_score

score2 = accuracy_score(y_test2, y_pred2) * 100     # elde edilen pred ile karşılaştırma

print("Accuracy using decision tree: ", round(score2,1), "%" )      #skor yazdırıldı
```

```
Accuracy using decision tree:  43.1 %
```

Model kurulup test tahmini yapıldıktan sonra elde edilen skor %43.1 olarak hesaplanmıştır.

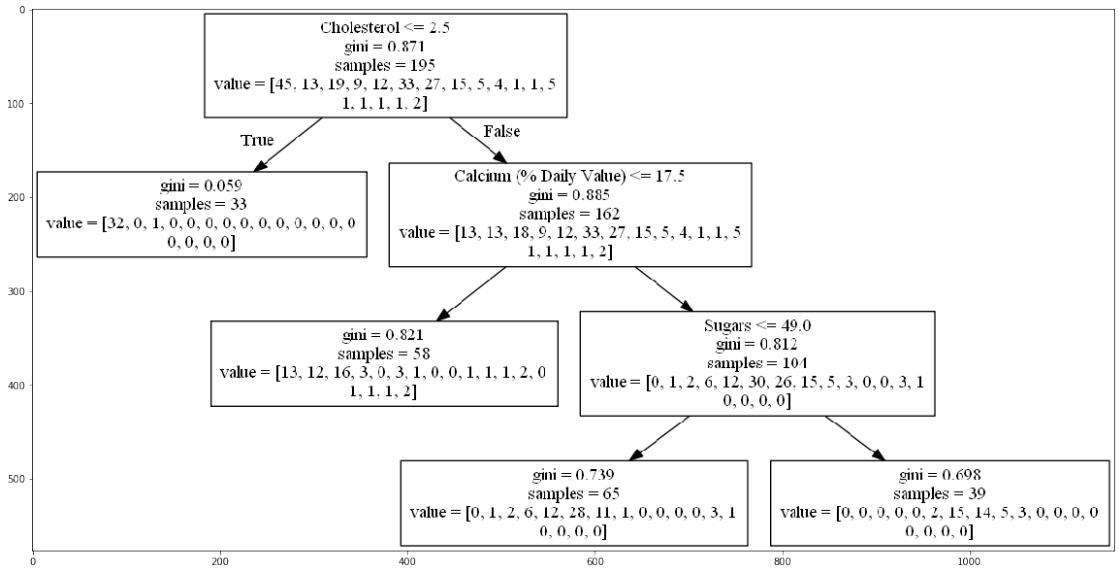
```
In [83]: y_pred2
```

```
Out[83]: array([ 4,  4, 10, 10,  4, 10,  4,  4,  4,  4, 10,  4,  0,  4, 15,  4,  4,
                10,  4, 15, 15, 10, 15, 15,  0,  0,  0, 10,  4,  0, 10,  4,  4, 10,
                 0,  4, 10, 15,  0, 10,  0,  4, 10, 10, 10, 10,  0,  0, 10, 10,  4,
                 4,  4, 15, 10, 15,  4,  4,  4,  4, 10, 10,  0, 15, 15], dtype=int64)
```

```
In [84]: print("Training size: {}; Test size: {}".format(len(train2), len(test2)))
```

```
Training size: 195; Test size: 65
```

```
In [85]: def show_tree(tree, features2, path):
             f2 = io.StringIO()
             export_graphviz(tree, out_file = f2, feature_names = features2)
             pydotplus.graph_from_dot_data(f2.getvalue()).write_png(path)
             img2 = misc.imread(path)
             plt.rcParams["figure.figsize"] = (20,20)
             plt.imshow(img2)

In [86]: show_tree(dt2, features2, 'dectr.png')
```

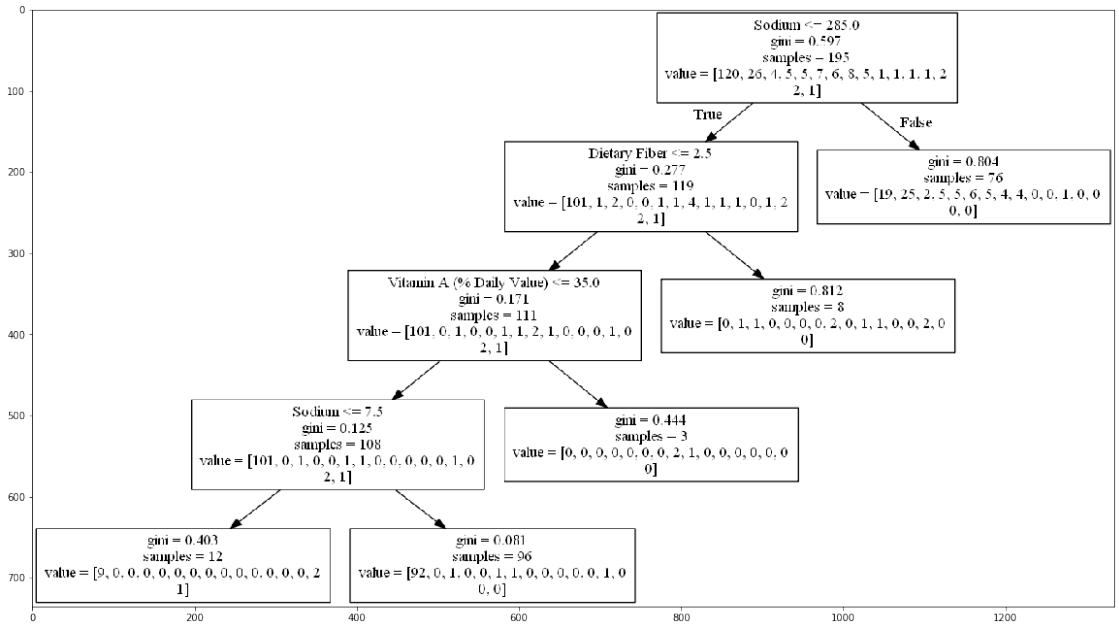


Daha sonra hedef değişkenim Vitamin C seçilerek işlem yaptırılmıştır.
Hedef değişkeni Vitamin C (% Daily Value) seçildi.

```
In [87]: train3, test3 = train_test_split(df1, test_size = 0.25)

         c3 = DecisionTreeClassifier(min_samples_split=100)

         features3 =['Calories', 'Calories from Fat', 'Cholesterol', 'Sodium',
                'Carbohydrates', 'Dietary Fiber', 'Sugars', 'Protein', 'Vitamin A (% Daily Value
                'Calcium (% Daily Value)', 'Iron (% Daily Value)']

         x_train3 = train3[features3]
         y_train3 = train3['Vitamin C (% Daily Value)']

         x_test3 = test3[features3]
         y_test3 = test3['Vitamin C (% Daily Value)']

         dt3 = c3.fit(x_train3,y_train3)
         y_pred3 = c3.predict(x_test3)

         from sklearn.metrics import accuracy_score

         score3 = accuracy_score(y_test3, y_pred3) * 100

         print("Accuracy using decision tree: ", round(score3,1), "%" )

Accuracy using decision tree:  63.1 %
```

Model kurulup test tahmini yapıldıktan sonra elde edilen skor %63.1 olarak hesaplanmıştır.

```
In [88]: y_pred3

Out[88]: array([25,  0,  0,  0,  0,  2, 25,  0,  0,  0,  0,  0,  2,  0,  2,  0,  2,
                2,  0,  0,  2,  0,  0,  2,  0,  0,  2,  2,  0,  0,  0,  0,  2,  0,
                2,  0,  2,  2,  0,  2,  0,  0,  0,  2,  0,  2,  2,  0,  0,  0,  2,
               25,  0,  0,  2,  0,  0,  0,  0,  2,  2,  0,  2,  0,  0], dtype=int64)

In [89]: def show_tree(tree, features3, path):
             f3 = io.StringIO()
             export_graphviz(tree, out_file = f3, feature_names = features3)
             pydotplus.graph_from_dot_data(f3.getvalue()).write_png(path)
             img3 = misc.imread(path)
             plt.rcParams["figure.figsize"] = (20,20)
             plt.imshow(img3)

In [90]: show_tree(dt3, features3, 'dectr3.png')
```



Her iki model karşılaştırıldığında Vitamin A'nın score değeri %43.1, Vitamin C'nin score değeri ise %63.1 olarak hesaplanmıştır.Bu iki skor değerine bakıldığında Vitamin A'nın tahmini Vitamin C'den az gözükmektedir. Fakat hangisini seçmem gerektiği konusunda sadece score puanı bana yardımcı olmayabilir. Vitamin A'nın dallanması Vitamin C'ye göre daha azdır. Vitamin C modeli dallanması fazla olduğundan ve her düğümde belli sayıda kayıtlar çıkarıldığından score puanının %63.1 gibi yüksek bir sonuö gelmesi beklenen bir durumdur. Dallanma sayısı daha az olduğundan bu modelin Vitamin A değerlerin, tahmin etmesi daha iyidir diye yorumlanabilir.

0.7 Sonuç

Bu projede öncelikle veri madencliğinin ne olduğu hakkında temel tanımlar vererek uygulanacak adımlardan bahsettim. Tanımda açıkladığım veri madenciliği adımlarını veri setime uygulayarak uygulamak istediğim yöntemleri Python ile gerçekleştirdim. İlk olarak kümeleme algoritmalarından olan k-means algoritmasını kullandım. Algoritmayı uygularken her bir adımda ilerleyişimden ve kodların ne anlama geldiklerinden bahsettim. Oluşan sonuçları grafikte gösterdim ve her bir grafiğin ne anlam ifade ettiğini, k-means kümelemenin veri setim için uygun olup olmadığını açıkladım. Daha sonra veri madenciliğinde bir tahmin modeli olan sınıflandırmayı açıkladım ve kullandığım karar ağacı hakkında bilgiler verdim. Karar ağacında gini algoritması kullanarak tahminleri yapmaya çalıştım. Karar ağacını grafikte gösterebilmek için graphiz kurulumunu yaptım. Daha sonra oluşan grafiği yorumlayıp bunları bölüm sonlarında belirttim.

0.7.1 Referanslar

Akkucuk, U. (2011). Veri Madenciliği: Kümeleme ve Sınıflama Algoritmaları. İstanbul: Yalın Yayıncılık.

Meltem, I. Ş. I. K., & ÇAMURCU, A. Y. (2007). K-Means, K-Medoids Ve Bulanık C-Means Algoritmalarının Uygulamalı Olarak Performanslarının Tespiti. İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi, 6(11), 31-45.

ADAK, M. F., & YURTAY, N. (2013). Gini Algoritmasını Kullanarak Karar Ağacı Oluşturmayı Sağlayan Bir Yazılımın Geliştirilmesi. INTERNATIONAL JOURNAL OF INFORMATICS TECHNOLOGIES, 6(3), 1-6.

Özkan, Y. (2008). Veri madenciliği yöntemleri. Papatya Yayıncılık Eğitim.

Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160, 3-24.

Özekes, S. (2003). Veri madenciliği modelleri ve uygulama alanları.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. Pattern recognition letters, 31(8), 651-666.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112). New York: springer.

SİLAHTAROĞLU, G., & ERGÜL, H. (2016). ŞEHİRLEŞME, MEKÂN–İNSAN ETKİLEŞİMİNİN BİREY ALGISINA YANSIMASI: BİR VERİ MADENCİLİĞİ ANALİZİ. Beykent Üniversitesi Fen ve Mühendislik Bilimleri Dergisi, 9(2).

GÜNER, Z. B. (2014). Veri Madenciliğinde Cart ve Lojistik Regresyon Analizinin Yeri: İlaç Provizyon Sistemi Verileri Üzerinde Örnek Bir Uygulama. Sosyal Güvence, (6).

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. IEEE transactions on pattern analysis and machine intelligence, 24(7), 881-892.

BİLGİN, M. Gerçek Veri Setlerinde Klasik Makine Öğrenmesi Yöntemlerinin Performans Analizi. Breast, 2(9), 683.

https://en.wikipedia.org/wiki/Decision_tree_learning

https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a