Introduction

## Gözetimsiz Öğrenme ve Temel Bileşen Analizi (PCA)

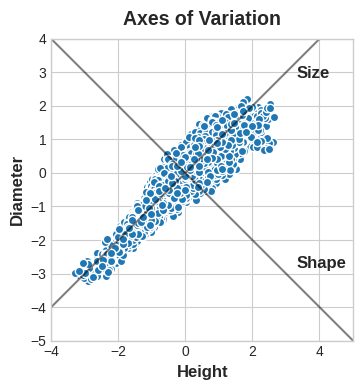
Bir önceki derste, özellik mühendisliği için model tabanlı ilk yöntemimizi, yani **kümelemeyi**, incelemiştik. Bu derste ise bir sonraki yönteme, **temel bileşen analizine (principal component analysis - PCA)** bakacağız. Kümeleme, veri kümesinin yakınlığa dayalı bir bölümlemesi olduğu gibi, PCA'yı da verideki **değişkenliğin bölümlemesi** olarak düşünebilirsiniz. PCA, verideki önemli ilişkileri keşfetmenize yardımcı olan harika bir araçtır ve aynı zamanda daha bilgilendirici özellikler oluşturmak için de kullanılabilir.

**(Teknik Not):** PCA genellikle **standartlaştırılmış** verilere uygulanır. Standartlaştırılmış verilerde "**değişkenlik**", "**korelasyon**" anlamına gelir. Standartlaştırılmamış verilerde ise "**değişkenlik**", "**kovaryans**" anlamına gelir. Bu dersteki tüm verilere, PCA uygulanmadan önce standartlaştırma yapılacaktır.

# Principal Component Analysis[¶](https://www.kaggle.com/code/ryanholbrook/principal-component-analysis#Principal-Component-Analysis)

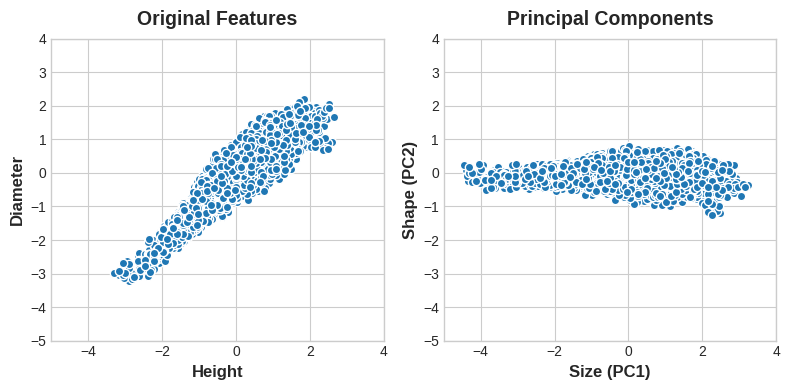
**Abalone** veri setinde, birkaç bin Tasmanya abalonundan (bir tür deniz salyangozu) alınmış fiziksel ölçümler bulunmaktadır. Şimdilik sadece birkaç özelliğe bakacağız: kabuklarının **'Yüksekliği' (Height)** ve **'Çapı' (Diameter)**.

Bu verinin içinde, abalonların birbirinden ne şekilde farklılaştığını açıklayan "**değişkenlik eksenleri**" olduğunu düşünebilirsiniz. Görsel olarak, bu eksenler, verinin doğal boyutları boyunca uzanan dik çizgiler olarak görünür ve her bir orijinal özellik için bir eksen bulunur.



Bu değişkenlik eksenlerine genellikle isimler verebiliriz. Daha uzun olan eksene **"Boyut" bileşeni** diyebiliriz: küçük yükseklik ve küçük çap (sol alt) ile büyük yükseklik ve büyük çap (sağ üst) arasındaki zıtlık. Daha kısa olan eksene ise **"Şekil" bileşeni** diyebiliriz: küçük yükseklik ve büyük çap (düz şekil) ile büyük yükseklik ve küçük çap (yuvarlak şekil) arasındaki zıtlık.

Dikkat ederseniz, abalonları 'Yükseklik' ve 'Çap'larına göre tanımlamak yerine, pekala 'Boyut' ve 'Şekil'lerine göre de tanımlayabiliriz. Aslında, **Temel Bileşen Analizi'nin (PCA)** tüm fikri budur: Veriyi orijinal özelliklerle tanımlamak yerine, değişkenlik eksenleriyle tanımlarız. **Değişkenlik eksenleri, yeni özellikler haline gelir.**



PCA'nın oluşturduğu yeni özellikler aslında orijinal özelliklerin doğrusal kombinasyonlarıdır (ağırlıklandırılmış toplamlar):

df["Size"] = 0.707 \* X["Height"] + 0.707 \* X["Diameter"]

df["Shape"] = 0.707 \* X["Height"] - 0.707 \* X["Diameter"]

Bu yeni özelliklere verinin **temel bileşenleri (principal components)** denir. Ağırlıkların kendisine ise **yüklemeler (loadings)** denir. Orijinal veri setindeki özellik sayısı kadar temel bileşen olacaktır: İki yerine on özellik kullanmış olsaydık, sonuç olarak on bileşen elde ederdik.

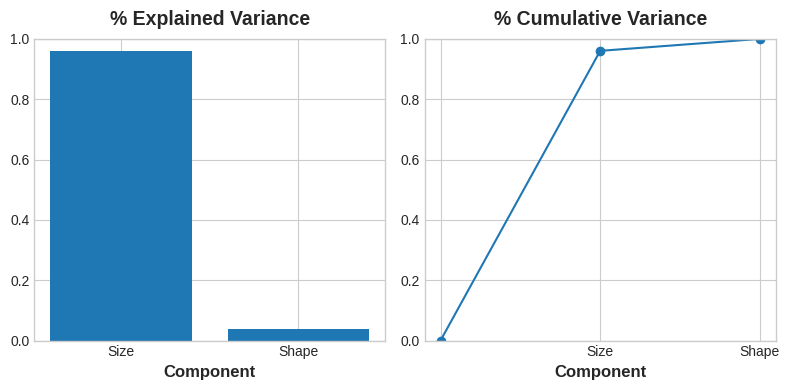
Bir bileşenin yüklemeleri, işaretler ve büyüklükler aracılığıyla hangi değişkenliği ifade ettiğini bize söyler:

| Features \ Components | Size (PC1) | Shape (PC2) |
| --- | --- | --- |
| Height | 0.707 | 0.707 |
| Diameter | 0.707 | -0.707 |

## Yüklemeler ve Açıklanan Değişkenlik

Bu yükleme tablosu, **Boyut (Size)** bileşeninde **Yükseklik (Height)** ve **Çapın (Diameter)** aynı yönde (aynı işaret) değiştiğini, ancak **Şekil (Shape)** bileşeninde zıt yönlerde (zıt işaret) değiştiğini bize söylüyor. Her iki bileşende de yüklemeler aynı büyüklüktedir ve bu nedenle özellikler her ikisine de eşit olarak katkıda bulunur.

PCA ayrıca her bir bileşendeki **değişkenlik miktarını** da bize söyler. Şekillerden, veride Boyut bileşeni boyunca, Şekil bileşeni boyunca olduğundan daha fazla değişkenlik olduğunu görebiliriz. PCA bunu, her bir bileşenin **açıklanan değişkenlik yüzdesi** aracılığıyla kesinleştirir.



Boyut bileşeni, Yükseklik ve Çap arasındaki değişimin çoğunu yakalar. Ancak, bir bileşendeki değişim miktarının, bir tahmin edici olarak ne kadar iyi olduğuyla mutlaka örtüşmediğini unutmamak önemlidir: bu, neyi tahmin etmeye çalıştığınıza bağlıdır.

# PCA for Feature Engineering

## PCA'yı Özellik Mühendisliğinde Kullanmak

Temel bileşen analizini (PCA), özellik mühendisliği için iki farklı şekilde kullanabilirsiniz.

### 1. Tanımlayıcı Bir Teknik Olarak Kullanım

PCA'yı açıklayıcı bir teknik olarak kullanabilirsiniz. Bileşenler size değişkenlik hakkında bilgi verdiğinden, bileşenler için **MI (Karşılıklı Bilgi)** puanlarını hesaplayabilir ve hangi tür değişkenliğin hedefiniz için en tahmin edici olduğunu görebilirsiniz. Bu, oluşturulacak özellik türleri için size fikir verebilir; örneğin, **'Boyut'** önemliyse **'Yükseklik'** ve **'Çapın'** çarpımı gibi bir özellik, **'Şekil'** önemliyse **'Yükseklik'** ve **'Çapın'** oranı gibi bir özellik oluşturabilirsiniz. Hatta yüksek puan alan bir veya daha fazla bileşen üzerinde kümeleme yapmayı bile deneyebilirsiniz.

### 2. Bileşenlerin Kendisini Özellik Olarak Kullanma

İkinci yol ise, **bileşenlerin kendisini özellik olarak kullanmaktır**. Bileşenler, verinin değişkenlik yapısını doğrudan ortaya çıkardıkları için, genellikle orijinal özelliklerden daha bilgilendirici olabilirler. İşte bazı kullanım durumları:

### Boyut İndirgeme (Dimensionality reduction)

Özellikleriniz aşırı derecede birbirini tekrar ettiğinde (özellikle **çoklu doğrusal** olduklarında), PCA bu tekrarlamayı sıfıra yakın varyanslı bir veya daha fazla bileşene ayırır. Bu bileşenler az veya hiç bilgi içermediği için bunları silebilirsiniz.

### Anomali Tespiti (Anomaly detection)

Orijinal özelliklerden belli olmayan sıra dışı değişkenlikler, genellikle düşük varyanslı bileşenlerde ortaya çıkar. Bu bileşenler, bir **anomali** veya **aykırı değer tespiti** görevinde oldukça bilgilendirici olabilir.

### Gürültü Azaltma (Noise reduction)

Bir sensör okumaları koleksiyonu, genellikle ortak bir arka plan gürültüsünü paylaşır. PCA bazen (bilgilendirici) sinyali daha az sayıda özellikte toplarken, gürültüyü olduğu gibi bırakabilir. Böylece **sinyal-gürültü oranını** artırır.

### Korelasyonu Giderme (Decorrelation)

Bazı makine öğrenimi algoritmaları, yüksek oranda ilişkili özelliklerle çalışmakta zorlanır. PCA, ilişkili özellikleri, algoritmanızın çalışması daha kolay olabilecek, ilişkisiz bileşenlere dönüştürür.

PCA, temel olarak verilerinizin ilişkisel yapısına doğrudan erişim sağlar. Şüphesiz kendi uygulamalarınızı geliştireceksiniz!

PCA'yı uygularken akılda tutulması gereken birkaç şey vardır:

* **Sadece Sayısal Özellikler:** PCA sadece sürekli miktarlar veya sayımlar gibi sayısal özelliklerle çalışır.
* **Ölçek Duyarlılığı:** PCA, ölçeğe duyarlıdır. Aksini gerektirecek iyi bir nedeniniz yoksa, PCA uygulamadan önce verilerinizi standartlaştırmak iyi bir uygulamadır.
* **Aykırı Değerleri Ele Alma:** Aykırı değerler sonuçlar üzerinde gereksiz bir etkiye sahip olabileceğinden, onları kaldırmayı veya kısıtlamayı düşünün.

# Example - 1985 Automobiles

Bu örnekte, Otomobil veri setimize geri dönecek ve PCA'yı özellikleri keşfetmek için tanımlayıcı bir teknik olarak kullanarak uygulayacağız. Alıştırmada diğer kullanım durumlarına bakacağız.

Bu gizli hücre verileri yükler ve plot\_variance ve make\_mi\_scores fonksiyonlarını tanımlar.

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

from IPython.display import display

from sklearn.feature\_selection import mutual\_info\_regression

plt.style.use("seaborn-whitegrid")

plt.rc("figure", autolayout=True)

plt.rc(

"axes",

labelweight="bold",

labelsize="large",

titleweight="bold",

titlesize=14,

titlepad=10,

)

def plot\_variance(pca, width=8, dpi=100):

*# Create figure*

fig, axs = plt.subplots(1, 2)

n = pca.n\_components\_

grid = np.arange(1, n + 1)

*# Explained variance*

evr = pca.explained\_variance\_ratio\_

axs[0].bar(grid, evr)

axs[0].set(

xlabel="Component", title="**% E**xplained Variance", ylim=(0.0, 1.0)

)

*# Cumulative Variance*

cv = np.cumsum(evr)

axs[1].plot(np.r\_[0, grid], np.r\_[0, cv], "o-")

axs[1].set(

xlabel="Component", title="% Cumulative Variance", ylim=(0.0, 1.0)

)

*# Set up figure*

fig.set(figwidth=8, dpi=100)

return axs

def make\_mi\_scores(X, y, discrete\_features):

mi\_scores = mutual\_info\_regression(X, y, discrete\_features=discrete\_features)

mi\_scores = pd.Series(mi\_scores, name="MI Scores", index=X.columns)

mi\_scores = mi\_scores.sort\_values(ascending=False)

return mi\_scores

df = pd.read\_csv("../input/fe-course-data/autos.csv")

Çeşitli mülkleri kapsayan dört özellik seçtik. Bu özelliklerin her biri, hedef fiyatla yüksek bir MI puanına sahip. Bu özellikler doğal olarak aynı ölçekte olmadığı için verileri standartlaştıracağız.

features = ["highway\_mpg", "engine\_size", "horsepower", "curb\_weight"]

X = df.copy()

y = X.pop('price')

X = X.loc[:, features]

*# Standardize*

X\_scaled = (X - X.mean(axis=0)) / X.std(axis=0)

Artık scikit-learn'ün PCA tahmin aracını kullanabilir ve temel bileşenleri oluşturabiliriz. Dönüştürülmüş veri kümesinin ilk birkaç satırını burada görebilirsiniz.

from sklearn.decomposition import PCA

*# Create principal components*

pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled)

*# Convert to dataframe*

component\_names = [f"PC**{**i+1**}**" for i **in** range(X\_pca.shape[1])]

X\_pca = pd.DataFrame(X\_pca, columns=component\_names)

X\_pca.head()

| PC1 | PC2 | PC3 | PC4 |
| --- | --- | --- | --- |
| 0 | 0.382486 | -0.400222 | 0.124122 | 0.169539 |
| 1 | 0.382486 | -0.400222 | 0.124122 | 0.169539 |
| 2 | 1.550890 | -0.107175 | 0.598361 | -0.256081 |
| 3 | -0.408859 | -0.425947 | 0.243335 | 0.013920 |
| 4 | 1.132749 | -0.814565 | -0.202885 | 0.224138 |

Uydurma işleminden sonra, PCA örneği, components\_ özniteliğinde yüklemeleri içerir. (Maalesef PCA terminolojisi tutarsızdır. X\_pca'daki dönüştürülmüş sütunları bileşenler olarak adlandıran kuralı izliyoruz; aksi takdirde bir adları olmaz.) Yüklemeleri bir veri çerçevesine saracağız.

loadings = pd.DataFrame(

pca.components\_.T, *# transpose the matrix of loadings*

columns=component\_names, *# so the columns are the principal components*

index=X.columns, *# and the rows are the original features*

)

loadings

| PC1 | PC2 | PC3 | PC4 |
| --- | --- | --- | --- |
| highway\_mpg | -0.492347 | 0.770892 | 0.070142 | -0.397996 |
| engine\_size | 0.503859 | 0.626709 | 0.019960 | 0.594107 |
| horsepower | 0.500448 | 0.013788 | 0.731093 | -0.463534 |
| curb\_weight | 0.503262 | 0.113008 | -0.678369 | -0.523232 |

Bir bileşenin yüklemelerinin işaretlerinin ve büyüklüklerinin, ne tür bir değişkenliği yakaladığını bize anlattığını hatırlayın. İlk bileşen (PC1), yüksek güçlü, lüks araçlarla düşük yakıt verimliliği ve daha küçük, daha ekonomik araçlarla yüksek yakıt verimliliği arasındaki zıtlığı gösteriyor. Buna "**Lüks/Ekonomi**" ekseni diyebiliriz. Aşağıdaki şekil, seçtiğimiz dört özelliğin çoğunlukla Lüks/Ekonomi ekseni boyunca değiştiğini gösteriyor.

*# Look at explained variance*

plot\_variance(pca);

Bileşenlerin MI puanlarına da bakalım. Şaşırtıcı olmayan bir şekilde, PC1 oldukça bilgilendirici; ancak kalan bileşenler, küçük farklılıklarına rağmen, fiyatla önemli bir ilişkiye sahip. Bu bileşenleri incelemek, ana Lüks/Ekonomi ekseninde yakalanmayan ilişkileri bulmak için faydalı olabilir.

mi\_scores = make\_mi\_scores(X\_pca, y, discrete\_features=False)

mi\_scores

PC1 1.013264

PC2 0.379156

PC3 0.306703

PC4 0.203329

Name: MI Scores, dtype: float64

Üçüncü bileşen beygir gücü ile boş ağırlık arasındaki karşıtlığı gösteriyor; spor arabalar ile station wagonlar arasında bir karşıtlık var gibi görünüyor.

*# Show dataframe sorted by PC3*

idx = X\_pca["PC3"].sort\_values(ascending=False).index

cols = ["make", "body\_style", "horsepower", "curb\_weight"]

df.loc[idx, cols]

Bu zıtlığı ifade etmek için yeni bir oran özelliği oluşturalım:

df["sports\_or\_wagon"] = X.curb\_weight / X.horsepower

sns.regplot(x="sports\_or\_wagon", y='price', data=df, order=2);