Unsupervised Learning

Denetimsiz Öğrenme (Unsupervised Learning)

Y	X1	X2	X3
1	10	92	3
0	12	87	4
0	34	34	6
1	12	12	3
1	34	45	1
0	45	12	7

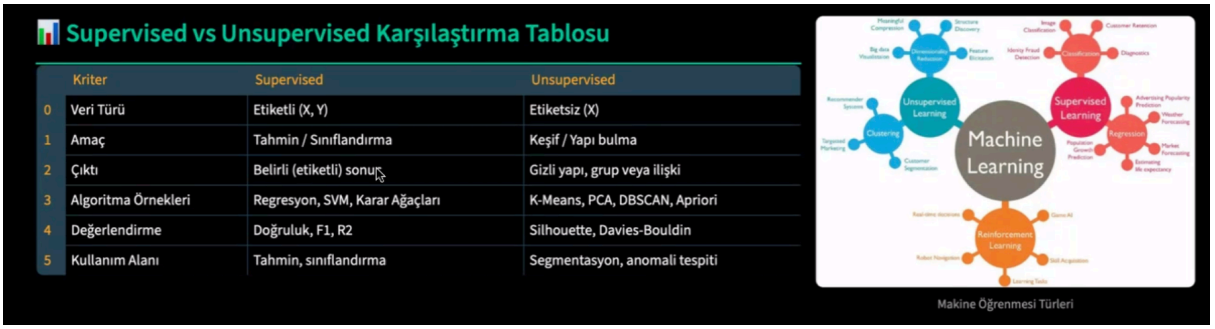
X1	X2	X3
10	92	3
12	87	4
34	34	6
12	12	3
34	45	1
45	12	7

Denetimsiz öğrenmenin başlıca özellikleri şunlardır:

- **Etiketlenmemiş Veri:** En belirgin ve temel özelliğidir. Eğitim veri setinde, çıktı değişkeni veya "doğru cevap" bulunmaz.
- **Keşif Odaklı:** Amacı, verilerdeki bilinmeyen yapıları, ilişkileri ve desenleri ortaya çıkarmaktır. Genellikle veri keşfi (Exploratory Data Analysis - EDA) ve ön işleme adımlarında kullanılır.
- **Yapı Bulma:** Algoritma, verilerdeki doğal grupları (kümeler) veya temel bileşenleri belirlemeye odaklanır.
- **İnsan Müdahalesi Azlığı:** Modelin eğitimi sırasında insan müdahalesi veya denetimi çok azdır ya da hiç yoktur. Algoritma kendi kendine öğrenir.
- **Uygulama Alanları:**
    - **Kümeleme (Clustering):** Müşteri segmentasyonu, belge sınıflandırması, görüntü analizi gibi alanlarda benzer veri noktalarını gruplamak için kullanılır (örn: K-Means, Hiyerarşik Kümeleme).
    - **Boyut İndirgeme (Dimensionality Reduction):** Yüksek boyutlu veri setlerindeki gürültüyü azaltmak, veri görselleştirmeyi kolaylaştırmak ve hesaplama yükünü düşürmek için kullanılır (örn: PCA, t-SNE).
    - **Birliktelik Kuralı Madenciliği (Association Rule Mining):** Veri setlerindeki öğeler arasında sıkça görülen ilişkileri bulur (örn: Pazar sepeti analizi - "X ürününü alanlar genellikle Y ürününü de alır").

- **Anomali Tespiti (Anomaly Detection):** Veri setindeki normalden sapmaları, aykırı değerleri veya dolandırıcılık gibi olağandışı durumları belirlemek için kullanılır.

- **Zorlukları:** Etiketli veri olmaması nedeniyle modelin başarısını ölçmek ve yorumlamak denetimli öğrenmeye göre daha karmaşık olabilir.

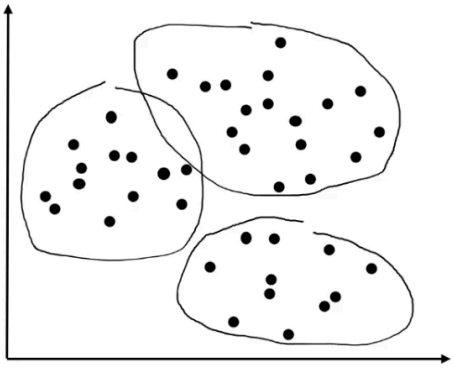


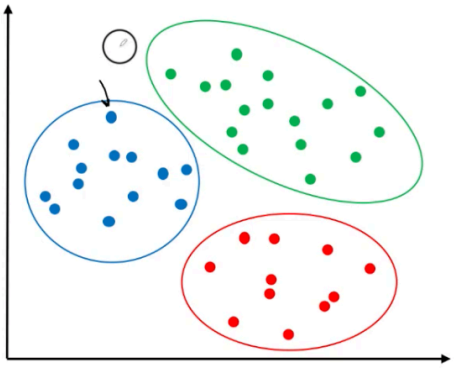
📊 Supervised vs Unsupervised Karşılaştırma Tablosu

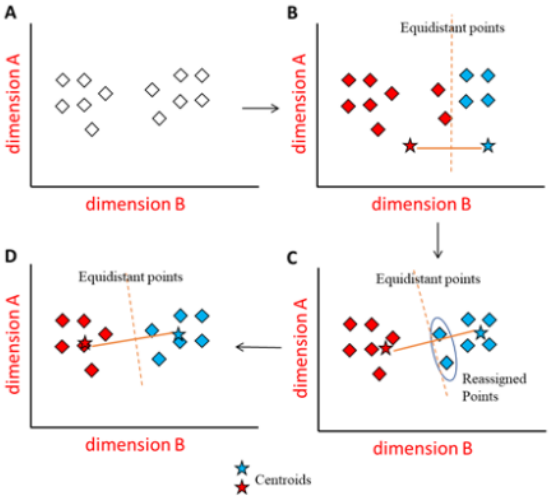
	Kriter	Supervised	Unsupervised
0	Veri Türü	Etiketli (X, Y)	Etiketsiz (X)
1	Amaç	Tahmin / Sınıflandırma	Keşif / Yapı bulma
2	Çıktı	Belirli (etiketli) sonuç	Gizli yapı, grup veya ilişki
3	Algoritma Örnekleri	Regresyon, SVM, Karar Ağaçları	K-Means, PCA, DBSCAN, Apriori
4	Değerlendirme	Doğruluk, F1, R2	Silhouette, Davies-Bouldin
5	Kullanım Alanı	Tahmin, sınıflandırma	Segmentasyon, anomali tespiti

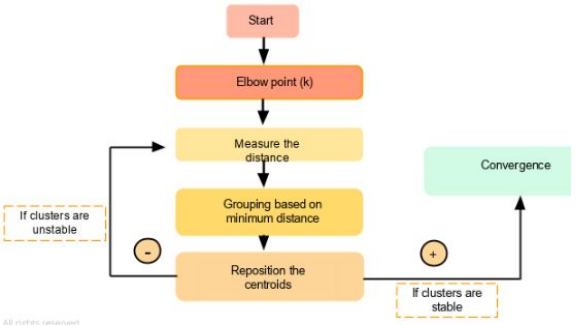
Makine Öğrenmesi Türleri

K-Ortalamalar (K-Means)


K-Means'ten Önce

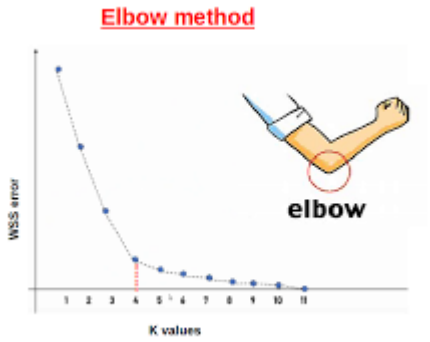

K-Means'ten Sonra





Step by Step: → Merkez bir gözlem birimidir.

- Adım 1: Küme sayısı belirlenir.
  - Algoritmanın ilk ve en kritik adımı, verilerin kaç kümeye ayrılacağını (K değeri) belirlemektir. Bu genellikle bir ön bilgiye dayalıdır veya **Dirsek Metodu (Elbow Method)** gibi tekniklerle belirlenir.
  - K değerinin doğru seçimi, kümeleme sonuçlarının anlamlılığı için hayati öneme sahiptir. Yanlış bir K değeri, verilerdeki doğal yapıları gözden kaçırmanıza veya yapay kümeler oluşturmanıza neden olabilir. Dirsek Metodu, farklı K değerleri için küme içi hata kareler toplamını (WCSS - Within-Cluster Sum of Squares) hesaplayarak bir grafik çizer ve hatanın düşüş hızının belirgin şekilde yavaşladığı "dirsek" noktasını optimal K olarak önerir.



- Adım 2: Rastgele k merkez seçilir.

    - Veri setinden rastgele K adet veri noktası seçilir ve bunlar her bir kümenin başlangıç merkezi (centroid) olarak atanır.
    - Başlangıç centroidlerinin seçimi algoritmanın sonucunu etkileyebilir. Rastgele seçimden kaynaklanan şans faktörünü azaltmak için **K-Means++ gibi daha akıllı başlangıç stratejileri kullanılır.** Bu stratejiler, centroidleri birbirinden mümkün olduğunca uzak seçerek daha iyi bir başlangıç noktası sağlar ve algoritmanın daha hızlı yakınsamasını (converge) teşvik eder.
- Adım 3: Her gözlem için k merkezlere uzaklıklar hesaplanır.
    - Veri setindeki her bir nokta, kendisine en yakın olan küme merkezine (centroidine) atanır. Yakınlık genellikle **Öklid mesafesi (Euclidean distance)** gibi bir uzaklık ölçütü kullanılarak hesaplanır.
- Adım 4: Her gözlem en yakin olduğu merkeze yani kümeye atanır.
    - Her küme için, o kümeye atanmış tüm veri noktalarının ortalaması (vektör ortalaması) alınarak yeni bir küme merkezi hesaplanır. Bu yeni merkez, o kümenin centroidi olur.
- Adim 5: Atama işlemlerinden sonra olusan kümeler için tekrar merkez hesaplamaları yapılır.
    - Adım 3 ve Adım 4, küme merkezleri artık önemli ölçüde değişmeyene veya maksimum iterasyon sayısına ulaşılana kadar tekrarlanır. Algoritma, centroidlerin pozisyonları stabil hale geldiğinde veya önceden belirlenmiş bir toleransın altına düştüğünde yakınsamış (converged) kabul edilir ve durur.
- Adim 6: Bu islem belirlenen bir iterasyon adedince tekrar edilir ve küme içi hata kareler toplamlarinin toplaminin (**total within-cluster variation)** minimum oldugu durumdaki
  gözlemlerin kümelenme yapısı nihai kümelenme olarak seçilir.

Application:

```
##################################
# K-Means
##################################

df = pd.read_csv("datasets/USArrests.csv", index_col=0)
```

```
df.head()
df.isnull().sum()
df.info()
df.describe().T

sc = MinMaxScaler((0, 1))
df = sc.fit_transform(df)
df[0:5]

kmeans = KMeans(n_clusters=4, random_state=17).fit(df)
kmeans.get_params()

kmeans.n_clusters
kmeans.cluster_centers_
kmeans.labels_
kmeans.inertia_
```

- **MinMaxScaler→ K-Means için kritik bir ön işleme adımı olan ölçeklendirmeyi gerçekleştirir.**
-
    - `sc = MinMaxScaler((0, 1))`: Bir `MinMaxScaler` nesnesi oluşturur. Bu ölçekleyici, verileri belirtilen aralığa (bu örnekte 0 ile 1 arasına) dönüştürecektir.
    - `df = sc.fit_transform(df)`: `MinMaxScaler`'ı DataFrame'e **uygular (`fit`)** ve ardından verileri bu aralığa **dönüştürür (`transform`)**. K-Means, uzaklık tabanlı bir algoritma olduğu için, farklı ölçeklerdeki değişkenlerin (örneğin, "tutuklanma sayısı" ile "şehirli nüfus oranı") kümeleme sürecini domine etmemesi için ölçeklendirme şarttır. Ölçekleme sonrası DataFrame artık bir NumPy dizisi haline gelir.
- `n_clusters=4`: Verilerin **4 kümeye ayrılmasını** istediğimizi belirtir. Bu K değeridir.
- `kmeans.cluster_centers_` özniteliği, modelin eğitimden sonra bulduğu **küme merkezlerinin (centroidlerin)** koordinatlarını (yani her bir kümenin ortalama değerlerini) bir NumPy dizisi olarak döndürür. Bu merkezler, her bir kümenin "temsilcisi" konumundadır.

- `kmeans.labels_` özniteliği, eğitim verisetindeki **her bir veri noktasının hangi kümeye atandığını** gösteren bir NumPy dizisi döndürür. Örneğin, ilk veri noktası 0. kümeye, ikinci veri noktası 1. kümeye vb. atanmış olabilir. Bu, veri noktalarını kümelerine göre gruplandırmak için kullanılır.
- `kmeans.inertia_` özniteliği, **küme içi kareler toplamı (Within-Cluster Sum of Squares - WCSS)** değerini döndürür. **Bu metrik, her bir veri noktasının atandığı kümenin merkezine olan uzaklığının karelerinin toplamıdır.** Daha düşük bir `inertia_` değeri, küme içi noktaların merkeze daha yakın olduğu ve dolayısıyla daha sıkı kümeler oluşturduğu anlamına gelir. `inertia_`, Dirsek Metodu gibi yöntemlerle optimal K değerini belirlemede kullanılan temel metriktir.

Özellik	**MinMaxScaler**	**StandardScaler**
Dönüşüm	Sabit bir aralığa (örn. [0, 1])	Ortalama 0, Standart Sapma 1
Aykırı Değerler	Çok duyarlı	Daha az duyarlı
Dağılım Şekli	Korur	Genellikle normal dağılıma yaklaştırır
Kullanım Alanı	**Sinir ağları, görüntü işleme, K-Means (uygunsa)**	Çoğu ML algoritması, aykırı değerler varsa tercih edilir

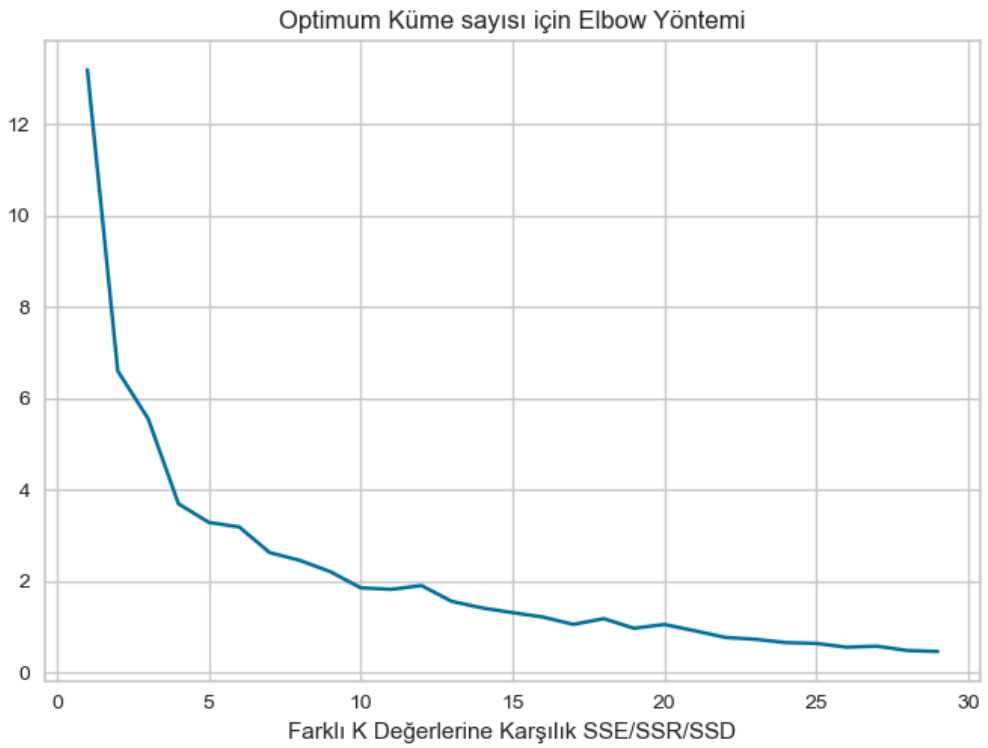
Optimum Küme Sayısı Belirleme

- Optimum K değerini bulmak, kümeleme analizinin en kritik adımlarından biridir. Çünkü çok az küme seçerseniz, verilerinizdeki doğal yapıları gözden kaçırırsınız. Çok fazla küme seçerseniz ise, anlamsız veya çok küçük gruplar oluşturmuş olursunuz.

  Burada iki farklı yaklaşımla K değeri belirleniyor: manuel implementasyon (Dirsek Metodu) ve otomatikleştirilmiş bir araç (`KElbowVisualizer`).

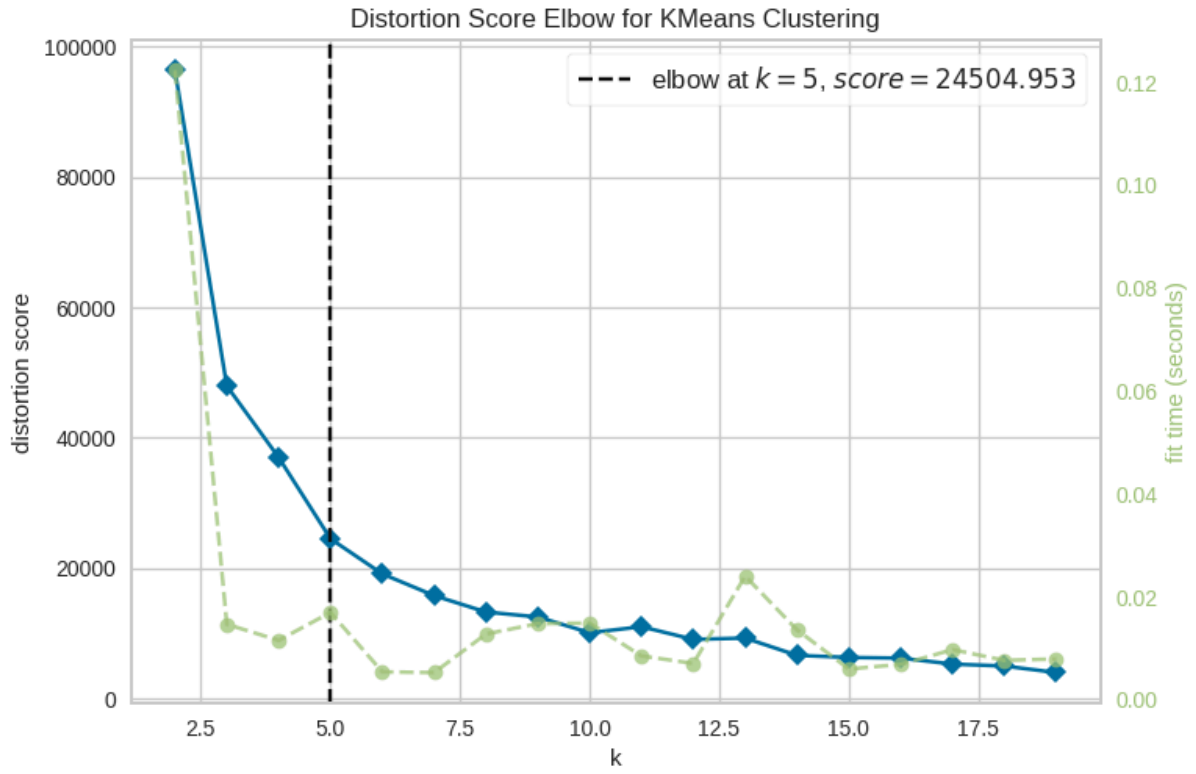
```
kmeans = KMeans()
ssd = []
K = range(1, 30)

for k in K:
    kmeans = KMeans(n_clusters=k).fit(df)
    ssd.append(kmeans.inertia_)
```

```
plt.plot(K, ssd, "bx-")
plt.xlabel("Farklı K Değerlerine Karşılık SSE/SSR/SSD")
plt.title("Optimum Küme sayısı için Elbow Yöntemi")
plt.show()
```



- `ssd = []`: **Kareler Hata Toplamı (Sum of Squared Distances - SSD)** veya **Küme İçi Kareler Toplamı (Within-Cluster Sum of Squares - WCSS)** olarak da bilinen `inertia_` değerlerini depolayacağımız boş bir liste oluşturulur. Amacımız bu değeri minimize etmektir.

```
kmeans = KMeans()
elbow = KElbowVisualizer(kmeans, k=(2, 20))
elbow.fit(df)
elbow.show()

elbow.elbow_value_
```



- `from yellowbrick.cluster import KElbowVisualizer`: `KElbowVisualizer` sınıfı içe aktarılır. Bu, Scikit-learn modellerini görselleştirmek için kullanılan bir kütüphanedir.
- `elbow = KElbowVisualizer(kmeans, k=(2, 20))`: Bir `KElbowVisualizer` nesnesi oluşturulur.
  - İlk argüman `kmeans`, kullanılacak modeli belirtir.
  - `k=(2, 20)` parametresi, K değerlerini 2'den 19'a kadar (20 dahil değil) deneyeceğimizi belirtir. Neden 1'den değil de 2'den başladığımızı düşünebilirsiniz: 1 küme, tüm veriyi tek bir gruba koymaktır ve bu genellikle pratik bir kümeleme çözümü değildir.
- • `elbow.elbow_value_`: Görselleştirici tarafından otomatik olarak belirlenen **optimum K değeri (dirsek noktası)** döndürülür. Bu, manuel grafik yorumlamasına göre daha nesnel bir sonuç sunar.→ 5

Final K-Means Model

```
###################################
# Final Cluster'ların Oluşturulması
###################################
```

```
kmeans = KMeans(n_clusters=elbow.elbow_value_).fit(df)

kmeans.n_clusters
kmeans.cluster_centers_
kmeans.labels_
df[0:5]

clusters_kmeans = kmeans.labels_

df = pd.read_csv("datasets/USArrests.csv", index_col=0)

df["cluster"] = clusters_kmeans

df.head()

df["cluster"] = df["cluster"] + 1

df[df["cluster"]==5]

df.groupby("cluster").agg(["count","mean","median"])

df.to_csv("clusters.csv")
```

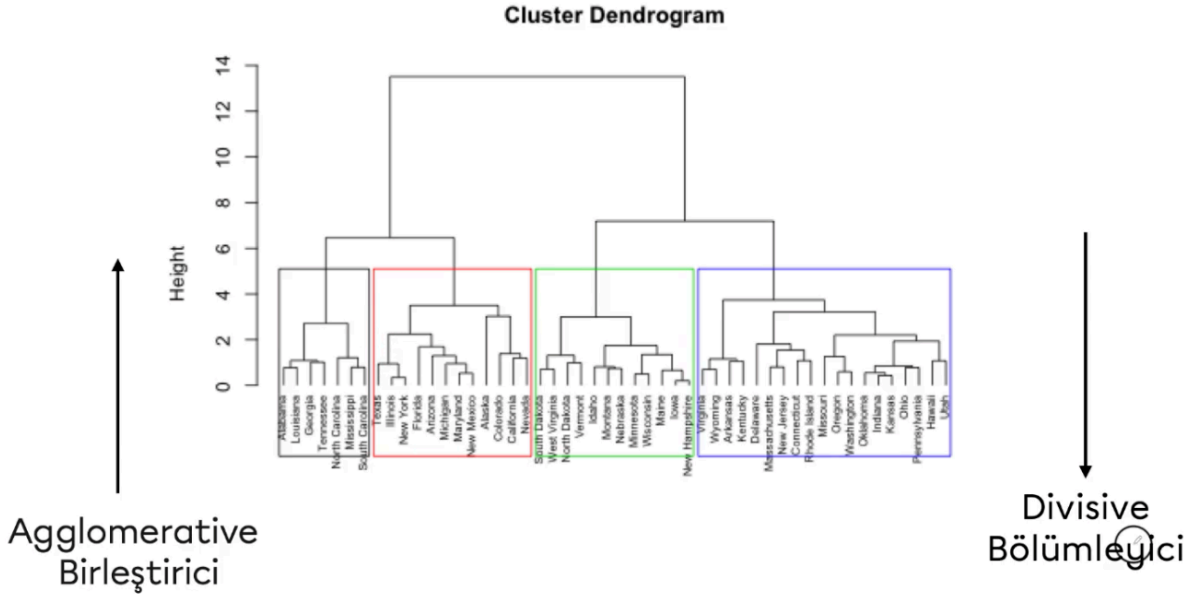
	Murder			Assault			UrbanPop			Rape		
	count	mean	median	count	mean	median	count	mean	median	count	mean	median
cluster												
1	14	8.214286	7.65	14	173.285714	167.5	14	70.642857	69.0	14	22.842857	23.10
2	10	2.950000	2.40	10	62.700000	56.5	10	53.900000	53.5	10	11.510000	11.00
3	12	11.766667	11.75	12	257.916667	254.5	12	68.416667	71.0	12	28.933333	25.05
4	10	5.590000	6.00	10	112.400000	111.5	10	65.600000	65.5	10	17.270000	16.45
5	4	11.950000	12.15	4	316.500000	317.5	4	68.000000	73.5	4	26.700000	29.40

	Murder	Assault	UrbanPop	Rape	cluster
Alabama	13.2	236	58	21.2	2
Alaska	10.0	263	48	44.5	2
Arizona	8.1	294	80	31.0	4
Arkansas	8.8	190	50	19.5	0
California	9.0	276	91	40.6	2

- kmeans = KMeans(n_clusters=elbow.elbow_value_).fit(df)
  - Bu satır, önceki adımda `KElbowVisualizer` tarafından belirlenen **optimum küme sayısını (`elbow.elbow_value_`)** kullanarak nihai K-Means modelini oluşturur ve eğitir.
    - `n_clusters=elbow.elbow_value_` : Otomatik olarak tespit edilen en iyi K değeri kullanılır. Bu, manuel grafik yorumlamanın getirdiği öznel kararları ortadan kaldırır.
    - `random_state=17` : Modelin tekrarlanabilirliğini sağlamak için `random_state` yine belirtilir.
    - `.fit(df)` : Model, ölçeklendirilmiş `df` verisi üzerinde eğitilir.

Hiyerarşik Kümeleme Analizi (Hierarchical Cluster Analysis

- Hiyerarşik kümeleme, her veri noktasını ayrı bir küme olarak ele alırken, K-ortalamalar belirli sayıda merkez nokta kullanır.



- Hiyerarşik Kümeleme Analizi (HCA), adından da anlaşılacağı gibi, verileriniz arasında bir **hiyerarşi** veya **ağaç yapısı** oluşturarak benzer verileri gruplar. Bunu bir aile ağacının oluşumu gibi düşünebilirsiniz. İlk başta herkes ayrı bireylerdir, sonra benzerliklerine göre gruplar (aileler) oluşmaya başlar, bu aileler daha büyük aile ağaçları içinde birleşir ve böyle devam eder.
- HCA'nın temel farkı, K-Means gibi başta belirli bir küme sayısı (K) söylemenize gerek olmamasıdır. Algoritma kendi kendine bir hiyerarşi oluşturur ve siz bu hiyerarşiye bakarak en uygun gördüğünüz noktadan "keserek" kümelerinizi belirlersiniz.

Hiyerarşik kümelemenin iki ana yolu vardır:

c

Bu, en yaygın kullanılan yöntemdir ve "alttan yukarı" bir yaklaşımdır. Aynen bir aile ağacının oluşumu gibi:

- **Adım 1: Her Nokta Bir Küme:** Başlangıçta veri setindeki her bir veri noktası (örneğin her bir insan veya her bir eyalet) kendi başına ayrı bir küme olarak kabul edilir. Diyelim ki 100 veri noktanız var, başlangıçta 100 kümeniz var demektir.
- **Adım 2: En Benzer İkililer Birleşir:** Algoritma, birbirine en yakın (en benzer) iki kümeyi (başlangıçta iki veri noktasını) bulur ve bunları tek bir yeni kümede birleştirir.

- **Adım 3: Tekrar ve Tekrar:** Bu birleştirme adımı, tüm veri noktaları tek bir büyük kümede toplanana kadar veya önceden belirlenmiş bir kriter (mesafe eşiği gibi) karşılanana kadar tekrarlanır. Her birleşmede, küme sayısı bir azalır.
- **Sonuç: Dendrogram:** Bu birleşme süreci, bir **dendrogram** adı verilen bir ağaç diyagramıyla görselleştirilir. Dendrogram, hangi noktaların veya kümelerin hangi seviyede birleştiğini, yani ne kadar benzer olduklarını gösterir.

2. Bölücü Yaklaşım (Divisive - Üstten Aşağı)

Bu yöntem ise "üstten aşağı" bir yaklaşımdır:

- **Adım 1: Her Şey Bir Küme:** Başlangıçta tüm veri noktaları tek bir büyük küme olarak kabul edilir.
- **Adım 2: En Az Benzer İkilere Bölünme:** Algoritma, bu büyük kümeyi, en az benzer iki alt kümeye ayırır.
- **Adım 3: Tekrar ve Tekrar:** Bu bölme işlemi, her veri noktası kendi başına ayrı bir küme olana kadar tekrarlanır.

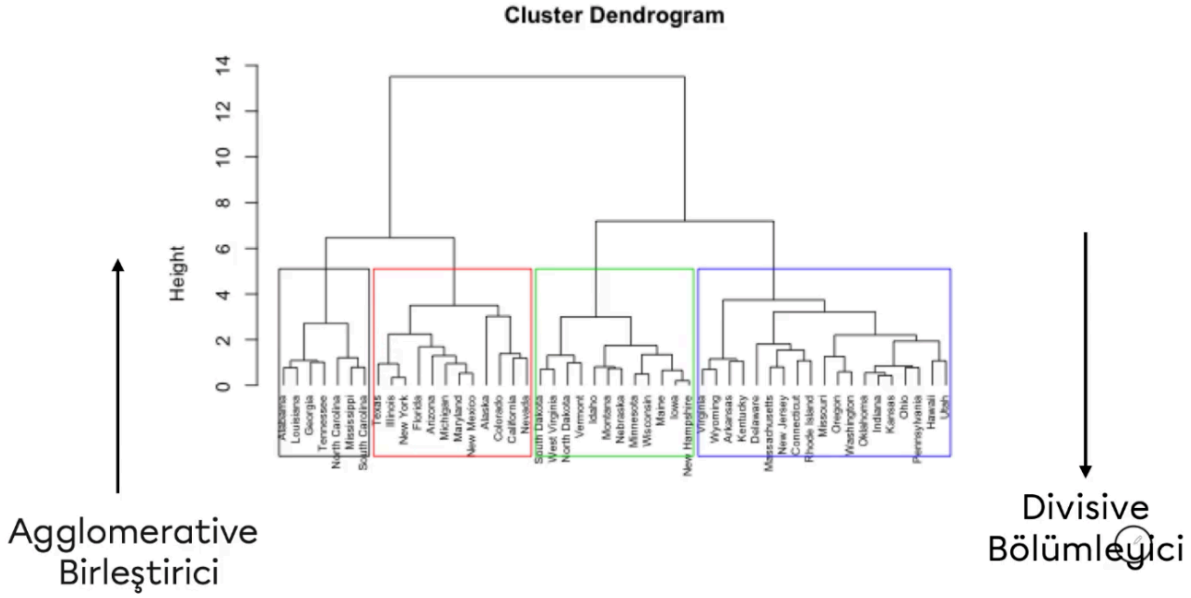
Anahtar Kavramlar

- **Mesafe Metriği:** "Benzerlik" veya "yakınlık" derken neyi kastettiğimizi belirleyen matematiksel ölçümdür. En yaygın olanı **Öklid** mesafesidir, ancak **Manhattan** mesafesi veya **kosinüs** benzerliği gibi başka ölçümler de kullanılabilir.
- **Bağlantı Kriteri (Linkage Criterion):** İki kümenin birbirine ne kadar yakın olduğunu belirlemek için kullanılır. İki kümeyi birleştirirken hangi noktalar arasındaki mesafeye bakacağımızı söyler:
    - **Tek Bağlantı (Single Linkage):** İki küme arasındaki en kısa mesafeye sahip iki nokta arasındaki uzaklığı kullanır. "Komşu" noktalar yüzünden bazen kümeleri zincirleme eğilimi gösterir.
    - **Tam Bağlantı (Complete Linkage):** İki küme arasındaki en uzak mesafeye sahip iki nokta arasındaki uzaklığı kullanır. Kümelerin daha kompakt olmasını sağlar.
    - **Ortalama Bağlantı (Average Linkage):** İki kümedeki tüm noktaların birbirine olan ortalama uzaklığını kullanır.

    - **Ward Metodu (Ward's Method):** Kümelerdeki varyansı (dağılımı) en az artıran iki kümeyi birleştirir. Genellikle iyi sonuçlar verdiği için popülerdir.
- **Dendrogram:** HCA'nın en önemli görsel çıktısıdır. Y ekseni (dikey eksen) kümeler arasındaki mesafeyi veya birleşme seviyesini gösterirken, X ekseni veri noktalarını gösterir. Dendrogramdaki uzun dikey çizgiler, o noktada birleşen kümelerin birbirinden ne kadar farklı olduğunu gösterir. Dendrogramı "keserek" (yatay bir çizgi çekerek) istediğiniz küme sayısını belirleyebilirsiniz. Çektiğiniz çizginin kestiği dikey dallar, sizin kümeleriniz olur.

```
###################################
# Hierarchical Clustering
###################################

df = pd.read_csv("datasets/USArrests.csv", index_col=0)

sc = MinMaxScaler((0, 1))
df = sc.fit_transform(df)

hc_average = linkage(df, "average")

#linkage: Hiyerarşik kümeleme algoritmasını uygulayan ana fonksiyon.
#dendrogram: Hiyerarşik kümeleme sonuçlarını ağaç yapısında görselleştir
en fonksiyon.
```

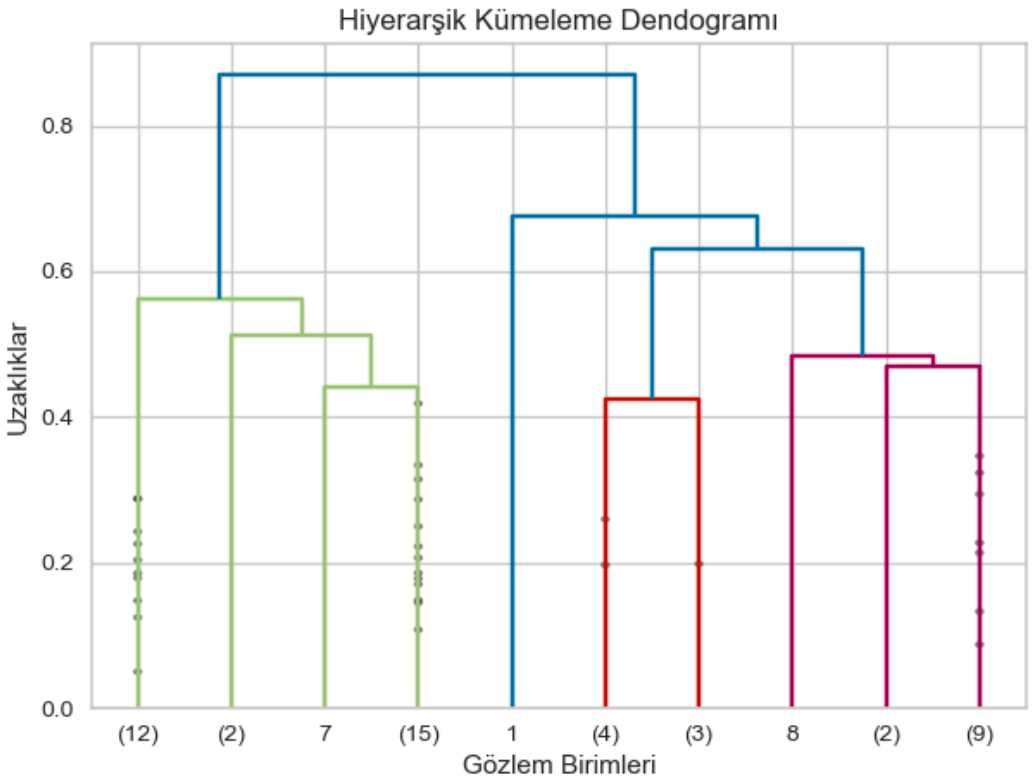
```
plt.figure(figsize=(10, 5))
plt.title("Hiyerarşik Kümeleme Dendogramı")
plt.xlabel("Gözlem Birimleri")
plt.ylabel("Uzaklıklar")
dendrogram(hc_average,
           leaf_font_size=10)
plt.show()
```



- `linkage(df, "average")`: `df` veri seti üzerinde hiyerarşik kümeleme yapar.
  - `df`: Kümeleme yapılacak ölçeklendirilmiş veri setidir.
  - `"average"`: **Bağlantı kriterini (linkage criterion)** belirler. Burada "average" (ortalama) bağlantı yöntemi kullanılmıştır. Bu, iki küme arasındaki mesafeyi, o iki kümedeki tüm noktaların birbirine olan ortalama uzaklığı olarak hesaplar. Diğer yaygın kriterler `"ward"` (Ward's metodu, kümelerdeki varyansı minimize eder), `"single"` (tek bağlantı, en yakın noktalar arası mesafe) ve `"complete"` (tam bağlantı, en uzak noktalar arası mesafe) olabilir.
- `hc_average` değişkeni, hiyerarşik kümelemenin sonuçlarını içeren bir NumPy dizisidir. Bu dizi, dendrogramı çizmek için gereken tüm bilgiyi barındırır: hangi kümelerin ne zaman ve hangi uzaklıkta birleştiğini gösterir.

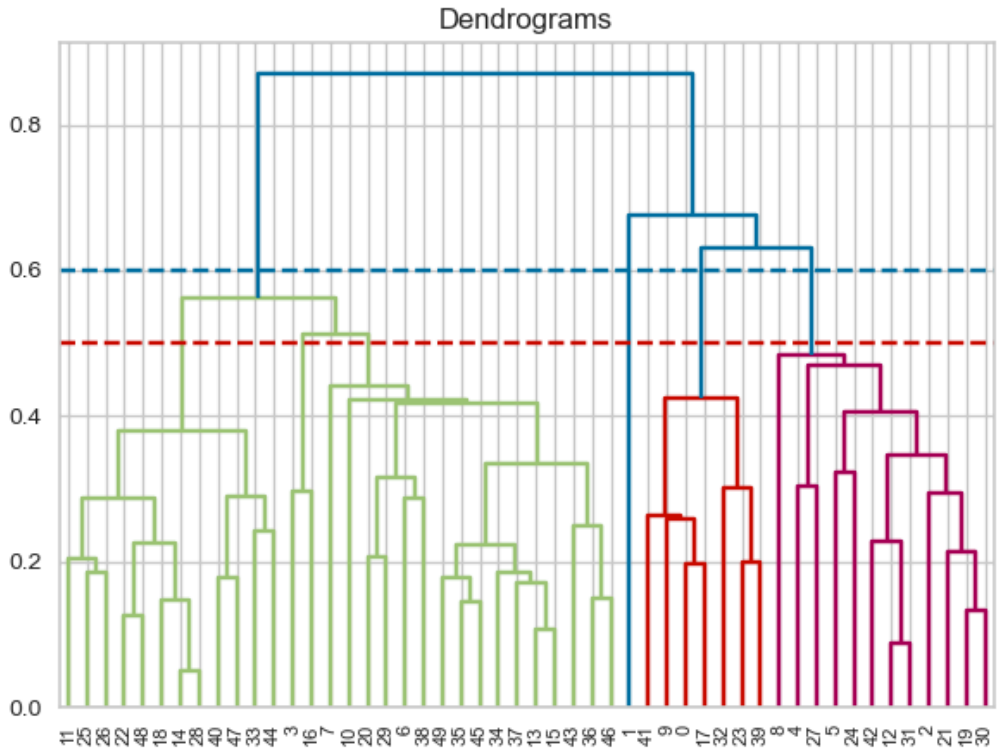
```
plt.figure(figsize=(7, 5))
plt.title("Hiyerarşik Kümeleme Dendogramı")
plt.xlabel("Gözlem Birimleri")
plt.ylabel("Uzaklıklar")
dendrogram(hc_average,
           truncate_mode="lastp",
           p=10,
           show_contracted=True,
```

```
            leaf_font_size=10)
plt.show()
```



Küme Sayısı Belirleme

```
###################################
# Kume Sayısını Belirlemek
###################################


plt.figure(figsize=(7, 5))
plt.title("Dendrograms")
dend = dendrogram(hc_average)
plt.axhline(y=0.5, color='r', linestyle='--')
plt.axhline(y=0.6, color='b', linestyle='--')
plt.show()
```



Final Modeli:

```
###################################
# Final Modeli Oluşturmak
###################################

from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=5, linkage="average")

clusters = cluster.fit_predict(df)

df = pd.read_csv("datasets/USArrests.csv", index_col=0)
df["hi_cluster_no"] = clusters

df["hi_cluster_no"] = df["hi_cluster_no"] + 1
```

```
df["kmeans_cluster_no"] = df["kmeans_cluster_no"]  + 1
df["kmeans_cluster_no"] = clusters_kmeans
```

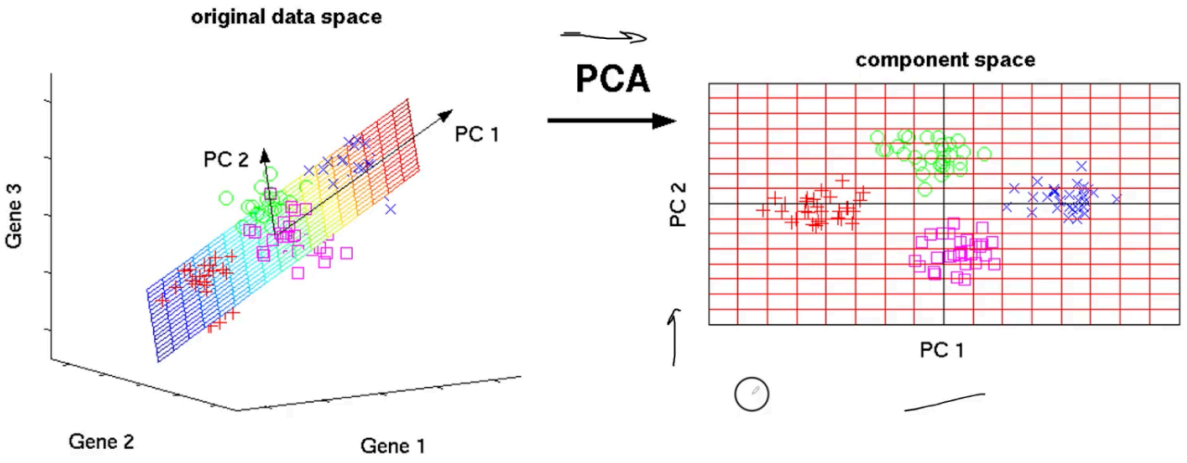
	Murder	Assault	UrbanPop	Rape	hi_cluster_no	kmeans_cluster_no
Alabama	13.2	236	58	21.2	2	3
Alaska	10.0	263	48	44.5	2	3
Arizona	8.1	294	80	31.0	2	5
Arkansas	8.8	190	50	19.5	1	1
California	9.0	276	91	40.6	2	3

K-Means vs Hiyerarşik

Özellik	K-Means Kümeleme	Hiyerarşik Kümeleme
Küme Sayısı (K)	Önceden belirtilmeli	Önceden belirtilmesi gerekmez, dendrogramdan seçilir
Küme Yapısı	Parçalı (Her nokta tek bir kümeye ait), çakışmaz	Hiyerarşik (İç içe kümeler)
Çıktı	Küme atamaları ve küme merkezleri	Dendrogram (Ağaç yapısı)
Küme Şekilleri	Genellikle küresel (spherical) kümeler için uygundur	Daha esnek, karmaşık şekilli kümeleri bulabilir
Hesaplama Maliyeti	Büyük veri setleri için daha hızlı ve verimli	Büyük veri setleri için daha yavaş ve yoğun
Görselleştirme	Doğrudan küme atamaları	Dendrogram ile küme oluşum süreci ve hiyerarşi
Global Optimum	Yerel optimuma takılabilir, başlangıç noktasına bağlı	Genellikle daha deterministiktir
Yorumlama	Küme merkezleri üzerinden yorumlanır	Dendrogram üzerinden kümeler arası ilişkiler yorumlanır

Temel Bileşen Analizi (Principal Component Analysis)

- Temel Bileşen Analizi (PCA), **boyut indirgeme (dimensionality reduction)** tekniklerinden biridir. Amacı, çok sayıda değişken (özellik/sütun) içeren bir veri setindeki bilgiyi, daha az sayıda yeni ve birbiriyle ilişkisiz (uncorrelated) değişkene sıkıştırmaktır. Bu yeni değişkenlere **"Temel Bileşenler" (Principal Components - PC'ler)** denir.
- Şöyle düşünün: Bir ressamın tuvaline bir obje çizerken farklı açılardan bakması gibi, PCA da veri setinize farklı "açılardan" bakar. Hangi açıdan bakıldığında objenin en çok bilgisini (varyansını/değişimini) görebiliyorsak, o açıyı birincil "görüş hattı" olarak seçer.
- Her bir temel bileşen, orijinal değişkenlerin bir doğrusal kombinasyonudur (yani, her bir orijinal değişkenden belirli bir oranda alarak oluşturulan yeni bir değişkendir). İlk temel bileşen (PC1), veri setindeki en fazla varyansı (değişimi) açıklayan yöndür. İkinci temel bileşen (PC2), PC1'e dik (ortogonal) olan ve kalan varyansın en çoğunu açıklayan yöndür ve bu böyle devam eder.
- Temel fikir, çok değişkenli verinin ana özelliklerini daha az sayıda değişken / bileşen ile temsil etmektir.
- **Curse of Dimensionality**
- Diğer bir ifade ile: küçük miktarda bir bilgi kaybını göze alıp değişken boyutunu azaltmaktır.



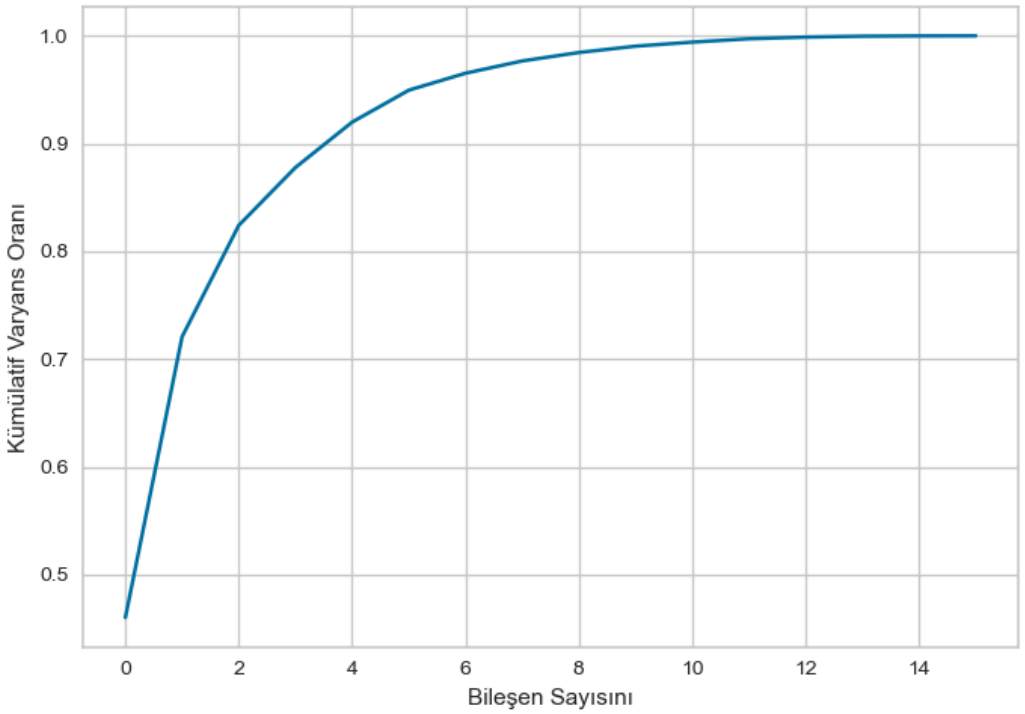
```
###################################
# Principal Component Analysis
###################################

df = pd.read_csv("datasets/Hitters.csv")
df.head()

num_cols = [col for col in df.columns if df[col].dtypes != "O" and "Salary" n
ot in col]

df[num_cols].head()

df = df[num_cols]
df.dropna(inplace=True)
df.shape

df = StandardScaler().fit_transform(df)

pca = PCA()
pca_fit = pca.fit_transform(df)

pca.explained_variance_ratio_
#array([4.60378552e-01, 2.60398491e-01, 1.03388605e-01, 5.36902121e-
02,
       # 4.20784091e-02, 2.96359092e-02, 1.57079101e-02, 1.13928108e-02,
       # 7.83230398e-03, 5.87669497e-03, 3.74765194e-03, 3.09384056e-0
3,
       # 1.55679403e-03, 8.59034766e-04, 2.86873704e-04, 7.59064046e-0
5])
np.cumsum(pca.explained_variance_ratio_)
#array([0.46037855, 0.72077704, 0.82416565, 0.87785586, 0.91993427,
       # 0.94957018, 0.96527809, 0.9766709 , 0.9845032 , 0.9903799 ,
       # 0.99412755, 0.99722139, 0.99877819, 0.99963722, 0.99992409,
       # 1.        ])
```

- `pca.explained_variance_ratio_`: Her bir **Temel Bileşenin (Principal Component)** veri setindeki toplam varyansın ne kadarını açıkladığını gösteren bir NumPy dizisidir. İlk değer, PC1'in açıkladığı varyans oranı, ikinci değer PC2'nin

açıkladığı varyans oranıdır ve bu böyle devam eder. **Bu oranlar, en çok bilgi taşıyan bileşenleri belirlememize yardımcı olur**.

- `np.cumsum(pca.explained_variance_ratio_)`: Her bir temel bileşenin **kümülatif (birikimli) olarak açıkladığı varyans oranını** hesaplar. Bu, çok önemli bir çıktıdır:
  - Örneğin, ilk değer PC1'in tek başına açıkladığı varyansı gösterir.
  - İkinci değer, PC1 ve PC2'nin birlikte açıkladığı varyansı gösterir.
  - Bu çıktıya bakarak, toplam varyansın belirli bir yüzdesini (örn. %80 veya %90) açıklamak için kaç Temel Bileşene ihtiyacımız olduğunu belirleyebiliriz. Bu, boyut indirgeme kararımızı verirken kullandığımız anahtar metriktir. Örneğin, `np.cumsum` çıktısı `[0.35, 0.60, 0.82, ...]` şeklinde ise, ilk 3 Temel Bileşenin toplam varyansın %82'sini açıkladığını görebiliriz ve bu, yeterli olabilir.

```
###################################
# Optimum Bileşen Sayısı
###################################

pca = PCA().fit(df)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel("Bileşen Sayısını")
plt.ylabel("Kümülatif Varyans Oranı")
plt.show()
```




```
##################################
# Final PCA'in Oluşturulması
##################################

pca = PCA(n_components=3)
pca_fit = pca.fit_transform(df)

pca.explained_variance_ratio_
np.cumsum(pca.explained_variance_ratio_)
#array([0.46037855, 0.72077704, 0.82416565])
```

Temel Bileşen Regresyon Modeli (PCR)

- **Temel Bileşen Regresyonu (PCR)**, Temel Bileşen Analizi (PCA) ile standart Lineer Regresyon'u birleştiren hibrit bir modelleme tekniğidir. Amacı, bağımsız değişkenler (özellikler) arasındaki **çoklu bağlantı**

(multicollinearity) sorununu çözmek ve modelin daha stabil, yorumlanabilir ve genellenebilir olmasını sağlamaktır.

Basitçe ifade etmek gerekirse:

1. **Önce PCA Yap:** Bir regresyon modelindeki bağımsız değişkenlerinizi (açıklayıcı özelliklerinizi) alırsınız.
2. **Temel Bileşenler Elde Et:** Bu bağımsız değişkenler üzerinde **Temel Bileşen Analizi (PCA)** uygularsınız. Böylece, orijinal çok sayıda birbiriyle ilişkili değişken yerine, daha az sayıda ve birbiriyle **ilişkisiz (uncorrelated)** yeni değişkenler (yani Temel Bileşenler) elde edersiniz.
3. **Sonra Regresyon Yap:** Elde ettiğiniz bu seçilmiş Temel Bileşenleri, bağımlı değişkeninizi tahmin etmek için bir **doğrusal regresyon modelinin** bağımsız değişkenleri olarak kullanırsınız.

```
##################################
# BONUS: Principal Component Regression
##################################

df = pd.read_csv("datasets/Hitters.csv")
df.shape

len(pca_fit)

num_cols = [col for col in df.columns if df[col].dtypes != "O" and "Salary" n
ot in col]
len(num_cols)

#Numerik olmayan kolonlar
others = [col for col in df.columns if col not in num_cols]

pd.DataFrame(pca_fit, columns=["PC1","PC2","PC3"]).head()

df[others].head()

final_df = pd.concat([pd.DataFrame(pca_fit, columns=["PC1","PC2","PC
3"]),
                      df[others]], axis=1)
final_df.head()
```

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

def label_encoder(dataframe, binary_col):
    labelencoder = LabelEncoder()
    dataframe[binary_col] = labelencoder.fit_transform(dataframe[binary_co
l])
    return dataframe

for col in ["NewLeague", "Division", "League"]:
    label_encoder(final_df, col)

final_df.dropna(inplace=True)

y = final_df["Salary"]
X = final_df.drop(["Salary"], axis=1)

lm = LinearRegression()
rmse = np.mean(np.sqrt(-cross_val_score(lm, X, y, cv=5, scoring="neg_me
an_squared_error")))
y.mean()
#np.float64(345.6021106351967)




cart = DecisionTreeRegressor()
rmse = np.mean(np.sqrt(-cross_val_score(cart, X, y, cv=5, scoring="neg_m
ean_squared_error")))
#np.float64(407.3288359888448)

cart_params = {'max_depth': range(1, 11),
               "min_samples_split": range(2, 20)}
```

```
# GridSearchCV
cart_best_grid = GridSearchCV(cart,
                              cart_params,
                              cv=5,
                              n_jobs=-1,
                              verbose=True).fit(X, y)

cart_final = DecisionTreeRegressor(**cart_best_grid.best_params_, random
_state=17).fit(X, y)

rmse = np.mean(np.sqrt(-cross_val_score(cart_final, X, y, cv=5, scoring="n
eg_mean_squared_error")))
#np.float64(330.1964109339104)
```

Elimizde bir veri seti var ve labelling yok ama bir sınıflandırma problemi çözmek istiyorum.

- Kümeleri oluşturduktan sonra⟶ Bunları sınıf olarak görüp yeni müşteriyi hangi sınıfa koymak bir sınıflandırma problemi olur.

PCA Görselleştirme

```
##################################
# Breast Cancer
##################################

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 500)

df = pd.read_csv("datasets/breast_cancer.csv")
df.head()
y = df["diagnosis"]
X = df.drop(["diagnosis", "id"], axis=1)
```

```
def create_pca_df(X, y):
    # Bu fonksiyon, PCA uygulayarak bağımsız değişkenleri (X) iki temel bile
şene indirger
    # ve indirgenmiş veriyi bağımlı değişken (y) ile birleştirerek bir DataFram
e döndürür.

    X = StandardScaler().fit_transform(X)
    # X (bağımsız değişkenler) veri setini standardize eder.
    # StandardScaler, her sütunu ortalaması 0 ve standart sapması 1 olacak ş
ekilde dönüştürür.
    # Bu adım, PCA gibi varyansa duyarlı algoritmalar için önemlidir, çünkü fa
rklı ölçeklerdeki özelliklerin etkisini eşitler.

    pca = PCA(n_components=2)
    # PCA (Temel Bileşen Analizi) modelini başlatır.
    # n_components=2, verilerin 2 Temel Bileşene (PC1 ve PC2) indirgenece
ğini belirtir.
    # Yani, en çok varyansı açıklayan ilk iki ana bileşen elde edilecektir.

    pca_fit = pca.fit_transform(X)
    # StandardScaler'dan dönen standardize edilmiş X verisi üzerinde PCA m
odelini eğitir (.fit(X))
    # ve ardından veriyi bu iki Temel Bileşene dönüştürür (.transform(X)).
    # pca_fit, artık her bir gözlemin iki yeni Temel Bileşen üzerindeki skorları
nı içeren bir NumPy dizisidir.

    pca_df = pd.DataFrame(data=pca_fit, columns=['PC1', 'PC2'])
    # pca_fit NumPy dizisini bir Pandas DataFrame'ine dönüştürür.
    # Bu DataFrame'in sütun isimleri 'PC1' ve 'PC2' olarak belirlenir.
    # Bu DataFrame, artık indirgenmiş bağımsız değişkenleri temsil eder.

    final_df = pd.concat([pca_df, pd.DataFrame(y)], axis=1)
    # pca_df (indirgenmiş bağımsız değişkenler) DataFrame'i ile y (bağımlı de
ğişken) DataFrame'ini birleştirir.
    # pd.DataFrame(y), y serisini veya dizisini bir DataFrame'e dönüştürür.
    # axis=1, birleştirme işleminin sütun bazında (yan yana) yapılacağını belir
tir.
    # Sonuç olarak, bağımsız değişkenlerin ilk iki temel bileşenini ve bağımlı
```

değişkeni içeren nihai bir DataFrame elde edilir.

    return final_df
    # Oluşturulan final_df'i fonksiyonun çıktısı olarak döndürür.
    # Bu DataFrame, görselleştirme veya başka analizler için kullanılabilir.

pca_df = create_pca_df(X, y)

	PC1	PC2	diagnosis
0	9.192837	1.948583	M
1	2.387802	-3.768172	M
2	5.733896	-1.075174	M
3	7.122953	10.275589	M
4	3.935302	-1.948072	M

PCA Grafiği: → Önemli Bir Grafik

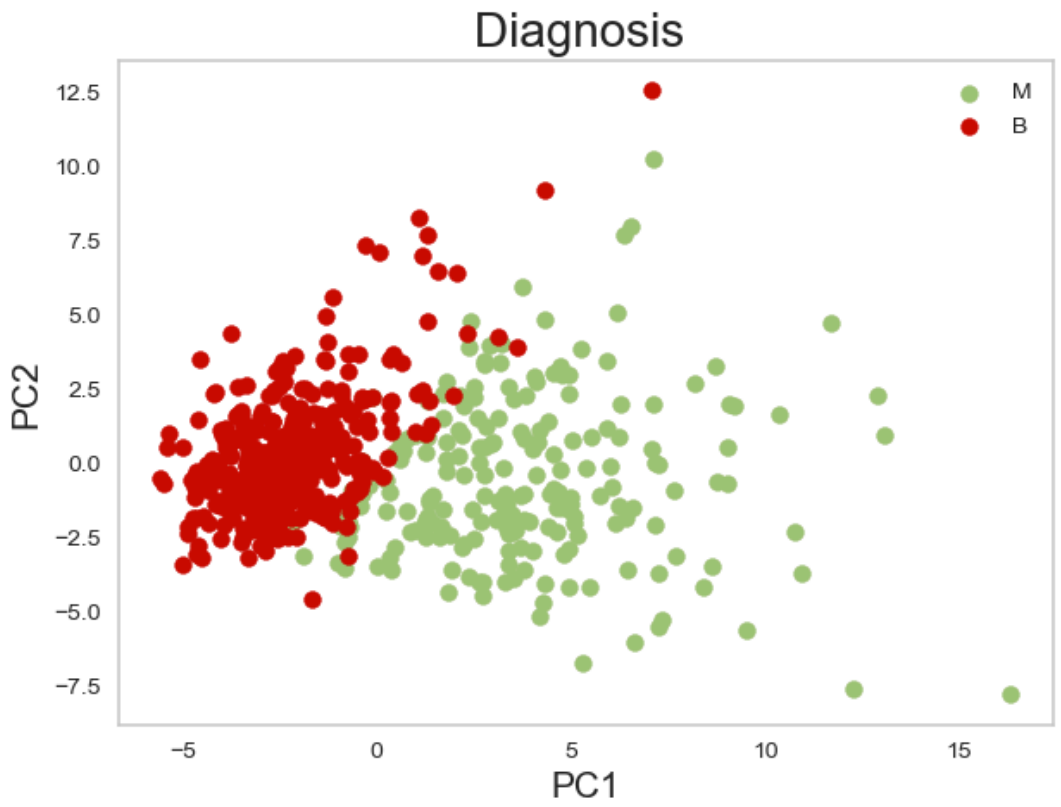
def plot_pca(dataframe, target):
    # Bu fonksiyon, PCA uygulanmış (PC1 ve PC2 sütunları içeren) bir DataFr
ame'i alır
    # ve belirlenen hedef değişkene (target) göre noktaları renklendirerek 2
boyutlu bir saçılım grafiği çizer.

    fig = plt.figure(figsize=(7, 5))
    # 7 inç genişliğinde ve 5 inç yüksekliğinde yeni bir Matplotlib figürü (çizi
m alanı) oluşturur.

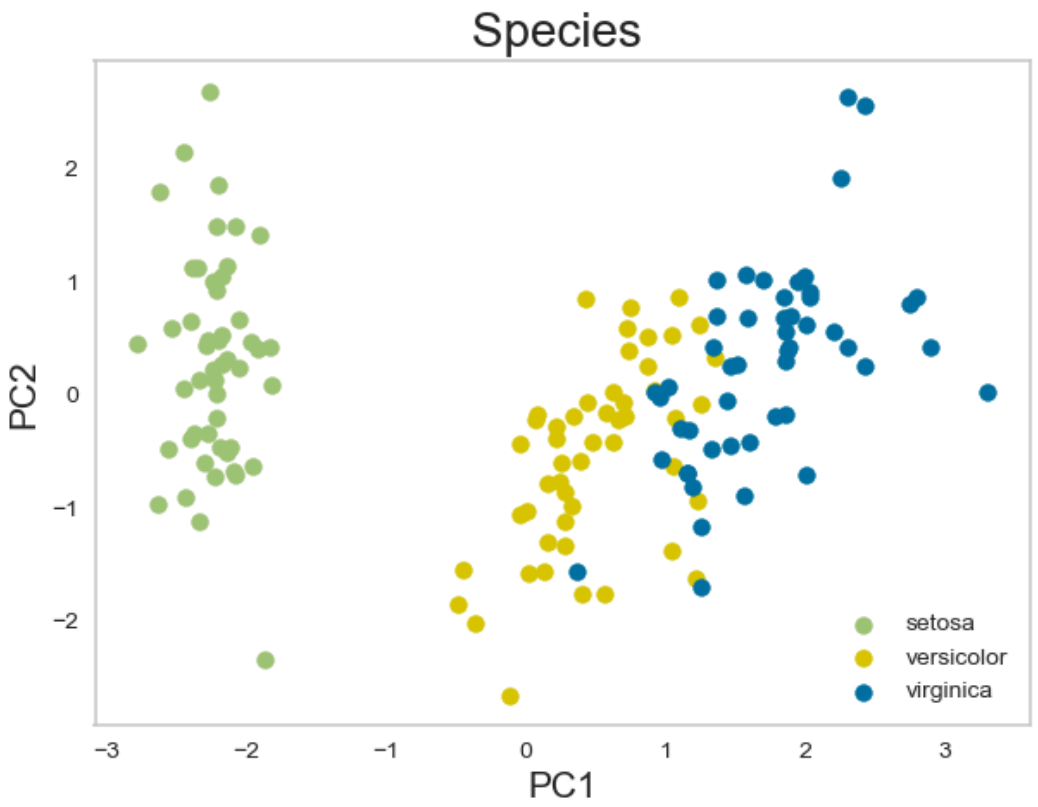
    ax = fig.add_subplot(1, 1, 1)
    # Figüre 1×1'lik bir ızgarada tek bir alt grafik (eksen - axis) ekler. Bu, grafi
ğimizin çizileceği alandır.

```
    ax.set_xlabel('PC1', fontsize=15)
    # X eksenine 'PC1' etiketini atar ve yazı tipi boyutunu 15 olarak ayarlar.
    # PC1, birinci temel bileşeni temsil eder.

    ax.set_ylabel('PC2', fontsize=15)
    # Y eksenine 'PC2' etiketini atar ve yazı tipi boyutunu 15 olarak ayarlar.
    # PC2, ikinci temel bileşeni temsil eder.

    ax.set_title(f'{target.capitalize()} ', fontsize=20)
    # Grafiğin başlığını ayarlar. Hedef değişkenin (target) adını başlıkta kullan
ır.
    # .capitalize() metodu, hedef değişkenin ilk harfini büyük yapar (örn. 'sal
ary' → 'Salary').
    # Başlığın yazı tipi boyutunu 20 olarak ayarlar.

    targets = list(dataframe[target].unique())
    # DataFrame'deki hedef sütunun (target) tüm benzersiz değerlerini alır v
e bir listeye dönüştürür.
    # Örneğin, eğer 'target' sütunu 'A', 'B', 'C' değerlerini içeriyorsa, targets
= ['A', 'B', 'C'] olur.
    # Bu, grafikteki farklı grupları temsil eder.

    # içerdeki import fonk çağrılam süresini uzatır - Bu yorum doğru bir tespi
ttir. Fonksiyon içinde import yapmak performansı düşürür.
    colors = random.sample(['r', 'b', "g", "y"], len(targets))
    # Grafikteki farklı hedef gruplarını renklendirmek için rastgele renkler seç
er.
    # random.sample, verilen listeden (burada 'r' kırmızı, 'b' mavi, 'g' yeşil,
'y' sarı)
    # 'targets' listesinin uzunluğu kadar benzersiz renk seçer.
    # Not: Eğer targets'ın uzunluğu mevcut renklerden fazlaysa hata verir ve
ya aynı renkleri tekrar kullanır.
    # Daha sağlam bir renk stratejisi için 'cmap' (renk haritası) kullanmak dah
a iyi olabilir.

    for t, color in zip(targets, colors):
        # Her bir benzersiz hedef değeri (t) ve buna karşılık gelen rengi (color)
```

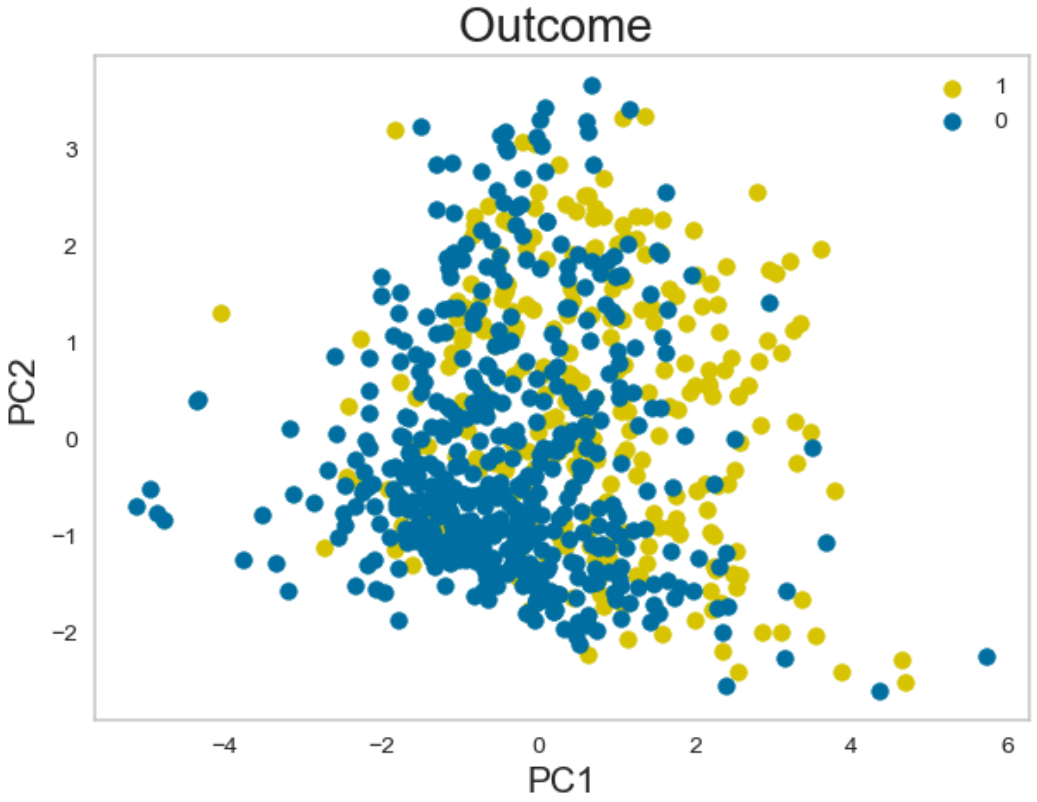
```
döngü içinde eşleştirir.

        indices = dataframe[target] == t
        # Mevcut hedef değeri (t) ile eşleşen satırların Boolean indeksini (True/False dizisi) oluşturur.
        # Bu, belirli bir hedef grubuna ait veri noktalarını seçmek için kullanılır.

        ax.scatter(dataframe.loc[indices, 'PC1'], dataframe.loc[indices, 'PC2'], c=color, s=50)
        # Seçilen (indices) veri noktalarını PC1 ve PC2 eksenleri üzerinde bir s açılım grafiği olarak çizer.
        # dataframe.loc[indices, 'PC1']: Belirli hedef grubuna ait PC1 değerleri ni seçer.
        # dataframe.loc[indices, 'PC2']: Belirli hedef grubuna ait PC2 değerleri ni seçer.
        # c=color: Bu gruba ait noktaların rengini belirler.
        # s=50: Noktaların boyutunu 50 olarak ayarlar.

    ax.legend(targets)
    # Grafiğe bir lejant (açıklama kutusu) ekler.
    # Her bir rengin hangi hedef grubunu temsil ettiğini gösterir (targets liste sindeki isimlerle).

    ax.grid()
    # Grafiğe ızgara çizgileri ekler.

    plt.show()
    # Oluşturulan grafiği ekranda gösterir.

plot_pca(pca_df, "diagnosis")
```




```
###################################
# Iris
###################################

import seaborn as sns
df = sns.load_dataset("iris")
df.shape
y = df["species"]
X = df.drop(["species"], axis=1)

pca_df = create_pca_df(X, y)

plot_pca(pca_df, "species")
```




```
###################################
# Diabetes
###################################

df = pd.read_csv("datasets/diabetes.csv")

y = df["Outcome"]
X = df.drop(["Outcome"], axis=1)

pca_df = create_pca_df(X, y)

plot_pca(pca_df, "Outcome")
```



PCA vs Others

Özellik	PCA (Temel Bileşen Analizi)	Kümeleme Modelleri
Ana Amaç	Boyut indirgeme, veri görselleştirme, gürültü azaltma	Veri noktalarını gruplama, gizli yapıları keşfetme
Çıktı	Orijinal verinin daha az boyutta yeni bir temsili (temel bileşenler)	Veri noktalarının ait olduğu küme etiketleri (veya olasılıkları)
Matematiksel Yapı	Doğrusal cebir, varyans ve kovaryans analizi	Mesafe ölçümleri, yoğunluk, olasılık dağılımları
Uygulama Şekli	Veri ön işleme adımı olarak sıkça kullanılır	Bağımsız bir veri keşfi veya segmentasyon aracıdır
Ön Şartlar	Değişkenlerin ölçeklendirilmesi genellikle önerilir	Çoğu algoritma için değişkenlerin ölçeklendirilmesi önemlidir
"Küme" Kavramı	Kendiliğinden bir kümeleme yapmaz, sadece veriyi yeniden düzenler	Verideki benzerliklere göre doğrudan kümeler oluşturur

RECAP Konuları:

- Merkez ne demektir?
- Elbow metodu
- PCA vs diğerleri?
- Kümeleme sınıflama segmentasyon arasında ne fark vardır?
  - Segmentasyon iş
  - Sınıflandırma → Label var
  - Kümeleme→ Label yok

Özellik	Sınıflandırma (Classification)	Kümeleme (Clustering)	Segmentasyon (Segmentation)
Öğrenme Tipi	Denetimli Öğrenme (Supervised)	Denetimsiz Öğrenme (Unsupervised)	Genellikle denetimsiz, ama bağlama göre değişir
Etiket Bilgisi	Var (Model eğitiminde kullanılır)	Yok (Model kendisi grupları keşfeder)	Pazarlamada genellikle yok (kümeleme ile), görüntüde çıktı etiketli olabilir
Amaç	Yeni veriyi önceden bilinen sınıflara atama	Verideki doğal, gizli grupları keşfetme	Geniş bir bütünü (pazarı, görüntüyü) anlamlı alt parçalara ayırma
Çıktı	Belirli bir sınıf etiketi ("spam", "kedi" vb.)	Grup kimliği/küme ataması (Küme 1, Küme 2 vb.)	Pazarlamada müşteri grupları; Görüntüde nesne bölgeleri
Örnek Kullanım	Hastalık teşhisi, spam tespiti, yüz tanıma	Müşteri segmentasyonu, genetik veri analizi	Pazarlama stratejileri, otonom araçlarda yol algılama
Analiz Akışı	Tahmin (Prediction)	Keşif (Discovery)	Bölümlendirme/Gruplandırma

- MR görüntülerinden kümelemeler yapılır label atanır sonrasında sınıflandırma problemi çalıştırılır. NASIL yapılır?
- K-Means vs hiyerarşik kümeleme?
- Agglomerative metod nedir?

- RFM vs Clustering Modelleri? Aradaki fark nedir?
  - RFM zaman kolonu varsa kullanılır→ Recency, frequency, monetary
  - Kümelemede bu olmak zorunda değil.
- PCA’ye niye ihtiyaç duyarız?
-