

Veri
anlam
içindir 😊

Temiz veri
=
Doğru sonuç 😊

Veri Önişleme



Selçuk Üniversitesi
Huğlu Meslek Yüksekokulu





Temiz veri,
daha iyi
modeller! ☺

İçindekiler

Veri Ön İşleme



Veriyi işle,
değeri keşfet! ➡

1 Veri Ön İşlemeye Giriş



- 1.1 Veri ön işleme nedir? 1
- 1.2 Neden gereklidir? 2
- 1.3 Veri ön işleme süreci 3

2 Veri Setinin Analizi



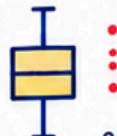
- 2.1 Veriyi tanıma 5
- 2.2 Veri türleri 6
- 2.3 Eksik, aykırı ve tekrarlı veriler 8

3 Eksik Veri İşleme



- 3.1 Eksik veri nedir? 11
- 3.2 Neden oluşur? 12
- 3.3 Tespit yöntemleri 13
- 3.4 Doldurma yöntemleri 15
- 3.5 Uygulama örnekleri 18

4 Aykırı Değer İşleme



- 4.1 Aykırı değer nedir? 21
- 4.2 Neden oluşur? 22
- 4.3 Tespit yöntemleri 23
- 4.4 İşleme yöntemleri 26
- 4.5 Uygulama örnekleri 29

5 Veri Dönüştürme



- 5.1 Normalizasyon 33
- 5.2 Standardizasyon 36
- 5.3 Min-Max Scaling 38
- 5.4 Z-score 40
- 5.5 Uygulama örnekleri 42

6 Kategorik Veri İşleme



- 6.1 Kategorik veri nedir? 45
- 6.2 Kodlama yöntemleri 47
- 6.3 One-Hot Encoding 50
- 6.4 Label Encoding 52
- 6.5 Uygulama örnekleri 54

7 Özellik Mühendisliği



- 7.1 Özellik nedir? 57
- 7.2 Yeni özellik oluşturma 59
- 7.3 Özellik seçimi 62
- 7.4 Zaman serisi özellikleri 65
- 7.5 Uygulama örnekleri 68

8 Veri Görselleştirme



- 8.1 Grafik türleri 71
- 8.2 Korelasyon analizi 74
- 8.3 Isı haritası (Heatmap) 77
- 8.4 Uygulama örnekleri 80

9 Veri Sızıntısı ve Train-Test Ayrımı



- 9.1 Veri sızıntısı nedir? 83
- 9.2 Nerelerde oluşur? 85
- 9.3 Doğru bölme stratejileri 87
- 9.4 Uygulama örnekleri 90

10 Uygulama ve Özet



- 10.1 Gerçek veri seti ile çalışma 93
- 10.2 Adım adım veri ön işleme 95
- 10.3 Sonuç ve anahtar kavramlar 99

Veri + Doğru İşlem = Daha İyi Sonuçlar! ☺

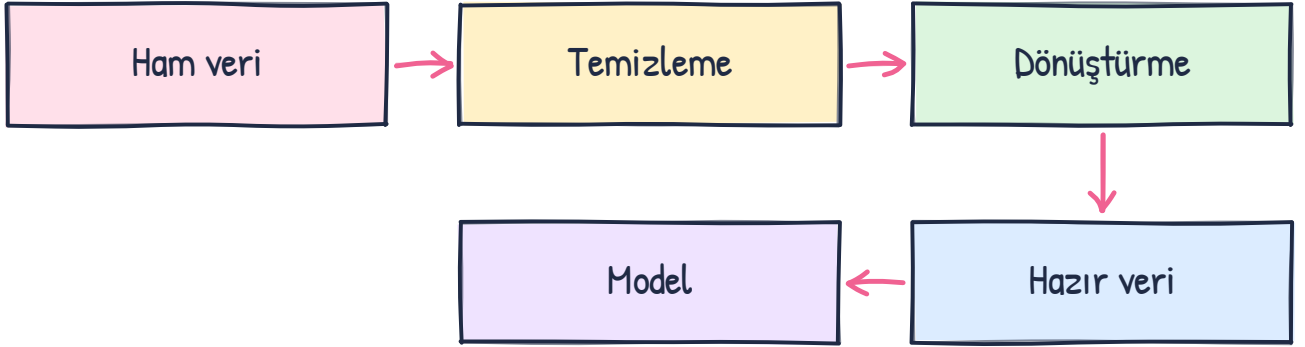


1.1 Veri ön işleme nedir?

Makine öğrenmesi modelleri veriden öğrenir. Ama gerçek hayattaki veri neredeyse hiçbir zaman "hazır" gelmez: eksik hücreler, yazım hataları, tutarsız birimler, tekrar eden satırlar... İşte **veri ön işleme** tam burada devreye girer.

Tanım

Veri ön işleme (data preprocessing), ham veriyi temizleme, düzenleme ve dönüştürerek bir analiz ya da makine öğrenmesi modeli için **kullanılabilir hâle getirme** sürecidir.



ham veriden modele giden yol

Bir benzetme: yemek yapmak

Aşçı sebzeleri doğrudan tencereye atmaz. Önce yıkar, bozuk kısımları ayıklar, doğrar, ölçer. Veri bilimci de aynısını yapar:

★ Yıkamak → **temizleme**

★ Doğramak → **dönüştürme**

★ Ayıklamak → **eksik / hatalı veriyi işleme**

★ Ölçmek → **ölçekleme**

İpucu

Veri bilimciler zamanlarının yaklaşık **%60-80'ini** veriyi hazırlamaya harcar. Yani bu ders, işin asıl kalbidir!

↳ Kötü malzemeyle iyi yemek olmaz; kötü veriyle iyi model olmaz :)

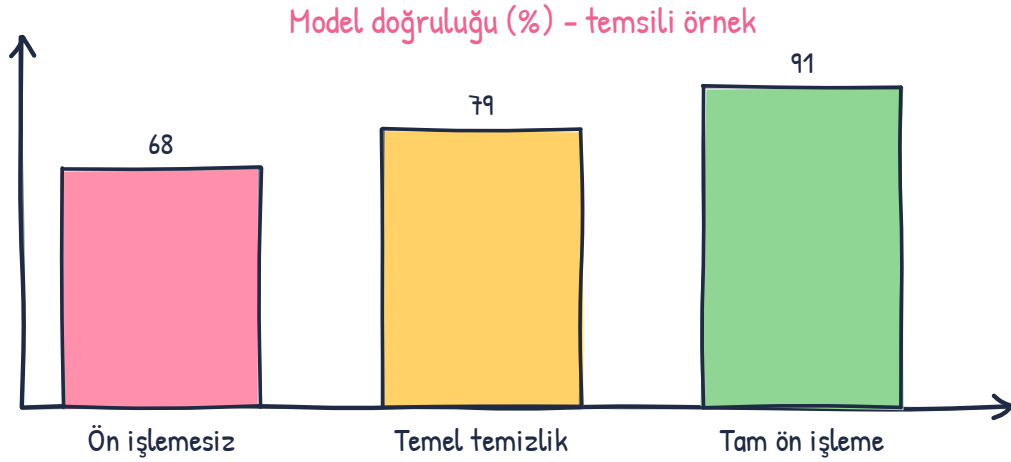


1.2 Neden gereklidir?

Bilgisayar bilimlerinin en ünlü sözlerinden biri: "Garbage in, garbage out" yani çöp girerse çöp çıkar. Model ne kadar gelişmiş olursa olsun, beslendiği veri kötüyse sonuç da kötü olur.

Ön işleme yapılmazsa neler olur?

- ★ Birçok algoritma **eksik değerle (NaN)** hiç çalışmaz, hata verir.
- ★ **Aykırı değerler** ortalamayı ve modeli yanlış yöne çeker.
- ★ Farklı ölçekteki sütunlar (yaş: 20-60, maaş: 20.000-200.000) bazı modellerde **büyük sayıların baskın** olmasına yol açar.
- ★ Modeller metni anlamaz; "Ankara", "İzmir" gibi **kategorik değerler sayıya çevrilmelidir.**
- ★ Tekrarlı satırlar bazı örneklere **fazladan ağırlık** verir.



Not

Grafikteki değerler temsildir. Ama gerçek projelerde de aynı model, sadece veri iyileştirilerek belirgin şekilde daha başarılı hâle gelebilir.

Unutma!

Daha iyi model aramadan önce **daha iyi veri** ara!



1.3 Veri ön işleme süreci

Veri ön işleme tek bir adım değil, birbirini takip eden bir **süreç** tir. Genel akış şöyledir:



tipik veri ön işleme hattı (pipeline)

1. **Veri toplama:** CSV, Excel, veritabanı, API, sensör gibi kaynaklardan veri alınır.
2. **Veriyi tanıma (EDA):** Satır/sütun sayısı, veri tipleri, temel istatistikler ve grafiklerle veri incelenir.
3. **Veri temizleme:** Eksik değerler, aykırı değerler, tekrarlı ve hatalı kayıtlar ele alınır.
4. **Veri dönüştürme:** Ölçekleme (normalizasyon, standardizasyon) ve kategorik kodlama yapılır.

İpucu

Bu adımlar çoğu zaman **döngüseldir**. Temizlik sırasında yeni bir sorun fark edip EDA'ya geri dönmek çok normaldir.



1.3 Veri ön işleme süreci (devam)

1. **Özellik mühendisliği:** Mevcut sütunlardan yeni, daha anlamlı sütunlar üretilir; gereksizler çıkarılır.
2. **Veriyi bölme:** Veri eğitim (train) ve test kümelerine ayrılır. Ön işlemenin *nerede* yapıldığı çok önemlidir (Bölüm 9).

Python'da temel araçlar

Kütüphane	Ne işe yarar?
pandas	Tablo verisini okuma, temizleme, düzenleme
numpy	Sayısal hesaplamalar, diziler
matplotlib / seaborn	Grafik çizme, görselleştirme
scikit-learn	Ölçekleme, kodlama, eksik veri, model

Python

```
import pandas as pd
import numpy as np

df = pd.read_csv("ogrenciler.csv") # veriyi oku
print(df.shape) # (satır, sütun) sayısı
df.head() # ilk 5 satıra bak
```

Özet

Veri ön işleme = tanı → temizle → dönüştür → zenginleştir → böl. Kitabın geri kalanında her adımı tek tek, örneklerle öğreneceğiz!



2.1 Veriyi tanıma

Bir doktor tedaviye başlamadan önce hastayı muayene eder. Biz de veriye dokunmadan önce onu **tanımalıyız**. Bu aşamaya **Keşifsel Veri Analizi (EDA - Exploratory Data Analysis)** denir.

İlk sorulması gereken sorular

- ★ Kaç satır (gözlem) ve kaç sütun (özellik) var?
- ★ Her sütunun **veri tipi** ne? (sayı, metin, tarih...)
- ★ Eksik değer var mı? Nereelerde?
- ★ Sayısal sütunların ortalaması, en küçük ve en büyük değeri ne?
- ★ Hedef değişken (tahmin etmek istediğimiz sütun) hangisi?

Python

```
df = pd.read_csv("musteriler.csv")
df.shape          # (1000, 6) → 1000 satır, 6 sütun
df.info()         # sütun adları, tipler, dolu hücre sayısı
df.describe()     # sayısal sütunların özet istatistikleri
df.isnull().sum() # her sütundaki eksik değer sayısı
```

Örnek veri setimiz: musteriler.csv

musteri_id	yas	gelir	sehir	cinsiyet	harcama
1	34	42000	Konya	K	3200
2	51	NaN	Ankara	E	5100
3	28	31000	İzmir	K	1800

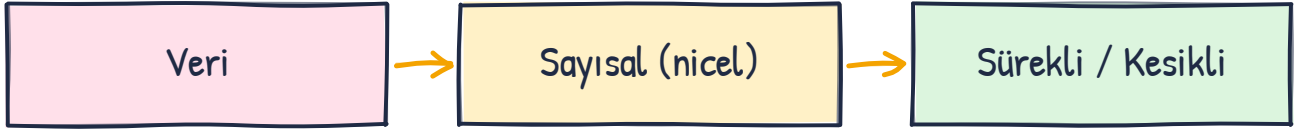
İpucu

`df.head()`, `df.tail()` ve `df.sample(5)` ile verinin başına, sonuna ve rastgele satırlarına mutlaka göz at. Sadece ilk satırlara bakmak bazı sorunları gizleyebilir!



2.2 Veri türleri

Doğru ön işleme yöntemini seçmek için önce her sütunun **hangi tür veri** içerdiğini bilmek gerekir. Veriler iki büyük gruba ayrılır:



sayısal verinin alt türleri

✎ Sayısal (nicel) veri

Ölçülebilen, sayıyla ifade edilen veridir.

- ★ **Sürekli:** Aralıktaki her değeri alabilir. Boy (172,5 cm), sıcaklık, maaş.
- ★ **Kesikli:** Sayılabilen tam değerler. Çocuk sayısı, satılan ürün adedi.

✎ Kategorik (nitel) veri

Grup veya etiket belirten veridir.

- ★ **Nominal:** Sıralama yoktur. Şehir, renk, kan grubu.
- ★ **Ordinal (sıralı):** Doğal bir sıra vardır. Eğitim durumu (ilkokul < lise < üniversite), memnuniyet (düşük < orta < yüksek).

Dikkat!

Posta kodu, telefon numarası, öğrenci numarası **sayı gibi görünse de** kategorik/kimlik bilgisidir! Bunların ortalamasını almak anlamsızdır.

✎ Diğer özel türler

- ★ **Tarih / zaman:** 2026-09-27, 14:35 gibi değerler (Bölüm 7'de işleyeceğiz).
- ★ **Metin:** Yorumlar, açıklamalar (serbest metin).
- ★ **İkili (binary):** Sadece iki değer alır: Evet/Hayır, 0/1.



2.2 Veri türleri (devam)

pandas'ta veri tipleri

pandas her sütuna bir **dtype** atar. Sütunun gerçek anlamı ile dtype'ı her zaman uyuşmayabilir, kontrol etmek bizim görevimiz.

pandas dtype	Anlamı	Örnek
int64	Tam sayı	yaş, adet
float64	Ondalıklı sayı	gelir, boy (NaN varsa int de float olur!)
object	Metin / karışık	şehir, isim
category	Kategori	cinsiyet
datetime64	Tarih-saat	kayıt tarihi
bool	Doğru/Yanlış	aktif_mi

Python

```
df.dtypes # tipleri gör
df["kayıt_tarihi"] = pd.to_datetime(df["kayıt_tarihi"])
df["cinsiyet"] = df["cinsiyet"].astype("category")
df["gelir"] = pd.to_numeric(df["gelir"], errors="coerce")
```

Not

errors="coerce", sayıya çevrilemeyen değerleri (ör. "bilinmiyor", "12.000 TL") hata vermek yerine **NaN** yapar. Sonra bunları eksik veri olarak ele alabiliriz.

Mini Soru

"Beden: S, M, L, XL" sütunu hangi veri türüdür? → **Kategorik - ordinal**, çünkü aralarında doğal bir sıralama var.



2.3 Eksik, aykırı ve tekrarlı veriler

EDA sırasında karşımıza en sık çıkan üç "kirli veri" problemi vardır. Önce bunları tanımayı öğrenelim, çözüm yöntemleri sonraki bölümlerde.

1) Eksik veri (missing value)

Bir hücrede değer olmaması durumudur. pandas'ta genellikle **NaN** (Not a Number) veya **None** olarak görünür.

ad	yas	gelir	sehir
Ayşe	34	42000	Konya
Mehmet	NaN	38000	NaN
Zeynep	29	NaN	İzmir

Python

```
df.isnull().sum()          # sütun bazında eksik sayısı
df.isnull().mean() * 100   # eksik oranı (%)
```

Dikkat!

Eksik değerler her zaman NaN olarak gelmez! "?", "-", "yok", 0, -999 gibi gizli eksikler de olabilir. `df["sutun"].unique()` ile kontrol et.

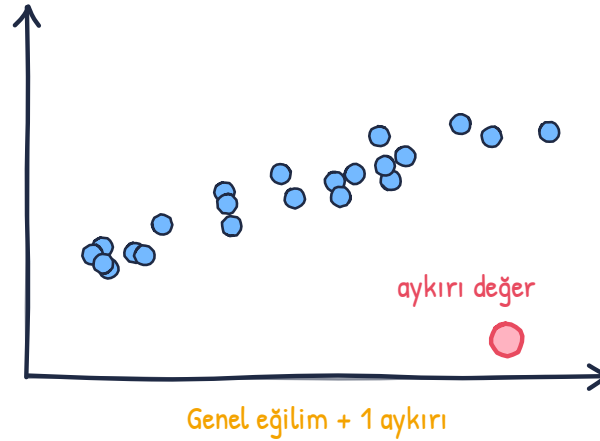
↳ Detaylar: Bölüm 3 - Eksik Veri İşleme



2.3 Eksik, aykırı ve tekrarlı veriler (devam)

2) Aykırı değer (outlier)

Verinin genel dağılımından **çok uzak** duran değerlerdir. Bir sınıfta yaşlar 18-25 arasındayken 240 değeri görmek gibi (muhtemelen 24 yazılmak istenmiş!).



çoğu nokta bir çizgi etrafında, biri çok uzakta

Aykırı değerler **hata** olabilir (yanlış girilmiş veri) ama **gerçek ve önemli** de olabilir (çok zengin bir müşteri, dolandırıcılık işlemi). Bu yüzden körü körüne silinmemelidir.

Python

```
df.describe()    # max ve min değerlere dikkat!
df.boxplot(column="gelir")    # kutu grafiğiyle hızlı kontrol
```

↳ Detaylar: Bölüm 4 - Aykırı Değer İşleme



2.3 Eksik, aykırı ve tekrarlı veriler (devam)

3) Tekrarlı veri (duplicate)

Aynı kaydın veri setinde **birden fazla kez** bulunmasıdır. Genellikle veri birleştirme veya form tekrar gönderme hatalarından kaynaklanır.

musteri_id	ad	sehir
101	Ali	Konya
102	Elif	Ankara
101	Ali	Konya

Python

```
df.duplicated().sum()           # kaç tekrarlı satır var?
df[df.duplicated()]             # tekrarlı satırları göster
df = df.drop_duplicates()        # tekrarları sil
df.drop_duplicates(subset=["musteri_id"], keep="first")
```

Diğer tutarsızlıklar

- ★ Yazım farkları: "istanbul", "İstanbul", "İSTANBUL " (sondaki boşluk!)
- ★ Birim farkları: bir satırda kg, diğerinde gram
- ★ Mantıksız değerler: yaş = -5, doğum tarihi gelecekte

Python

```
df["sehir"] = df["sehir"].str.strip().str.title()
```

Özet

EDA'nın hedefi: veriyi **anlamak** ve sorunların bir **listesini** çıkarmak. Temizliğe ancak bundan sonra geçilir!



3.1 Eksik veri nedir?

Tanım

Eksik veri, bir gözlemde (satırda) belirli bir değişken için **değerin kaydedilmemiş** olmasıdır.

Gerçek hayattaki veri setlerinin neredeyse tamamında eksik veri bulunur. Anketlerde cevaplanmayan sorular, bozuk sensörler, sonradan eklenen sütunlar... Hepsi boş hücreler bırakır.

Python'da eksik değer görünümleri

Gösterim	Nerede görülür?
NaN	numpy / pandas sayısal sütunlar
None	Python nesneleri, metin sütunları
NaT	Tarih sütunlarındaki eksik (Not a Time)
"", "?", "-"	Gizli eksikler - elle yakalanmalı

Python

```
df.replace(["?", "-", "", "yok"], np.nan, inplace=True)
```

Eksik veri neden sorun?

- ★ scikit-learn modellerinin çoğu NaN görünce **hata verir**.
- ★ Eksikler rastgele değilse sonuçlar **yanlı (bias)** olur.
- ★ Veri miktarı azalır, modelin öğrenme gücü düşer.

Unutma!

Eksik veri bir **bilgi** dir! "Neden eksik?" sorusunun cevabı, nasıl dolduracağımızı belirler.

↳ Mesela gelirini yazmayan kişilerin çoğu yüksek gelirliyse...



3.2 Neden oluşur?

Eksik verinin oluşma sebebi, hangi yöntemin doğru olduğunu belirler. İstatistikte üç temel eksiklik türü vardır:

MCAR - Tamamen Rastgele Eksik

Eksiklik hiçbir değişkenle ilişkili değildir, tamamen şans eseridir.

↳ Örnek: Laboratuvarda bir tüp yanlışlıkla kırıldı.

MAR - Rastgele Eksik

Eksiklik, veri setindeki başka bir değişkenle ilişkilidir ama eksik değer kendisiyle ilişkili değildir.

↳ Örnek: Gençler anketteki "gelir" sorusunu daha çok boş bırakıyor.

MNAR - Rastgele Olmayan Eksik

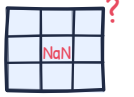
Eksiklik, eksik olan değer kendisiyle ilişkilidir. En zor durumdur.

↳ Örnek: Kilosu çok yüksek olanlar kilo sorusunu cevaplamıyor.

Tür	Silmek güvenli mi?	Dolduruma uygun mu?
MCAR	Genelde evet	Evet
MAR	Riskli	Evet (diğer sütunlar yardımıyla)
MNAR	Hayır	Dikkatli; alan bilgisi gerekir

Pratik sebepler

- ★ Veri girişi hataları, formda zorunlu olmayan alanlar
- ★ Cihaz/sensör arızaları, bağlantı kopması
- ★ Farklı kaynakların birleştirilmesi (bir kaynakta sütun yoktur)



3.3 Tespit yöntemleri

Eksik veriyi işlemekten önce **ne kadar** ve **nerede** olduğunu bulmalıyız.

1) Sayısal tespit

Python

```
df.isnull().sum()           # sütun başına eksik sayısı
(df.isnull().mean()*100).round(1) # yüzde olarak
df.isnull().any(axis=1).sum() # en az 1 eksiği olan satır sayısı
df.info()                  # "Non-Null Count" sütununa bak
```

Çıktı

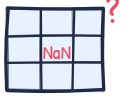
```
yas      12
gelir    87
sehir     5
cinsiyet  0
harcama   0
```

2) Eksiklik oranına göre yorum

Eksik oranı	Genel yaklaşım
%0 - %5	Silmek ya da basit doldurma genelde yeterli
%5 - %30	Doldurma (ortalama, medyan, model tabanlı)
%30 - %60	Dikkatli ol, "eksik mi?" bayrağı ekle
%60 üstü	Sütunu silmeyi düşün

Not

Bu oranlar kesin kurallar değil, **başlangıç rehberi** dir. Önemli bir sütunda %40 eksik olsa bile onu tutmak isteyebilirsiniz.



3.3 Tespit yöntemleri (devam)

3) Görsel tespit

Sayılar bize "ne kadar" olduğunu söyler; grafikler ise **eksikliklerin deseni**ni gösterir.

Python

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Eksik değer haritası")
plt.show()

df.isnull().sum().plot(kind="bar") # sütun bazında çubuk grafik
```

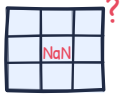


Isı haritasında neye bakılır?

- ★ **Dikey şerit:** 0 sütunda çok eksik var.
- ★ **Yatay şerit:** Bazı satırlarda neredeyse her şey eksik (bozuk kayıt olabilir).
- ★ **Birlikte eksilen sütunlar:** Aynı satırlarda eksikler → ortak bir sebep (MAR) olabilir.

İpucu

missingno kütüphanesi (`msno.matrix(df)`) eksik veriyi görselleştirmek için harika bir araçtır.



3.4 Doldurma yöntemleri

Eksik veriyle başa çıkmanın iki ana yolu vardır: **silme** veya **doldurmak** (imputation).

Yol 1: Silme

Python

```
df.dropna()           # herhangi bir NaN olan satırı sil
df.dropna(subset=["gelir"]) # sadece gelir boşsa sil
df.dropna(thresh=4)    # en az 4 dolu değeri olanları tut
df.drop(columns=["faks_no"]) # çok eksikli sütunu tamamen at
```

Artıları

- ★ Çok basit ve hızlı
- ★ Uydurma değer eklenmez

Eksileri

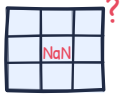
- ★ Veri kaybı!
- ★ MCAR değilse sonuçları yanıltıcı yapar

Dikkat!

`dropna()` ile 1000 satırlık veriden 400 satır silinebilir! Silmeden önce kaç satır kaybedeceğini mutlaka kontrol et.

Yol 2: Basit istatistiksel doldurma

Yöntem	Ne zaman?
Ortalama (mean)	Sayısal, simetrik dağılım, aykırı yok
Medyan	Sayısal, çarpık dağılım veya aykırı var
Mod (en sık)	Kategorik sütunlar
Sabit değer	"Bilinmiyor", 0 gibi anlamlı bir değer varsa



3.4 Doldurma yöntemleri (devam)

pandas ile doldurma

Python

```
df["yas"] = df["yas"].fillna(df["yas"].mean())
df["gelir"] = df["gelir"].fillna(df["gelir"].median())
df["sehir"] = df["sehir"].fillna(df["sehir"].mode()[0])
df["meslek"] = df["meslek"].fillna("Bilinmiyor")
```

Neden bazen medyan daha iyi?

Gelirler: 30, 32, 35, 38, **500** (bin TL)

Ortalama = 127

Medyan = 35

Tek bir çok zengin kişi ortalamayı 127'ye çıkardı; oysa tipik bir kişinin geliri 35 civarında. **Medyan aykırı değerlerden etkilenmez.**

Grup bazlı doldurma

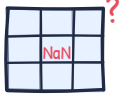
Daha akıllıca bir yol: eksik değeri, benzer satırların istatistiğiyle doldurmak.

Python

```
# Her şehrin kendi medyan geliriyle doldur
df["gelir"] = df.groupby("sehir")["gelir"] \
    .transform(lambda x: x.fillna(x.median()))
```

İpucu

Doldurmadan önce bir **bayrak sütunu** eklemek, "eksiklik" bilgisini modele aktarır: `df["gelir_eksik"] = df["gelir"].isnull().astype(int)`



3.4 Doldurma yöntemleri (devam)

📌 Zaman serilerinde doldurma

Sıralı verilerde (günlük sıcaklık, stok fiyatı) komşu değerler kullanılır:

Python

```
df["sicaklik"].ffill()      # bir önceki değerle doldur
df["sicaklik"].bfill()     # bir sonraki değerle doldur
df["sicaklik"].interpolate() # iki değer arasını doğrusal tamamla
```

Gün	Ham	ffill	interpolate
Pzt	20	20	20
Salı	NaN	20	22
Çarş	24	24	24

📌 İleri yöntemler

- ★ **KNN Imputer:** Eksik satıra en çok benzeyen *k* komşunun değerlerinin ortalamasını kullanır.
- ★ **Iterative Imputer (MICE):** Her eksik sütunu, diğer sütunlardan bir regresyon modeli ile tahmin eder.

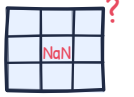
Python

```
from sklearn.impute import SimpleImputer, KNNImputer
imp = SimpleImputer(strategy="median") # mean, most_frequent, constant
X_dolu = imp.fit_transform(X)

knn = KNNImputer(n_neighbors=5)
X_dolu = knn.fit_transform(X)
```

Unutma!

Doldurma istatistiklerini (ortalama, medyan...) sadece eğitim verisinden hesapla! Aksi hâlde veri sızıntısı olur (Bölüm 9).



3.5 Uygulama örnekleri

Örnek 1: Öğrenci notları

ogrenci	vize	final	devamsizlik
A	70	80	2
B	NaN	65	5
C	90	NaN	0
D	55	60	NaN
E	85	95	1

Python

```
import pandas as pd, numpy as np
df = pd.DataFrame({
    "vize": [70, np.nan, 90, 55, 85],
    "final": [80, 65, np.nan, 60, 95],
    "devamsizlik": [2, 5, 0, np.nan, 1]})

print(df.isnull().sum())
df["vize"] = df["vize"].fillna(df["vize"].mean()) # 75.0
df["final"] = df["final"].fillna(df["final"].median()) # 72.5
df["devamsizlik"] =
df["devamsizlik"].fillna(df["devamsizlik"].median()) # 1.5
```

Mini Soru

Neden "devamsizlik" için ortalama yerine medyan seçmek mantıklı olabilir? → Devamsızlık değerleri genelde **sağa çarpıktır** (çoğu öğrenci az, birkaçı çok devamsız).



3.5 Uygulama örnekleri (devam)

Örnek 2: Kategorik eksik + bayrak

Python

```
df = pd.DataFrame({
    "sehir": ["Konya", None, "Ankara", "Konya", None],
    "gelir": [40, 55, np.nan, 38, np.nan]})

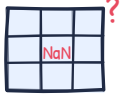
df["gelir_eksik"] = df["gelir"].isnull().astype(int)
df["sehir"] = df["sehir"].fillna(df["sehir"].mode()[0])
df["gelir"] = df["gelir"].fillna(df["gelir"].median())
print(df)
```

Çıktı

	sehir	gelir	gelir_eksik
0	Konya	40.0	0
1	Konya	55.0	0
2	Ankara	40.0	1
3	Konya	38.0	0
4	Konya	40.0	1

Not

Şehirdeki iki eksik "Konya" ile dolduruldu çünkü en sık değer Konya. Ama bu Konya'nın payını yapay olarak artırdı! Eksik oranı yüksekse "Bilinmiyor" kategorisi daha dürüst bir seçim olabilir.



3.5 Uygulama örnekleri (devam)

Örnek 3: scikit-learn ile farklı sütunlara farklı strateji

Python

```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

sayisal = ["yas", "gelir"]
kategorik = ["sehir", "meslek"]

on_isleme = ColumnTransformer([
    ("num", SimpleImputer(strategy="median"), sayisal),
    ("cat", SimpleImputer(strategy="most_frequent"), kategorik)])

X_train_dolu = on_isleme.fit_transform(X_train) # öğren + uygula
X_test_dolu = on_isleme.transform(X_test)      # sadece uygula
```

Hangi yöntemi seçmeliyim?

Durum	Önerilen yöntem
Çok az eksik (MCAR)	Satırı silmek
Sütunun çoğu boş	Sütunu silmek
Sayısal, simetrik	Ortalama
Sayısal, çarpık / aykırı var	Medyan veya KNN
Kategorik	Mod veya "Bilinmiyor"
Zaman serisi	ffill / interpolate

Özet

Tespit et → nedenini düşün → yöntemi seç → bayrak ekle → sadece train'den öğren. Eksik veri işlemede altın kural budur!

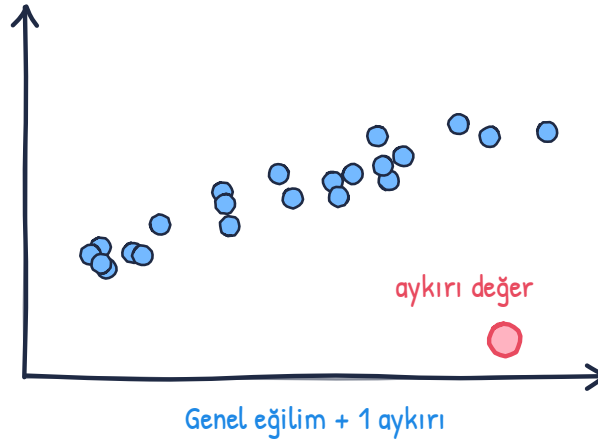


4.1 Aykırı değer nedir?

Tanım

Aykırı değer (outlier), veri setindeki diğer gözlemlerden belirgin şekilde farklı olan, dağılımın çok dışında kalan değerdir.

Bir sınıftaki öğrencilerin boyları: 165, 170, 168, 172, 175, 169, 210. Buradaki 210 cm diğerlerinden çok uzakta duruyor. Bu bir aykırı değerdir.



noktaların çoğu bir eğilimi izlerken biri kopuyor

Aykırı değer türleri

- ★ **Tek değişkenli:** Tek bir sütunda uç değer (yaş = 150).
- ★ **Çok değişkenli:** Tek tek normal ama birlikte tuhaf değerler. (Yaş = 12 normal, gelir = 50.000 normal; ama 12 yaşında 50.000 TL gelir tuhaf!)
- ★ **Bağılamsal:** Bir bağlamda normal, başka bağlamda aykırı. (Temmuz'da 35°C normal, Ocak'ta 35°C aykırı.)

Unutma!

Aykırı ≠ Hatalı. Aykırı değer "farklı" demektir; yanlış olduğu anlamına gelmez!



4.2 Neden oluşur?

Aykırı değerın kaynağını bilmek, onu silmek mi yoksa tutmak mı gerektiğine karar vermemizi sağlar.

Hata kaynaklı aykırılar (düzeltilmeli)

- ★ **Veri girişi hatası:** 24 yerine 240 yazmak, virgülü unutmak.
- ★ **Ölçüm hatası:** Bozuk sensör, yanlış kalibre edilmiş cihaz.
- ★ **Birim karışıklığı:** Bazı satırlar gram, bazıları kilogram.
- ★ **İşleme hatası:** Hatalı birleştirme, yanlış dönüştürme.
- ★ **Kod değerleri:** Eksik yerine girilmiş -999 veya 99999.

Doğal aykırılar (genelde korunmalı)

- ★ Gerçekten çok yüksek gelirli bir müşteri
- ★ Rekor kıran bir sporcu
- ★ Kredi kartı **dolandırıcılığı** veya siber saldırı

Örnek

Bir bankanın işlem verisinde 3 saniyede 15 farklı şehirden alışveriş aykırı bir değerdir. Bu tam olarak **dolandırıcılık tespit** modelinin yakalamak istediği şeydir. Onu silmek, modeli işe yaramaz yapar!

Aykırıların modele etkisi

- ★ Ortalama ve standart sapmayı bozar.
- ★ Doğrusal regresyon çizgisini kendine doğru çeker.
- ★ Ölçekleme sonuçlarını (özellikle Min-Max) sıkıştırır.

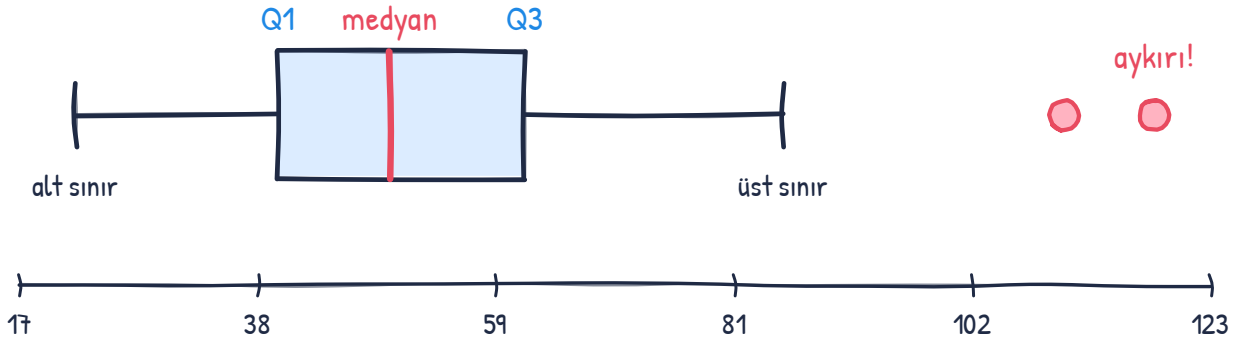


4.3 Tespit yöntemleri

Aykırı değerleri bulmanın üç yolu vardır: **görsel**, **istatistiksel** ve **model tabanlı** yöntemler.

1) Görsel yöntem: Kutu grafiği (Boxplot)

Kutu grafiği verinin dağılımını 5 sayıyla özetler: minimum, Q1, medyan, Q3, maksimum. Kutunun dışında kalan noktalar aykırı adaylarıdır.



kutu grafiği: kırmızı noktalar aykırı adayları

- ★ **Q1 (1. çeyrek):** Verinin %25'i bu değerinin altında.
- ★ **Medyan (Q2):** Verinin tam ortası.
- ★ **Q3 (3. çeyrek):** Verinin %75'i bu değerinin altında.

Python

```
import seaborn as sns
sns.boxplot(x=df["gelir"])
```

İpucu

Histogram ve dağılım (scatter) grafikleri de aykırıları görmek için çok kullanışlıdır.



4.3 Tespit yöntemleri (devam)

2) IQR (Çeyrekler Arası Açıklık) yöntemi

Kutu grafiğinin arkasındaki matematiktir. Dağılım normal olmasa da çalışır, bu yüzden en popüler yöntemdir.

$$IQR = Q3 - Q1$$

$$\text{Alt sınır} = Q1 - 1.5 \times IQR \quad \text{Üst sınır} = Q3 + 1.5 \times IQR$$

Bu sınırların dışında kalan her değer **aykırı** kabul edilir.

Elle hesaplayalım

Veri: 10, 12, 13, 14, 15, 16, 18, 19, 45

- ★ $Q1 = 13, Q3 = 18 \rightarrow IQR = 5$
- ★ $\text{Alt sınır} = 13 - 7.5 = 5.5$
- ★ $\text{Üst sınır} = 18 + 7.5 = 25.5$
- ★ $45 > 25.5 \rightarrow 45 \text{ aykırıdır!}$

Python

```
Q1 = df["gelir"].quantile(0.25)
Q3 = df["gelir"].quantile(0.75)
IQR = Q3 - Q1
alt, ust = Q1 - 1.5*IQR, Q3 + 1.5*IQR
aykirilar = df[(df["gelir"] < alt) | (df["gelir"] > ust)]
print(len(aykirilar), "aykırı değer bulundu")
```

↳ 1.5 yerine 3 kullanılırsa sadece "aşırı" aykırılar yakalanır.



4.3 Tespit yöntemleri (devam)

3) Z-score yöntemi

Bir değer ortalamadan kaç standart sapma uzakta olduğunu ölçer.

$$z = (x - \mu) / \sigma$$

- ★ Genelde $|z| > 3$ olan değerler aykırı sayılır.
- ★ Normal dağılımda verinin %99.7'si $\pm 3\sigma$ içinde kalır.

Python

```
from scipy import stats
z = np.abs(stats.zscore(df["gelir"]))
aykiriilar = df[z > 3]
```

Dikkat!

Z-score, ortalama ve standart sapmayı kullanır; bunlar da aykırılarından etkilenir! Küçük veri setlerinde veya çarpık dağılımlarda IQR daha güvenilirdir.

4) Model tabanlı yöntemler

Çok değişkenli aykırıları için kullanılır:

Yöntem	Fikir
Isolation Forest	Aykırıları rastgele bölmelerle çok çabuk "izole" edilir
LOF (Local Outlier Factor)	Komşularına göre yoğunluğu düşük olan noktalar
DBSCAN	Hiçbir kümeye ait olmayan noktalar = gürültü

Python

```
from sklearn.ensemble import IsolationForest
iso = IsolationForest(contamination=0.02, random_state=42)
df["aykiri"] = iso.fit_predict(df[["yas", "gelir"]]) # -1 = aykırı
```



4.4 İşleme yöntemleri

Aykırıyı bulduk. Şimdi ne yapacağız? Önce kendimize soralım: Bu değer hata mı, gerçek mi?

1) Silme (trimming)

Değer açıkça hatalıysa ve düzeltilemiyorsa satır silinebilir.

Python

```
df_temiz = df[(df["gelir"] ≥ alt) & (df["gelir"] ≤ ust)]
print(f"{len(df) - len(df_temiz)} satır silindi")
```

Dikkat!

Silme veri kaybına yol açar ve modelin gerçek dünyadaki uç durumları hiç görmemesine neden olabilir. Az sayıda ve kesin hatalıysa tercih et.

2) Düzeltme

Hatanın kaynağı biliniyorsa değer düzeltilir:

- ★ 240 yaş → 24 (fazladan yazılmış 0)
- ★ 1750 kg → 1.75 m (yanlış sütuna girilmiş)
- ★ -999 → NaN (sonra eksik veri olarak işlenir)

Python

```
df["yas"] = df["yas"].replace(-999, np.nan)
```

3) Eksik olarak işaretleme

Hatalı değer NaN'a çevrilir ve Bölüm 3'teki doldurma yöntemleri uygulanır.



4.4 İşleme yöntemleri (devam)

4) Baskılama (capping / winsorizing)

Aykırı değerler silinmez; alt ve üst sınıra "kırpılır". Böylece satır korunur ama uç değerlerin etkisi azalır.

Python

```
df["gelir"] = df["gelir"].clip(lower=alt, upper=ust)
```

Önce	22	40	55	62	118
Sonra (üst = 95)	22	40	55	62	95

Yüzdelik tabanlı baskılama da sık kullanılır:

Python

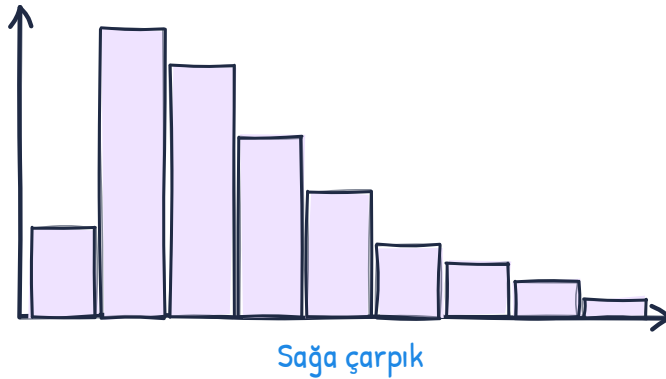
```
p1, p99 = df["gelir"].quantile([0.01, 0.99])
df["gelir"] = df["gelir"].clip(p1, p99)
```

5) Dönüşüm uygulama

Sağa çarpık verilerde (gelir, ev fiyatı, nüfus) logaritma almak, uç değerleri merkeze yaklaştırır.

Python

```
df["gelir_log"] = np.log1p(df["gelir"]) # log(1 + x), 0 değerleri için güvenli
```



log dönüşümü öncesi: uzun sağ kuyruk



4.4 İşleme yöntemleri (devam)

6) Ayır değerlendirme / koruma

Aykırı değer gerçek ve önemliyse **dokunulmaz**. Hatta ayrı bir özellik hâline getirilebilir:

Python

```
df["yuksek_harcama"] = (df["harcama"] > ust).astype(int)
```

7) Aykırılara dayanıklı yöntemler seçmek

Veriye dokunmak yerine aykırılardan az etkilenen araçları kullanabiliriz:

- ★ Ortalama yerine **medyan**
- ★ StandardScaler yerine **RobustScaler** (Bölüm 5)
- ★ Doğrusal regresyon yerine **karar ağaçları** gibi ağaç tabanlı modeller

Karar tablosu

Durum	Önerilen işlem
Açıkça hatalı, düzeltilemez	Sil veya NaN yap
Hata, kaynağı belli	Düzeltil
Gerçek ama modeli bozuyor	Baskıla veya log dönüşümü
Gerçek ve önemli (dolandırıcılık)	Koru, bayrak ekle
Emin değilim	Hem ile hem de olmadan dene, sonuçları karşılaştı

Unutma!

Aykırı değer sınırlarını (Q1, Q3, IQR) da **sadece eğitim verisinden** hesaplamalısın!



4.5 Uygulama örnekleri

Örnek 1: IQR ile aykırı bulma

Python

```
import pandas as pd
maas = pd.Series([28, 30, 31, 33, 35, 36, 38, 40, 42, 150],
name="maas")

Q1, Q3 = maas.quantile(0.25), maas.quantile(0.75)
IQR = Q3 - Q1
alt, ust = Q1 - 1.5*IQR, Q3 + 1.5*IQR
print(f"Q1={Q1}, Q3={Q3}, IQR={IQR}")
print(f"Sınırlar: {alt} - {ust}")
print("Aykırıları:", maas[(maas < alt) | (maas > ust)].tolist())
```

Çıktı

Q1=31.5, Q3=39.5, IQR=8.0
 Sınırlar: 19.5 - 51.5
 Aykırıları: [150]

Etkisini görelim

İstatistik	Aykırı dahil	Aykırı hariç
Ortalama	46.3	34.8
Medyan	35.5	35.0
Std. sapma	36.7	4.7

Not

Tek bir değer ortalamayı 11.5 birim, standart sapmayı yaklaşık 8 kat değiştirdi. Medyan ise neredeyse hiç etkilenmedi!



4.5 Uygulama örnekleri (devam)

Örnek 2: Baskılama ile satırı koruma

Python

```
maas_capped = maas.clip(lower=alt, upper=ust)
print(maas_capped.tolist())
print("Yeni ortalama:", round(maas_capped.mean(), 1))
```

Çıktı

```
[28, 30, 31, 33, 35, 36, 38, 40, 42, 51.5]
Yeni ortalama: 36.4
```

150 değeri 51.5'e çekildi. Veri kaybı olmadı; aykırının etkisi yumuşadı.

Örnek 3: Birden fazla sütunda IQR fonksiyonu

Python

```
def aykiri_baskila(df, sutunlar, k=1.5):
    df = df.copy()
    for s in sutunlar:
        q1, q3 = df[s].quantile([0.25, 0.75])
        iqr = q3 - q1
        df[s] = df[s].clip(q1 - k*iqr, q3 + k*iqr)
    return df

df_yeni = aykiri_baskila(df, ["yas", "gelir", "harcama"])
```

İpucu

Kendi fonksiyonunu yazmak, aynı işlemi farklı veri setlerinde **tekrar tekrar** kullanmanı sağlar.



4.5 Uygulama örnekleri (devam)

Örnek 4: Hatalı kodların yakalanması

Bir sensör verisinde cihaz arıza verdiği -999 yazılıyor:

Python

```
df = pd.DataFrame({"sicaklik": [21.5, 22.0, -999, 23.1, 22.8, -999, 24.0]})
print(df["sicaklik"].describe()[["mean", "min"]])

df["sicaklik"] = df["sicaklik"].replace(-999, np.nan)
df["sicaklik"] = df["sicaklik"].interpolate()
print(df["sicaklik"].tolist())
```

Çıktı

```
mean    -269.2
min      -999.0
[21.5, 22.0, 22.55, 23.1, 22.8, 23.4, 24.0]
```

Dikkat!

Ortalama sıcaklık -269.2°C çıktı! Bu açıkça imkânsız. `describe()` çıktısındaki mantıksız min/max değerleri her zaman ilk ipucudur.

Mini Soru

Bir e-ticaret sitesinde tek siparişte 500 adet ürün alan müşteri aykırı mıdır, silinmeli midir? → Aykırıdır ama muhtemelen **gerçek** bir toptan alıcıdır. Silmek yerine "toptan müşteri" bayrağı eklemek daha akıllıca.



4.5 Uygulama örnekleri (devam)

Örnek 5: Isolation Forest ile çok değişkenli aykırı

Python

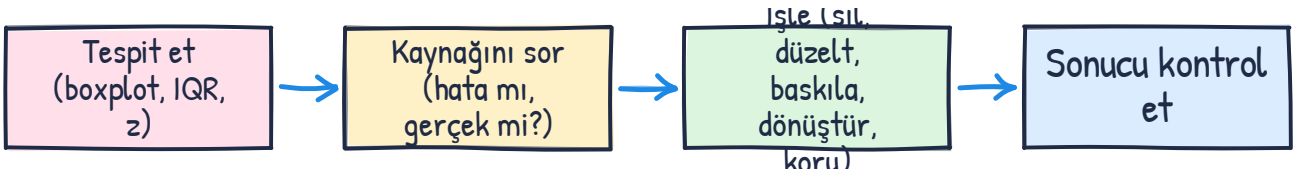
```
from sklearn.ensemble import IsolationForest
X = df[["yas", "gelir", "harcama"]]
iso = IsolationForest(contamination=0.03, random_state=42)
df["etiket"] = iso.fit_predict(X)          # 1 = normal, -1 = aykırı
print(df["etiket"].value_counts())
print(df[df["etiket"] == -1].head())
```

Çıktı

```
1      970
-1      30
```

`contamination=0.03`: verinin yaklaşık %3'ünün aykırı olduğunu varsayıyoruz.

Bölüm özeti



Özet

Aykırı değerle ilk soru her zaman "Bu değer nasıl oluştu?" olmalı. Cevap, çözümü belirler. Emin değilsen verini silme; baskıla veya dayanıklı yöntem kullan.



5.1 Normalizasyon

Veri setimizde **yaş** 18-65 arasında, **gelir** ise 15.000-250.000 arasında değişiyor olsun. İki sütun da önemli ama sayıların büyüklükleri çok farklı!

Neden ölçekleme yapmalıyız?

Mesafe kullanan algoritmalar (KNN, K-Means, SVM) iki nokta arasındaki uzaklığı hesaplar:

$$\text{mesafe} = \sqrt{(\text{yaş1} - \text{yaş2})^2 + (\text{gelir1} - \text{gelir2})^2}$$

Kişi	Yaş	Gelir
A	25	40.000
B	60	41.000

Yaş farkı 35, gelir farkı 1.000. Mesafeyi neredeyse tamamen **gelir** belirliyor; oysa bu iki kişi yaş olarak çok farklı! Ölçekleme, her sütunu aynı "dile" çevirir.

Tanım

Normalizasyon, sayısal değerleri ortak bir ölçeğe (genellikle **[0, 1]** aralığına) taşıma işlemidir. Verinin şeklini değiştirmez, sadece "cetvelini" değiştirir.

Ham veri



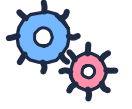
Min-Max



Z-score



aynı 4 değer, üç farklı ölçekte



5.1 Normalizasyon (devam)

Hangi algoritmalar ölçeklemeye ihtiyaç duyar?

Ölçekleme şart / çok faydalı

- ★ KNN, K-Means
- ★ SVM
- ★ Lojistik ve doğrusal regresyon (özellikle düzenleme ile)
- ★ Yapay sinir ağı
- ★ PCA

Genelde gerekmez

- ★ Karar ağaçları
- ★ Random Forest
- ★ XGBoost, LightGBM
- ★ Naive Bayes

↳ Ağaçlar "x > 5 mi?" diye sorar; ölçek önemli değildir.

Normalizasyon ailesi

Yöntem	Sonuç aralığı	Bölüm
Min-Max Scaling	[0, 1]	5.3
Z-score (standardizasyon)	ortalama 0, std 1	5.2 / 5.4
Robust Scaling	medyan 0, IQR'a göre	5.4
MaxAbs Scaling	[-1, 1]	5.5
Log dönüşümü	çarpıklığı azaltır	5.1

Not

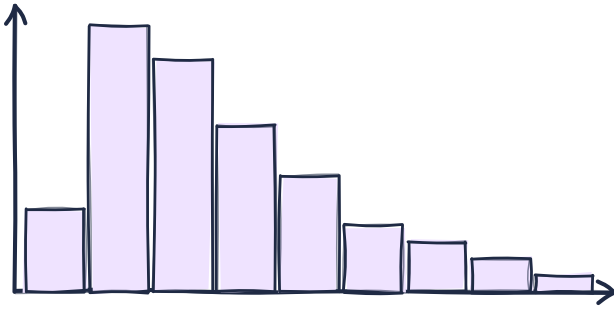
Türkçe kaynaklarda "normalizasyon" kelimesi bazen tüm ölçekleme yöntemleri için, bazen de sadece **Min-Max** için kullanılır. Bağlama dikkat et!



5.1 Normalizasyon (devam)

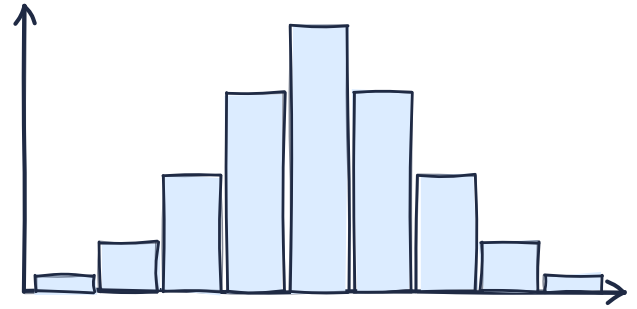
Dağılımı değiştiren dönüşümler

Ölçekleme sadece aralığı değiştirir. Ama bazen verinin **şeklini** de düzeltmek isteriz: özellikle sağa çarpık verilerde.



Sağa çarpık

önce: sağa çarpık



Normal (simetrik)

sonra: daha simetrik

Log dönüşümü

Büyük değerleri çok, küçük değerleri az "sıkıştırır". Gelir, fiyat, nüfus, tıklanma sayısı gibi verilerde çok etkilidir.

Python

```
df["gelir_log"] = np.log1p(df["gelir"]) # 0 değerleri için güvenli
```

gelir	1.000	10.000	100.000	1.000.000
log10	3	4	5	6

Diğer dönüşümler

- ★ **Karekök:** `np.sqrt(x)` - log'dan daha yumuşak.
- ★ **Box-Cox / Yeo-Johnson:** En uygun dönüşümü otomatik bulur.

Python

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method="yeo-johnson") # negatif değerlerde de çalışır
X_yeni = pt.fit_transform(X_train)
```



5.2 Standardizasyon

Tanım

Standardizasyon, veriyi ortalaması 0 ve standart sapması 1 olacak şekilde dönüştürmektir. Sonuçtaki değerlere z-skoru denir.

$$z = (x - \mu) / \sigma$$

- ★ μ (mü): sütunun ortalaması
- ★ σ (sigma): sütunun standart sapması

Sezgisel anlamı

Z-skoru bir değerın ortalamadan kaç standart sapma uzakta olduğunu söyler:

z	Anlamı
0	Tam ortalamada
+1	Ortalamanın 1 std üstünde
-2	Ortalamanın 2 std altında (oldukça düşük)
+3	Çok yüksek, aykırı olabilir

Normalizasyondan farkı

Min-Max

- ★ Sonuç her zaman [0, 1]
- ★ Aykırıları çok hassas
- ★ Sınırları bilinen veride iyi (piksel: 0-255)

Standardizasyon

- ★ Sonuç sınırsız (genelde -3 ile +3 arası)
- ★ Aykırılardan daha az etkilenir
- ★ Normal dağılıma yakın veride çok iyi



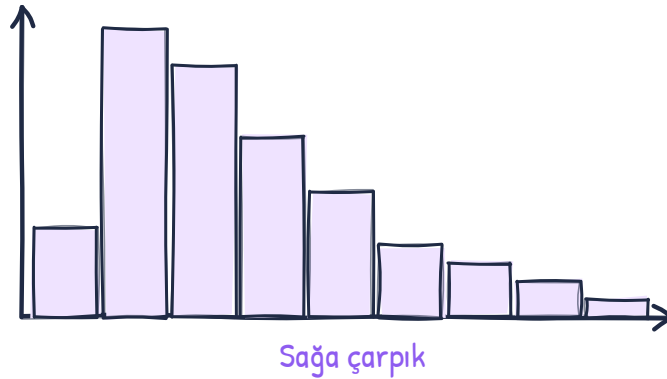
5.2 Standardizasyon (devam)

Neden bu kadar popüler?

- ★ Lojistik regresyon, SVM ve sinir ağları gibi modeller, **sıfır etrafında ortalanmış** verilerle daha hızlı ve kararlı öğrenir.
- ★ **PCA** gibi varyansa dayalı yöntemlerde her sütunun eşit başlaması gerekir.
- ★ Farklı birimlerdeki (cm, kg, TL) sütunları **karşılaştırılabilir** yapar.

Standardizasyon verinin şeklini değiştirmez!

Çarpık bir dağılım, standardizasyondan sonra da çarpık kalır. Sadece merkez 0'a kayar ve genişlik ayarlanır.



standardizasyon sonrası da şekil aynı kalır

Dikkat!

Yaygın bir yanlış: "Standardizasyon veriyi normal dağılıma çevirir." **Yanlış!** Şekli düzeltmek için log veya PowerTransformer gibi dönüşümler gerekir.

İpucu

Hangi ölçekleyiciyi seçeceğinden emin değilsen, genelde **StandardScaler** ile başla. Sonra diğerlerini dene ve model başarısını karşılaştıır.



5.3 Min-Max Scaling

Min-Max ölçekleme, her değeri sütunun en küçük ve en büyük değerine göre $[0, 1]$ aralığına taşır.

$$x' = (x - \min) / (\max - \min)$$

- ★ En küçük değer → 0
- ★ En büyük değer → 1
- ★ Diğerleri aradaki orana göre

Elle hesaplayalım

Gelir (TL): 20.000, 35.000, 50.000, 90.000 → min = 20.000, max = 90.000, fark = 70.000

x	Hesap	x'
20.000	0 / 70.000	0.00
35.000	15.000 / 70.000	0.21
50.000	30.000 / 70.000	0.43
90.000	70.000 / 70.000	1.00

İstenilen aralığa ölçekleme

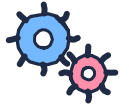
$[0, 1]$ yerine $[a, b]$ aralığı da seçilebilir:

$$x' = a + (x - \min)(b - a) / (\max - \min)$$

Python

```
MinMaxScaler(feature_range=(-1, 1))
```

↳ Görüntü işleme: pikseller 0-255 → $x/255$ ile $[0, 1]$



5.3 Min-Max Scaling (devam)

scikit-learn ile Min-Max

Python

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_s = scaler.fit_transform(X_train[["yas", "gelir"]])
X_test_s = scaler.transform(X_test[["yas", "gelir"]])

print(scaler.data_min_) # öğrenilen min değerler
print(scaler.data_max_) # öğrenilen max değerler
```

fit, transform ve fit_transform farkı

Metot	Ne yapar?	Nerede?
fit()	min/max, ortalama gibi değerleri öğrenir	Sadece train
transform()	Öğrendiklerini uygular	Train ve test
fit_transform()	İkisini birden yapar	Sadece train

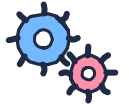
Dikkat!

Test verisinde asla `fit` veya `fit_transform` kullanma! Test setindeki bir değer train'in max'ından büyükse 1'den büyük çıkabilir; bu normaldir ve doğrudur.

Min-Max'in zayıf noktası: aykırılar

Maaşlar: 30, 32, 35, 38, 500 → Min-Max sonrası: 0.00, 0.004, 0.011, 0.017, 1.00

Normal değerlerin hepsi 0'a yığıldı! Aykırı varsa önce onu işle ya da RobustScaler kullan.



5.4 Z-score

Z-score ölçeklemesi, 5.2'de öğrendiğimiz standardizasyonun uygulamasıdır. Şimdi adım adım hesaplayalım.

Elle hesaplama

Sınav notları: 50, 60, 70, 80, 90

1. Ortalama: $\mu = (50+60+70+80+90) / 5 = 70$
2. Farklar: -20, -10, 0, 10, 20
3. Karelerin ortalaması (varyans): $(400+100+0+100+400) / 5 = 200$
4. Standart sapma: $\sigma = \sqrt{200} \approx 14.14$
5. $z = (x - 70) / 14.14$

Not	50	60	70	80	90
z-skoru	-1.41	-0.71	0.00	0.71	1.41

Not

scikit-learn'ün StandardScaler'ı n'e bölünen (popülasyon) standart sapmayı kullanır. pandas'ın `.std()` metodu ise varsayılan olarak n-1'e böler. Bu yüzden sonuçlar çok az farklı çıkabilir.

Python

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)
print(scaler.mean_, scaler.scale_) # öğrenilen  $\mu$  ve  $\sigma$ 
```



5.4 2-score (devam)

2-score'un iki görevi

1) Ölçekleme Sütunları karşılaştırılabilir yapar. Model eğitimi öncesi kullanılır.

2) Aykırı tespiti

$z > 3$ olan değerleri işaretler (Bölüm 4.3).

Robust Scaling: aykırılara dayanıklı kardeş

Ortalama ve standart sapma yerine **medyan** ve **IQR** kullanır:

$$x' = (x - \text{medyan}) / \text{IQR}$$

Python

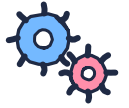
```
from sklearn.preprocessing import RobustScaler
X_s = RobustScaler().fit_transform(X_train)
```

Hangi ölçekleyici ne zaman?

Durum	Öneri
Veri yaklaşık normal, aykırı az	StandardScaler
Sınırlı aralık gerekiyor (sinir ağı, görüntü)	MinMaxScaler
Aykırı değer çok	RobustScaler
Çok çarpık dağılım	Log / PowerTransformer + Scaler
Seyrek (sparse) veri, çok sıfır	MaxAbsScaler

Unutma!

Hedef değişkeni (y) ölçeklemek genelde gerekmez. Özellikleri (X) ölçekle; regresyonda y'yi ölçeklersen tahminleri **geri dönüştürmeyi** (inverse_transform) unutma!



5.5 Uygulama örnekleri

Örnek 1: Üç ölçekleyiciyi karşılaştıralım

Python

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
RobustScaler

df = pd.DataFrame({"yas": [18, 25, 40, 60]})
df["minmax"] = MinMaxScaler().fit_transform(df[["yas"]])
df["zscore"] = StandardScaler().fit_transform(df[["yas"]])
df["robust"] = RobustScaler().fit_transform(df[["yas"]])
print(df.round(2))
```

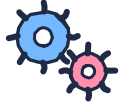
Çıktı

	yas	minmax	zscore	robust
0	18	0.00	-1.10	-0.67
1	25	0.17	-0.67	-0.34
2	40	0.52	0.26	0.34
3	60	1.00	1.51	1.26

Mini Soru

Min-Max sonrası 40 yaş neden tam ortada (0.5) değil de 0.52? → Çünkü min=18, max=60 ve orta nokta 39'dur. 40, ortanın biraz üstünde kalır.

↳ Sıralama hiçbir yöntemde değişmez; sadece sayıların "cetveli" değişir.



5.5 Uygulama örnekleri (devam)

Örnek 2: KNN'de ölçeklemenin etkisi

Python

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

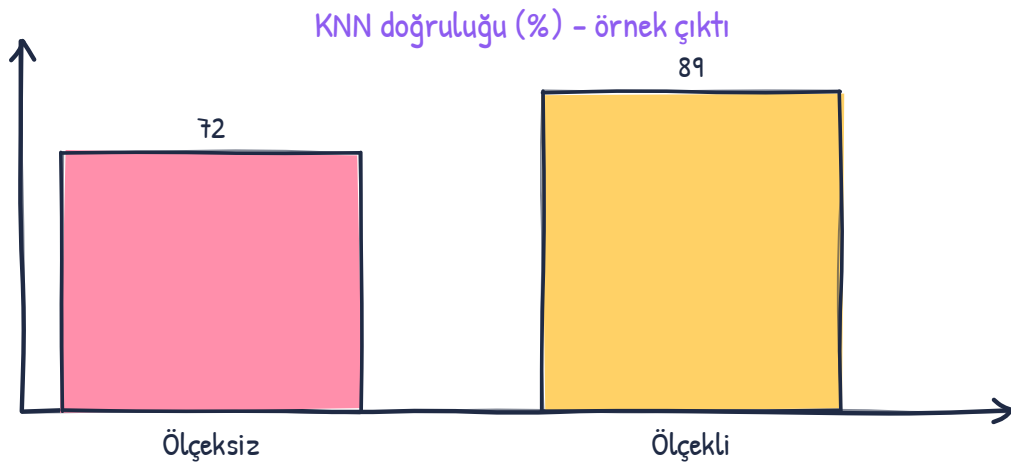
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.2, random_state=42)

# 1) Ölçeklemeden
knn = KNeighborsClassifier().fit(X_tr, y_tr)
print("Ölçeksiz :", knn.score(X_te, y_te))

# 2) Ölçekleyerek
sc = StandardScaler().fit(X_tr)
knn = KNeighborsClassifier().fit(sc.transform(X_tr), y_tr)
print("Ölçekli :", knn.score(sc.transform(X_te), y_te))
```

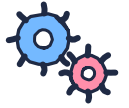
Çıktı

Ölçeksiz : 0.72
Ölçekli : 0.89



Not

Sayılar veri setine göre değişir. Ama mesafe tabanlı modellerde ölçekleme genellikle belirgin bir fark yaratır.



5.5 Uygulama örnekleri (devam)

Örnek 3: Pipeline ile güvenli ölçekleme

Ölçekleyici ile modeli bir **Pipeline** içine koymak, fit/transform karışıklığını ve veri sızıntısını önler.

Python

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

model = Pipeline([
    ("olcek", StandardScaler()),
    ("model", LogisticRegression())])

model.fit(X_tr, y_tr) # scaler sadece X_tr'den öğrenir
print(model.score(X_te, y_te)) # X_te otomatik transform edilir
```

Örnek 4: Log + ölçekleme birlikte

Python

```
df["gelir_log"] = np.log1p(df["gelir"])
df["gelir_std"] = StandardScaler().fit_transform(df[["gelir_log"]])
```

Özet

- 1) Mesafe/gradyan kullanan modellerde ölçekle.
- 2) Aykırı varsa RobustScaler veya önce aykırıyı işle.
- 3) Çarpıksa önce log.
- 4) Ölçekleyiciyi sadece train üzerinde fit et - en kolayı Pipeline!

6.1 Kategorik veri nedir?

Tanım

Kategorik veri, bir gözlemin hangi **gruba, sınıfa veya etikete** ait olduğunu belirten veridir. Değerleri genellikle metindir ve sınırlı sayıda seçenekten oluşur.

Günlük hayattan örnekler

- ★ Şehir: Konya, Ankara, İzmir
- ★ Cinsiyet: Kadın, Erkek
- ★ Kan grubu: A, B, AB, 0
- ★ Ödeme türü: Nakit, Kart, Havale
- ★ Eğitim: İlkokul, Lise, Lisans
- ★ Beden: S, M, L, XL
- ★ Memnuniyet: Düşük, Orta, Yüksek
- ★ Sonuç: Geçti, Kaldı

Neden sorun?

Makine öğrenmesi modelleri **matematik** yapar; "Konya" ile "Ankara"yı toplayamaz, çarpamaz. Bu yüzden kategorik veriler **sayıya çevrilmelidir**. Bu işleme **kodlama (encoding)** denir.

Renk	R_Kırmızı	R_Mavi	R_Yeşil
Kırmızı	1	0	0
Mavi	0	1	0
Yeşil	0	0	1
Mavi	0	1	0

aynı renk sütununun sayısal (one-hot) hâli

Unutma!

Doğru kodlama yöntemi, kategorinin **nominal** mi yoksa **ordinal** mi olduğuna bağlıdır!

6.1 Kategorik veri nedir? (devam)

Nominal ve ordinal (hatırlatma)

Tür	Sıra var mı?	Örnek	Uygun kodlama
Nominal	Hayır	Şehir, renk, marka	One-Hot
Ordinal	Evet	Eğitim, beden, not harfi	Ordinal / Label
İkili (binary)	-	Evet/Hayır, E/K	0 / 1

Kardinalite: kaç farklı değer var?

Kardinalite, bir kategorik sütundaki benzersiz değer sayısıdır.

- ★ **Düşük kardinalite:** Cinsiyet (2), kan grubu (4), kıta (7)
- ★ **Yüksek kardinalite:** İlçe (970+), ürün kodu (binlerce), posta kodu

Python

```
df["sehir"].unique()           # kaç farklı şehir?
df["sehir"].value_counts()     # her şehirden kaç tane?
df.select_dtypes(include="object").columns  # tüm metin sütunları
```

Çıktı

```
Konya      412
Ankara      301
İzmir       198
Diğer       89
```

Dikkat!

81 ilin tamamı için one-hot kodlama 80 yeni sütun demek! Yüksek kardinalitede farklı stratejiler gerekir (6.2).

6.2 Kodlama yöntemleri

Kategorik veriyi sayıya çevirmenin birçok yolu vardır. İşte en önemlileri:

Yöntem	Kısaca	En uygun durum
Label Encoding	Her kategoriye bir tamsayı	Hedef değişken, ağaç modelleri
Ordinal Encoding	Sıraya uygun tamsayılar	Sıralı kategoriler
One-Hot Encoding	Her kategoriye ayrı 0/1 sütunu	Nominal, düşük kardinalite
Binary Encoding	Kategori numarasının ikili yazımı	Orta-yüksek kardinalite
Frequency Encoding	Kategorinin görülme sıklığı	Yüksek kardinalite
Target Encoding	Kategorideki hedef ortalaması	Yüksek kardinalite (dikkatli!)

Ordinal Encoding

Sıralı kategorilerde sırayı **biz belirleriz**:

Python

```
egitim_sira = {"İlkokul": 0, "Ortaokul": 1, "Lise": 2, "Lisans": 3,
               "Yüksek Lisans": 4}
df["egitim_kod"] = df["egitim"].map(egitim_sira)
```

Python

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder(categories=[["S", "M", "L", "XL"]])
df[["beden_kod"]] = enc.fit_transform(df[["beden"]])
```

İpucu

Sırayı elle vermezsen OrdinalEncoder **alfabetik** sıralar: L=0, M=1, S=2, XL=3. Bu yanlış bir sıralamadır!

6.2 Kodlama yöntemleri (devam)

Binary (ikili) kodlama

Sadece iki değeri olan sütunlar en kolay durumdur:

Python

```
df["cinsiyet"] = df["cinsiyet"].map({"K": 0, "E": 1})
df["aktif_mi"] = df["aktif_mi"].astype(int) # True/False → 1/0
```

Frequency (frekans) kodlama

Her kategori, veri setinde ne sıklıkla geçtiği ile değiştirilir.

Python

```
frekans = df["sehir"].value_counts(normalize=True)
df["sehir_fre"] = df["sehir"].map(frekans)
```

sehir	sehir_fre
Konya	0.41
Ankara	0.30
İzmir	0.20

↳ Tek sütun üretir; yüksek kardinalitede çok pratiktir.

Nadir kategorileri gruplamak

Çok az görülen kategorileri "Diğer" altında toplamak sütun sayısını azaltır:

Python

```
sayilar = df["sehir"].value_counts()
nadir = sayilar[sayilar < 30].index
df["sehir"] = df["sehir"].replace(nadir, "Diğer")
```

6.2 Kodlama yöntemleri (devam)

Target (hedef) kodlama

Her kategori, o kategorideki **hedef değişkenin ortalaması** ile değiştirilir.

sehir	satın_aldı (y)
Konya	1
Konya	0
Ankara	1
Ankara	1

→ Konya = 0.5, Ankara = 1.0

Python

```
ort = df_train.groupby("sehir")["satın_aldı"].mean()
df_train["sehir_te"] = df_train["sehir"].map(ort)
df_test["sehir_te"] = df_test["sehir"].map(ort) # train'den
öğrenilen değerler!
```

Dikkat!

Target encoding, hedef bilgisini doğrudan özelliğe taşıdığı için **veri sızıntısına** çok açıktır. Sadece train'den hesapla; mümkünse çapraz doğrulama ve yumuşatma (smoothing) kullan. scikit-learn 1.3+ sürümündeki **TargetEncoder** bunu otomatik yapar.

Karar şeması

1. Sadece 2 değer mi var? → 0 / 1
2. Değerler arasında doğal bir sıra var mı? → Ordinal Encoding
3. Sıra yok, az sayıda değer var → One-Hot Encoding
4. Sıra yok, çok sayıda değer var → Frekans / Target / gruplama

6.3 One-Hot Encoding

Tanım

One-Hot Encoding, her kategori için ayrı bir sütun oluşturur. Satır o kategoriye aitse sütuna 1, değilse 0 yazılır.

Neden gerekli?

Nominal verilerde Label Encoding (Konya=0, Ankara=1, İzmir=2) modele yanlış bir sıra öğretir: "İzmir > Ankara > Konya" veya "İzmir = 2 × Ankara" gibi. One-hot bu sorunu çözer; tüm kategoriler eşit uzaklıkta olur.

sehir	sehir_Ankara	sehir_İzmir	sehir_Konya
Konya	0	0	1
Ankara	1	0	0
İzmir	0	1	0
Konya	0	0	1

pandas ile: get_dummies

Python

```
df_kodlu = pd.get_dummies(df, columns=["sehir"], dtype=int)
df_kodlu = pd.get_dummies(df, columns=["sehir"], drop_first=True, dtype=int)
```

Kukla değişken tuzağı (dummy variable trap)

3 şehir için aslında 2 sütun yeterli: Ankara=0 ve İzmir=0 ise kişi kesinlikle Konya'lıdır. Tüm sütunları tutmak doğrusal modellerde çoklu doğrusal bağlantı sorununa yol açabilir. `drop_first=True` bir sütunu siler.

↳ Ağaç modelleri ve düzenlileştirmeli modellerde bu genelde sorun değildir.

A	0
B	1
C	2

6.3 One-Hot Encoding (devam)

📌 scikit-learn ile: OneHotEncoder

Gerçek projelerde `OneHotEncoder` tercih edilir çünkü train'de öğrendiği kategorileri hatırlar.

Python

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
X_train_ohe = ohe.fit_transform(X_train[["sehir"]])
X_test_ohe = ohe.transform(X_test[["sehir"]])
print(ohe.get_feature_names_out())
```

Çıktı

```
['sehir_Ankara' 'sehir_Konya' 'sehir_İzmir']
```

📌 get_dummies'in gizli tehlikesi

Test verisinde train'de olmayan bir şehir ("Bursa") varsa ya da bir şehir hiç yoksa, `get_dummies` farklı sayıda sütun üretir ve model hata verir!

İpucu

`handle_unknown="ignore"`: Test'te yeni bir kategori gelirse tüm sütunlar 0 yazılır, hata oluşmaz.

📌 Artıları ve eksileri

Artıları

- ★ Sahte sıralama yaratmaz
- ★ Anlaşılması kolay
- ★ Doğrusal modellerle uyumlu

Eksileri

- ★ Çok sütun (boyut patlaması)
- ★ Seyrek matrisler, bellek maliyeti
- ★ Yüksek kardinalitede verimsiz

6.4 Label Encoding

Tanım

Label Encoding, her benzersiz kategoriye bir **tamsayı** atar: 0, 1, 2, ... Kategoriler genellikle alfabetik sıraya göre numaralanır.

meyve	meyve_kod
Elma	0
Kiraz	1
Muz	2
Elma	0

Python

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["meyve_kod"] = le.fit_transform(df["meyve"])
print(le.classes_) # ['Elma' 'Kiraz' 'Muz']
print(le.inverse_transform([2, 0])) # ['Muz' 'Elma']
```

pandas alternatifi

Python

```
df["meyve_kod"] = df["meyve"].astype("category").cat.codes
```

Nerede kullanılır?

- ★ **Hedef değişkeni (y)** kodlamak: "Geçti/Kaldı", "Kedi/Köpek/Kuş". LabelEncoder aslında bunun için tasarlanmıştır.
- ★ **Ağaç tabanlı modeller:** Karar ağaçları sayıların büyüklüğünü "anlam" olarak değil, **bölme noktası** olarak kullandığı için genelde sorun olmaz.

6.4 Label Encoding (devam)

Nerede sorun çıkarır?

Doğrusal regresyon, KNN, SVM, sinir ağları gibi modeller sayıların büyüklüğünü gerçek bir değer olarak yorumlar.

Örnek

Renk: Kırmızı = 0, Mavi = 1, Yeşil = 2 → Model, "Yeşil, Kırmızı'dan 2 kat daha fazla renk" veya "Mavi, Kırmızı ile Yeşil'in ortalaması" gibi anlamsız ilişkiler öğrenebilir!

Label vs Ordinal vs One-Hot

	Label	Ordinal	One-Hot
Sütun sayısı	1	1	Kategori sayısı
Sıra bilgisi	Alfabetik (anlamsız)	Bizim verdiğimiz	Yok
Nominal veri	X (ağaçlar hariç)	X	✓
Ordinal veri	X	✓	✓ (sıra kaybolur)
Hedef (y)	✓	-	-

Test setinde yeni etiket sorunu

LabelEncoder, fit sırasında görmediği bir değerle karşılaşarsa ValueError verir. Özellikler için OrdinalEncoder'i `handle_unknown="use_encoded_value"`, `unknown_value=-1` ile kullanmak daha güvenlidir.

Unutma!

Özellikler (X) için: OrdinalEncoder / OneHotEncoder. Hedef (y) için: LabelEncoder.

6.5 Uygulama örnekleri

Örnek 1: Karışık bir veri seti

musteri	sehir	egitim	cinsiyet	uyelik
1	Konya	Lise	K	Altın
2	Ankara	Lisans	E	Gümüş
3	İzmir	Lisans	K	Bronz
4	Konya	Y. Lisans	E	Altın

Önce her sütunu inceleyelim:

- ★ **sehir** → nominal, 3 değer → **One-Hot**
- ★ **egitim** → ordinal (Lise < Lisans < Y. Lisans) → **Ordinal**
- ★ **cinsiyet** → ikili → **0 / 1**
- ★ **uyelik** → ordinal (Bronz < Gümüş < Altın) → **Ordinal**

Python

```
df["cinsiyet"] = df["cinsiyet"].map({"K": 0, "E": 1})
df["egitim"] = df["egitim"].map({"Lise": 0, "Lisans": 1, "Y. Lisans": 2})
df["uyelik"] = df["uyelik"].map({"Bronz": 0, "Gümüş": 1, "Altın": 2})
df = pd.get_dummies(df, columns=["sehir"], dtype=int)
```

Mini Soru

Neden "uyelik" için One-Hot yerine Ordinal seçtik? → Çünkü Altın > Gümüş > Bronz şeklinde anlamalı bir sıra var ve bu bilgi modele faydalı olabilir.

6.5 Uygulama örnekleri (devam)

Örnek 1'in sonucu

musteri	egitim	cinsiyet	uyelik	sehir_Ankara	sehir_Konya	sehir_...
1	0	0	2	0	1	0
2	1	1	1	1	0	0
3	1	0	0	0	0	1
4	2	1	2	0	1	0

Artık bütün sütunlar sayısal. Model eğitime hazır!

Örnek 2: ColumnTransformer ile profesyonel yol

Python

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

on_isleme = ColumnTransformer([
    ("ohe", OneHotEncoder(handle_unknown="ignore"), ["sehir"]),
    ("ord", OrdinalEncoder(categories=[["Lise", "Lisans", "Y. Lisans"],
                                      ["Bronz", "Gümüş", "Altın"]]),
     ["egitim", "uyelik"])],
    remainder="passthrough") # diğer sütunları olduğu gibi bırak

X_train_k = on_isleme.fit_transform(X_train)
X_test_k = on_isleme.transform(X_test)
```

İpucu

`remainder="passthrough"` yazılmazsa, listelenmeyen sütunlar varsayılan olarak silinir!

A	0
B	1
C	2

6.5 Uygulama örnekleri (devam)

Örnek 3: Hedef değişkeni kodlama

Python

```
from sklearn.preprocessing import LabelEncoder
y = ["Geçti", "Kaldı", "Geçti", "Geçti", "Kaldı"]
le = LabelEncoder()
y_kod = le.fit_transform(y)
print(y_kod)                    # [0 1 0 0 1]
# Modelin tahminini tekrar metne çevirmek:
print(le.inverse_transform([1, 0])) # ['Kaldı' 'Geçti']
```

Örnek 4: Yüksek kardinalite - ürün kategorisi

Python

```
print(df["urun_kodu"].nunique())    # 2350 farklı ürün!
frek = df_train["urun_kodu"].value_counts(normalize=True)
df_train["urun_frek"] = df_train["urun_kodu"].map(frek)
df_test["urun_frek"] = df_test["urun_kodu"].map(frek).fillna(0)
```

↳ Test'te train'de hiç görülmemiş ürün → frekans 0

Özet

Nominal → One-Hot · Ordinal → Ordinal (sırayı sen ver!) · İkili → 0/1 ·
Yüksek kardinalite → frekans / target / grupta · Hedef (y) →
LabelEncoder. Ve her zaman: encoder'ı sadece train üzerinde fit et.



7.1 Özellik nedir?

Tanım

Özellik (feature), modelin tahmin yaparken kullandığı girdi değişkenidir. Tablodaki her sütun (hedef değişken hariç) bir özelliktir.

	yas	gelir	sehir	satin_aldi
Özellik mi?	✓	✓	✓	X (hedef, y)

- ★ Özelliklerin tamamına X, hedefe y denir.
- ★ Diğer isimleri: *değişken, nitelik, bağımsız değişken, attribute, predictor.*

Özellik mühendisliği nedir?

Tanım

Özellik mühendisliği (feature engineering), alan bilgisi ve yaratıcılık kullanarak ham veriden modelin daha iyi öğrenebileceği yeni ve anlamlı özellikler üretme, dönüştürme ve seçme sürecidir.



Ünlü bir söz: *"Uygulamalı makine öğrenmesi temelde özellik mühendisliğidir."* (Andrew Ng) Çoğu zaman daha karmaşık bir model seçmekten çok daha fazla kazanç sağlar.

↳ İyi özellik = modelin işini kolaylaştıran özellik



7.1 Özellik nedir? (devam)

İyi bir özelliğin nitelikleri

- ★ **İlişkili:** Hedef değişkenle gerçek bir bağı var.
- ★ **Bilgilendirici:** Her satırda aynı değer olan sütun hiçbir şey öğretmez.
- ★ **Bağımsız:** Başka bir özelliğin kopyası değil.
- ★ **Tahmin anında elde edilebilir:** Geleceğe ait bilgi içermiyor (Bölüm 9!).
- ★ **Anlaşılır:** Mümkünse yorumlanabilir.

Bir örnekle sezgi

Bir evin fiyatını tahmin ediyoruz. Elimizde **toplam alan** ve **oda sayısı** var.

alan (m ²)	oda	fiyat
120	3	3.2 M
90	3	2.4 M

Yeni bir özellik: **oda başına alan = alan / oda**. İki ev aynı oda sayısına sahip ama ilki çok daha ferah. Bu yeni özellik bu farkı doğrudan gösterir!

Özellik mühendisliğinin 3 ana başlığı

Başlık	Soru	Bölüm
Özellik oluşturma	Ne ekleyebilirim?	7.2
Özellik seçimi	Neleri çıkarmalıyım?	7.3
Özel veri türleri	Tarih/zaman nasıl işlenir?	7.4

Unutma!

En güçlü özellik mühendisliği aracı kütüphane değil, **alan bilgisidir (domain knowledge)**. Verinin geldiği alanı anlamak için uzmanlarla konuş!



7.2 Yeni özellik oluşturma

1) Matematiksel birleştirme

Mevcut sayısal sütunları toplayarak, çıkararak, bölerek veya çarparak yeni özellikler oluşturulabilir.

Python

```
df["vki"] = df["kilo"] / (df["boy"] / 100) ** 2 # Vücut Kitle İndeksi
df["kisi_basi_gelir"] = df["hane_geliri"] / df["hane_kisi"]
df["toplam_harcama"] = df["market"] + df["giyim"] + df["eglence"]
df["harcama_orani"] = df["harcama"] / df["gelir"]
```

2) Gruplama (binning / ayırıklaştırma)

Sürekli bir değişkeni aralıklara bölerek kategorik hâle getirmek:

Python

```
df["yas_grubu"] = pd.cut(df["yas"],
                        bins=[0, 17, 30, 45, 65, 120],
                        labels=["Çocuk", "Genç", "Yetişkin", "Orta
yaş", "Yaşlı"])
df["gelir_dilimi"] = pd.qcut(df["gelir"], q=4, labels=["D1", "D2",
"D3", "D4"])
```

pd.cut Aralık sınırlarını biz belirleriz (eşit genişlik veya elle).

pd.qcut Her grupta yaklaşık eşit sayıda gözlem olur (çeyrekler).

İpucu

Gruplama, aykırı değerlerin etkisini azaltır ve doğrusal olmayan ilişkileri basit modellere öğretmeye yardımcı olur. Ama **bilgi kaybına** da yol açar.



7.2 Yeni özellik oluşturma (devam)

3) Grup istatistikleri (aggregation)

Bir kişinin veya kategorinin geçmiş davranışını özetleyen özellikler çok güçlüdür.

Python

```
# Her müşterinin sipariş istatistikleri
ozet = siparisler.groupby("musteri_id").agg(
    siparis_sayisi=("siparis_id", "count"),
    ort_tutar=("tutar", "mean"),
    max_tutar=("tutar", "max"))
df = df.merge(ozet, on="musteri_id", how="left")

# Kişinin geliri, kendi şehrinin ortalamasına göre nerede?
df["sehir_ort_gelir"] = df.groupby("sehir")["gelir"].transform("mean")
df["gelir_fark"] = df["gelir"] - df["sehir_ort_gelir"]
```

4) Metinden özellik çıkarma

Python

```
df["isim_uzunlugu"] = df["urun_adi"].str.len()
df["kelime_sayisi"] = df["yorum"].str.split().str.len()
df["indirim_var"] = df["aciklama"].str.contains("indirim",
case=False).astype(int)
df["e_posta_domain"] = df["email"].str.split("@").str[1]
```

5) Etkileşim (interaction) özellikleri

İki özelliğin birlikte etkisini yakalar:

Python

```
df["yas_x_gelir"] = df["yas"] * df["gelir"]
```



7.2 Yeni özellik oluşturma (devam)

6) Polinom özellikler

Doğrusal bir modelin eğri şeklindeki ilişkileri öğrenmesini sağlar: x , x^2 , x^3 ...

Python

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(df[["a", "b"]])
print(poly.get_feature_names_out())
```

Çıktı

```
['a' 'b' 'a^2' 'a b' 'b^2']
```

Dikkat!

Polinom özellik sayısı çok hızlı artar! 10 özellik, 2. derece → 65 özellik; 3. derece → 285 özellik. Aşırı öğrenmeye (overfitting) dikkat!

7) Bayrak (flag) özellikleri

Evet/hayır şeklinde önemli durumları işaretler:

Python

```
df["hafta_sonu_mu"] = df["tarih"].dt.dayofweek.isin([5, 6]).astype(int)
df["gelir_eksik"] = df["gelir"].isnull().astype(int)
df["ilk_siparis"] = (df["siparis_sayisi"] == 1).astype(int)
```

Özellik oluştururken sorulacak sorular

- ★ Bu alanda bir uzman karar verirken neye bakar?
- ★ Oranlar, farklar, toplamalar anlamlı mı?
- ★ Zaman, konum ya da metin içinde gizli bilgi var mı?
- ★ Bu özellik tahmin anında gerçekten elimde olacak mı?



7.3 Özellik seçimi

Her özellik faydalı değildir. Gereksiz özellikler modeli **yavaşlatır**, **karmaşıktırır** ve **aşırı öğrenmeye** yol açabilir.

Tanım

Özellik seçimi (feature selection), modelin performansına en çok katkı sağlayan özellik alt kümesini belirleme işlemidir.

📌 Neden özellik seçmeliyiz?

- ★ Daha **basit** ve yorumlanabilir modeller
- ★ Daha **hızlı** eğitim ve tahmin
- ★ **Aşırı öğrenme** riskinin azalması
- ★ "Boyutun laneti" (curse of dimensionality) etkisinin azalması

📌 Üç ana yaklaşım

Yaklaşım	Nasıl çalışır?	Örnek
Filtre (filter)	İstatistiksel ölçüyle, modelden bağımsız puanlar	Korelasyon, varyans, χ^2 , ANOVA
Sarmalayıcı (wrapper)	Farklı alt kümelerle modeli tekrar tekrar dener	RFE, ileri/geri seçim
Gömülü (embedded)	Model eğitimi sırasında seçim yapılır	Lasso (L1), ağaç önem skorları

↳ Filtre: hızlı ama kaba · Wrapper: iyi ama yavaş · Gömülü: ikisinin ortası



7.3 Özellik seçimi (devam)

Filtre yöntemleri

1) **Düşük varyans:** Hemen her satırda aynı değere sahip sütunlar bilgi taşımaz.

Python

```
from sklearn.feature_selection import VarianceThreshold
sec = VarianceThreshold(threshold=0.01)
X_yeni = sec.fit_transform(X_train)
```

2) **Yüksek korelasyonlu özellik çiftleri:** İki özellik neredeyse aynı bilgiyi taşıyorsa biri silinebilir (ör. "boy_cm" ve "boy_inch").

Python

```
kor = X_train.corr().abs()
ust = kor.where(np.triu(np.ones(kor.shape), k=1).astype(bool))
silinecek = [s for s in ust.columns if any(ust[s] > 0.9)]
X_train = X_train.drop(columns=silinecek)
```

3) **Hedefle ilişki:** Her özelliğin hedefle ilişkisini ölçüp en iyi *k* tanesini seçmek:

Python

```
from sklearn.feature_selection import SelectKBest, f_classif
sec = SelectKBest(score_func=f_classif, k=5)
X_sec = sec.fit_transform(X_train, y_train)
print(X_train.columns[sec.get_support()])
```

Not

Sınıflandırma için `f_classif`, `chi2`, `mutual_info_classif`; regresyon için `f_regression`, `mutual_info_regression` kullanılır.



7.3 Özellik seçimi (devam)

🔪 Sarmalayıcı: RFE (Recursive Feature Elimination)

Modeli eğitir, en önemsiz özelliği atar, tekrar eğitir... İstenen sayıya ulaşana kadar devam eder.

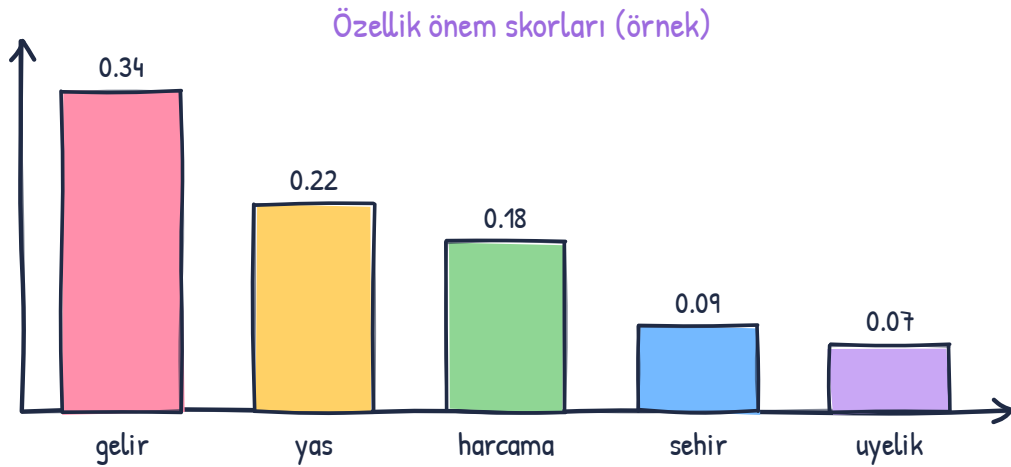
Python

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
rfe = RFE(LogisticRegression(max_iter=1000), n_features_to_select=5)
rfe.fit(X_train, y_train)
print(X_train.columns[rfe.support_])
```

🔪 Gömülü: Ağaç tabanlı önem skorları

Python

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=42).fit(X_train, y_train)
onem = pd.Series(rf.feature_importances_, index=X_train.columns)
onem.sort_values(ascending=False).plot(kind="barh")
```



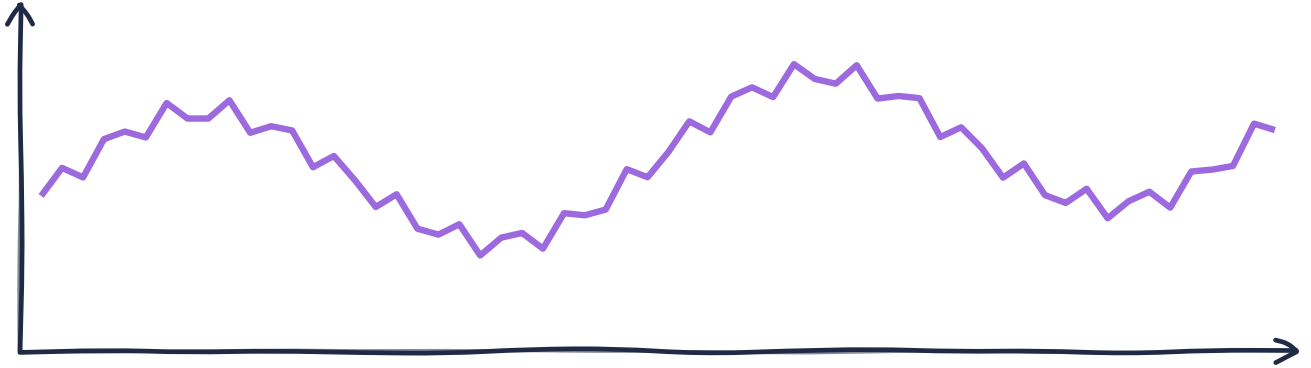
Dikkat!

Özellik seçimini tüm veri üzerinde yapıp sonra train/test'e bölmek bir veri sızıntısıdır! Seçim de train üzerinde yapılmalıdır.



7.4 Zaman serisi özellikleri

Tarih ve saat sütunları, doğru işlenirse **en zengin bilgi kaynaklarından** biridir. Ama "2026-09-27 14:35" hâliyle model için anlamsızdır.



Zaman serisi: trend + mevsimsellik

zaman serileri: trend ve mevsimsellik içerir

1) Tarih parçalarını çıkarma

Python

```
df["tarih"] = pd.to_datetime(df["tarih"])
df["yıl"] = df["tarih"].dt.year
df["ay"] = df["tarih"].dt.month
df["gun"] = df["tarih"].dt.day
df["haftanın_gunu"] = df["tarih"].dt.dayofweek # 0 = Pazartesi
df["saat"] = df["tarih"].dt.hour
df["ceyrek"] = df["tarih"].dt.quarter
df["hafta_sonu"] = (df["haftanın_gunu"] ≥ 5).astype(int)
```

2) Süre / fark özellikleri

Python

```
df["uyelik_gun"] = (pd.Timestamp("2026-09-27") -
df["kayit_tarihi"]).dt.days
df["yas"] = (df["tarih"] - df["dogum_tarihi"]).dt.days // 365
```



7.4 Zaman serisi özellikleri (devam)

3) Döngüsel (cyclical) kodlama

Saat 23 ile saat 0 aslında 1 saat arayla birbirine çok yakındır. Ama sayı olarak aralarında 23 fark var! Ay (12 → 1) ve haftanın günü için de aynı sorun geçerlidir.

Çözüm: değeri bir çember üzerine yerleştirmek, yani sinüs ve kosinüs kullanmak.

$$\text{saat_sin} = \sin(2\pi \cdot \text{saat} / 24) \quad \text{saat_cos} = \cos(2\pi \cdot \text{saat} / 24)$$

Python

```
df["saat_sin"] = np.sin(2 * np.pi * df["saat"] / 24)
df["saat_cos"] = np.cos(2 * np.pi * df["saat"] / 24)
df["ay_sin"]   = np.sin(2 * np.pi * df["ay"] / 12)
df["ay_cos"]   = np.cos(2 * np.pi * df["ay"] / 12)
```

↳ İki sütun birlikte her saati çember üzerinde benzersiz bir noktaya taşır.

4) Özel günler

Python

```
tatiller = pd.to_datetime(["2026-01-01", "2026-04-23", "2026-05-19",
                           "2026-10-29"])
df["tatil_mi"] = df["tarih"].isin(tatiller).astype(int)
```

İpucu

Bayram, okul tatili, maaş günü (ayın 15'i), kampanya dönemi gibi özellikler satış tahmininde çok etkilidir.



7.4 Zaman serisi özellikleri (devam)

5) Gecikme (lag) özellikleri

Geçmiş değerler, geleceğin en iyi tahmincisidir. "Dünkü satış", "geçen haftaki satış" gibi.

Python

```
df = df.sort_values("tarih")
df["satis_dun"] = df["satis"].shift(1)
df["satis_7gun_once"] = df["satis"].shift(7)
```

6) Hareketli (rolling) istatistikler

Python

```
df["satis_ort_7"] = df["satis"].shift(1).rolling(window=7).mean()
df["satis_std_7"] = df["satis"].shift(1).rolling(window=7).std()
```

tarih	satis	satis_dun	satis_ort_3
01.09	100	NaN	NaN
02.09	120	100	NaN
03.09	90	120	NaN
04.09	130	90	103.3

Dikkat!

Rolling hesaplamadan önce `shift(1)` yapmazsak, bugünün satışı kendi ortalamasına dahil olur. Bu, geleceği görmek demektir, yani **veri sızıntısı!**

Unutma!

Zaman serisinde veriyi **rastgele karıştırarak bölme!** Geçmiş → train, gelecek → test olmalı (Bölüm 9.3).



7.5 Uygulama örnekleri

Örnek 1: E-ticaret müşteri verisi

musteri	dogum	kayit	siparis	toplam_tutar
A	1995-03-12	2022-01-15	14	8.400
B	1980-11-02	2025-06-20	2	3.100
C	2001-07-25	2023-09-01	30	6.000

Python

```
bugun = pd.Timestamp("2026-09-27")
df["yas"] = (bugun - df["dogum"]).dt.days // 365
df["uyelik_gun"] = (bugun - df["kayit"]).dt.days
df["ort_sepet"] = df["toplam_tutar"] / df["siparis"]
df["aylik_siparis"] = df["siparis"] / (df["uyelik_gun"] / 30)
df["sadi_musteri"] = (df["siparis"] ≥ 10).astype(int)
```

Çıktı

	musteri	yas	uyelik_gun	ort_sepet	aylik_siparis	sadi_musteri
0	A	31	1716	600.0	0.24	1
1	B	45	464	1550.0	0.13	0
2	C	25	1122	200.0	0.80	1

Mini Soru

C müşterisi az harcıyor ama çok sık alışveriş yapıyor. Hangi yeni özellik bunu en iyi yakalar? → **aylik_siparis** (sıklık) ve **ort_sepet** (tutar) birlikte.



7.5 Uygulama örnekleri (devam)

Örnek 2: Ev fiyatı tahmini

Python

```
df["ev_yasi"] = 2026 - df["yapim_yili"]
df["oda_basi_alan"] = df["alan"] / df["oda"]
df["kat_orani"] = df["kat"] / df["bina_kat_sayisi"]
df["en_ust_kat"] = (df["kat"] == df["bina_kat_sayisi"]).astype(int)
df["yeni_bina"] = (df["ev_yasi"] ≤ 5).astype(int)
df["fiyat_log"] = np.log1p(df["fiyat"]) # hedef çarpık → log
```

Örnek 3: Günlük satış verisi

Python

```
df["tarih"] = pd.to_datetime(df["tarih"])
df = df.sort_values("tarih")
df["gun"] = df["tarih"].dt.dayofweek
df["gun_sin"] = np.sin(2*np.pi*df["gun"]/7)
df["gun_cos"] = np.cos(2*np.pi*df["gun"]/7)
df["lag_1"] = df["satis"].shift(1)
df["lag_7"] = df["satis"].shift(7)
df["ort_7"] = df["satis"].shift(1).rolling(7).mean()
df = df.dropna() # ilk 7 günün lag değerleri boş
```

Not

Lag ve rolling özellikleri, veri setinin başında kaçınılmaz olarak NaN üretir. Bu satırlar genelde silinir.



7.5 Uygulama örnekleri (devam)

Örnek 4: Özellik seçimi ile sadeleştirme

Python

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42).fit(X_train, y_train)
onem = pd.Series(rf.feature_importances_, index=X_train.columns)
print(onem.sort_values(ascending=False).head(6).round(3))

secilen = onem[onem > 0.02].index
X_train_s, X_test_s = X_train[secilen], X_test[secilen]
```

Çıktı

alan	0.412
oda_basi_alan	0.176
ev_yasi	0.121
semt_freke	0.094
kat_orani	0.061
yeni_bina	0.034

Oluşturduğumuz `oda_basi_alan` ve `ev_yasi` özellikleri, en önemli ilk 3'e girdi!

Özellik mühendisliği kontrol listesi

- ★ Tarih sütunlarını parçaladım mı?
- ★ Anlamlı oranlar ve farklar ekledim mi?
- ★ Çarpık sayısal sütunlara log uyguladım mı?
- ★ Gereksiz ve tekrarlı özellikleri çıkardım mı?
- ★ Hiçbir özellik geleceğe ait bilgi taşıyor mu?

Özet

Özellik mühendisliği = `oluştur + dönüştür + seç`. En iyi fikirler verinin geldiği alanı anlamaktan doğar!



8.1 Grafik türleri

"Bir grafik bin satıra bedeldir." Görselleştirme, sayılarda gizlenen desenleri, aykırıları ve ilişkileri **bir bakışta** görmemizi sağlar. Veri ön işlemenin hem başında (keşif) hem sonunda (kontrol) kullanılır.

Doğru grafiği seçmek

Amaç	Grafik
Tek sayısal değişkenin dağılımı	Histogram, KDE, kutu grafiği
Kategorilerin sayısı / karşılaştırma	Çubuk (bar) grafik
İki sayısal değişkenin ilişkisi	Dağılım (scatter) grafiği
Zamana göre değişim	Çizgi (line) grafiği
Çok değişkenin ilişkisi	Isı haritası, pairplot
Oranlar (az kategori)	Pasta grafik (dikkatli kullan)

Kullanacağımız kütüphaneler

Python

```
import matplotlib.pyplot as plt    # temel çizim kütüphanesi
import seaborn as sns             # istatistiksel, daha şık grafikler
sns.set_theme(style="whitegrid")
```

İpucu

Her grafiğe mutlaka **başlık**, **eksen etiketleri** ve gerekiyorsa **birim** ekle. Etiketsiz grafik, eksik cümle gibidir!



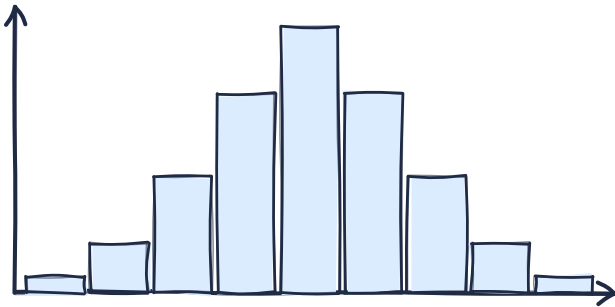
8.1 Grafik türleri (devam)

Histogram

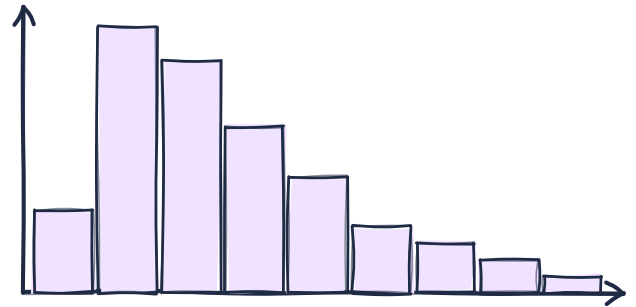
Sayısal bir değişkenin değerlerini aralıklara (bin) böler ve her aralıkta kaç gözlem olduğunu gösterir. **Dağılımın şeklini** görmenin en iyi yoludur.

Python

```
sns.histplot(df["yas"], bins=20, kde=True)
plt.title("Yaş dağılımı"); plt.xlabel("Yaş"); plt.show()
```



Normal (simetrik)



Sağa çarpık

Histogramda şunlara bak: Simetrik mi, çarpık mı? Tek tepeli mi, çok tepeli mi? Uçlarda yalnız çubuklar (aykırıları) var mı?

Kutu grafiği (boxplot)

Medyan, çeyrekler ve aykırı değerleri gösterir. Grupları **karşılaştırmak** için idealdir.

Python

```
sns.boxplot(data=df, x="şehir", y="gelir")
plt.title("Şehirlere göre gelir"); plt.show()
```



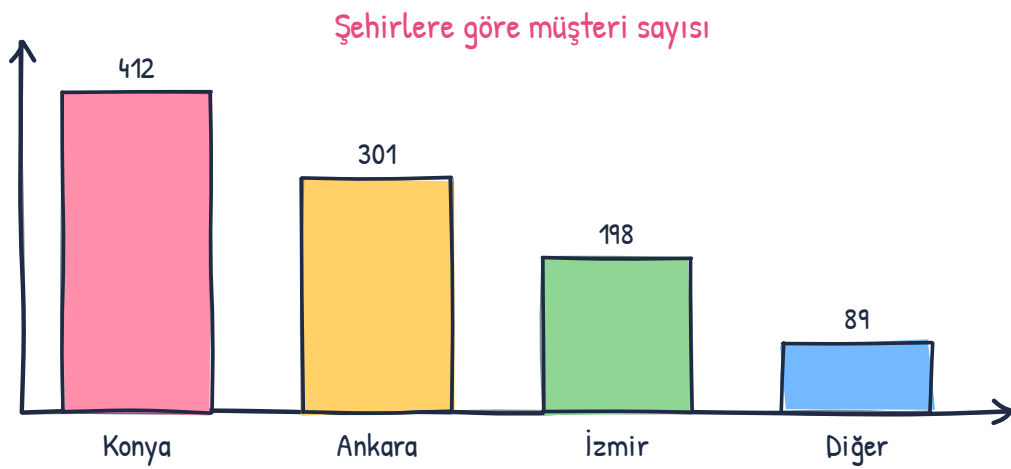
8.1 Grafik türleri (devam)

Çubuk grafik (bar chart)

Kategorik değişkenlerin frekanslarını veya grup ortalamalarını karşılaştırır.

Python

```
sns.countplot(data=df, x="şehir") # sayılar
sns.barplot(data=df, x="şehir", y="harcama") # ortalamalar
```



Dağılım grafiği (scatter plot)

İki sayısal değişken arasındaki ilişkiyi gösterir. Her nokta bir gözlemdir.

Python

```
sns.scatterplot(data=df, x="gelir", y="harcama", hue="cinsiyet")
```

Çizgi grafiği (line plot)

Zamana bağlı değişimi gösterir.

Python

```
df.groupby("ay")["satis"].sum().plot(marker="o")
```

Not

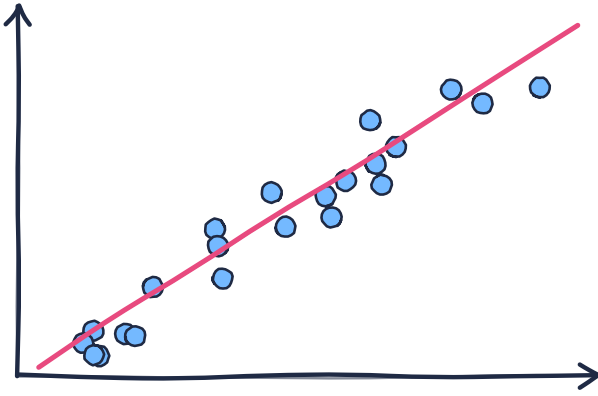
Pasta grafik çok sayıda dilimde okunması zor hâle gelir. 5'ten fazla kategori varsa **çubuk grafik** genelde daha iyidir.



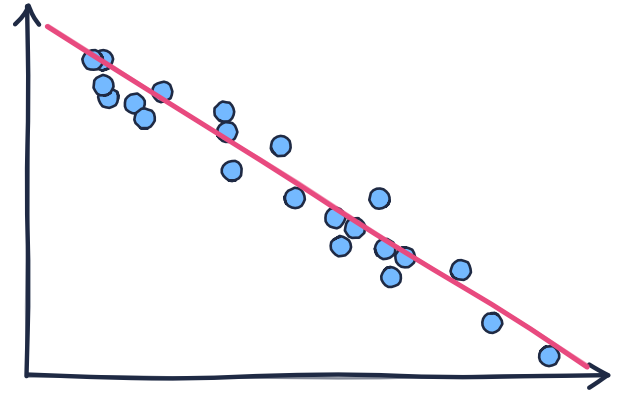
8.2 Korelasyon analizi

Tanım

Korelasyon, iki sayısal değişken arasındaki doğrusal ilişkinin yönünü ve gücünü ölçer. Korelasyon katsayısı (r) -1 ile +1 arasında değer alır.



Pozitif ilişki ($r \approx +0.9$)



Negatif ilişki ($r \approx -0.9$)

r değeri	Yorum
+0.7 ile +1.0	Güçlü pozitif: biri artınca diğeri de artar
+0.3 ile +0.7	Orta pozitif
-0.3 ile +0.3	Zayıf / ilişki yok
-0.7 ile -0.3	Orta negatif
-1.0 ile -0.7	Güçlü negatif: biri artınca diğeri azalır

↳ Örnek: Çalışma saati $\uparrow \rightarrow$ sınav notu $\uparrow$ (pozitif) .
Devamsızlık $\uparrow \rightarrow$ not $\downarrow$ (negatif)



8.2 Korelasyon analizi (devam)

✎ Korelasyon türleri

Yöntem	Ne ölçer?	Ne zaman?
Pearson	Doğrusal ilişki	Sayısal, normale yakın, aykırı az
Spearman	Sıralamaya dayalı (monoton) ilişki	Çarpık veri, aykırı var, ordinal veri
Kendall	Sıralama uyumu	Küçük veri setleri

Python

```
df["gelir"].corr(df["harcama"]) # iki sütun
(Pearson)
df.corr(numeric_only=True) # korelasyon matrisi
df.corr(method="spearman", numeric_only=True) # Spearman
df.corr(numeric_only=True)["harcama"].sort_values(ascending=False)
```

Çıktı

```
harcama      1.00
gelir        0.78
uyelik_gun   0.41
yas          0.12
iade_sayisi -0.35
```

✎ Korelasyon analizi ön işlemede ne işe yarar?

- ★ Hedefle yüksek korelasyonlu özellikleri bulmak (güçlü tahminçiler)
- ★ Birbiriyle çok yüksek korelasyonlu özellikleri tespit etmek (tekrarlı bilgi → biri çıkarılabilir)
- ★ Beklenmedik yüksek korelasyonlar → olası veri sızıntısı işareti!



8.2 Korelasyon analizi (devam)

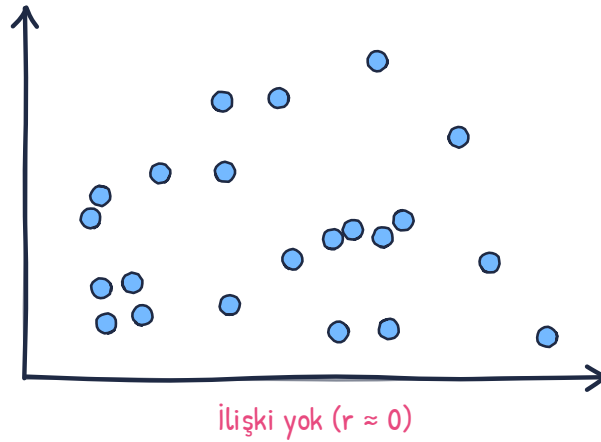
✎ Korelasyon ≠ Nedensellik!

Dikkat!

Yaz aylarında dondurma satışları ile boğulma vakaları arasında güçlü pozitif korelasyon vardır. Ama dondurma boğulmaya neden olmaz! İkisini de etkileyen üçüncü bir değişken var: sıcak hava.

✎ Korelasyonun görmediği ilişkiler

Pearson sadece doğrusal ilişkileri ölçer. U şeklindeki bir ilişkide $r \approx 0$ çıkabilir ama güçlü bir ilişki vardır!



$r \approx 0$, ama bu "hiç ilişki yok" demek değildir; mutlaka grafiğe bak!

✎ Tek bir aykırı değer korelasyonu değiştirebilir

Zayıf ilişkili 20 noktaya tek bir uç nokta eklendiğinde r değeri 0.1'den 0.7'ye çıkabilir. Bu yüzden:

1. Korelasyonu hesapla
2. Dağılım grafiğini çiz
3. Aykırı varsa Spearman ile karşılaştır

Unutma!

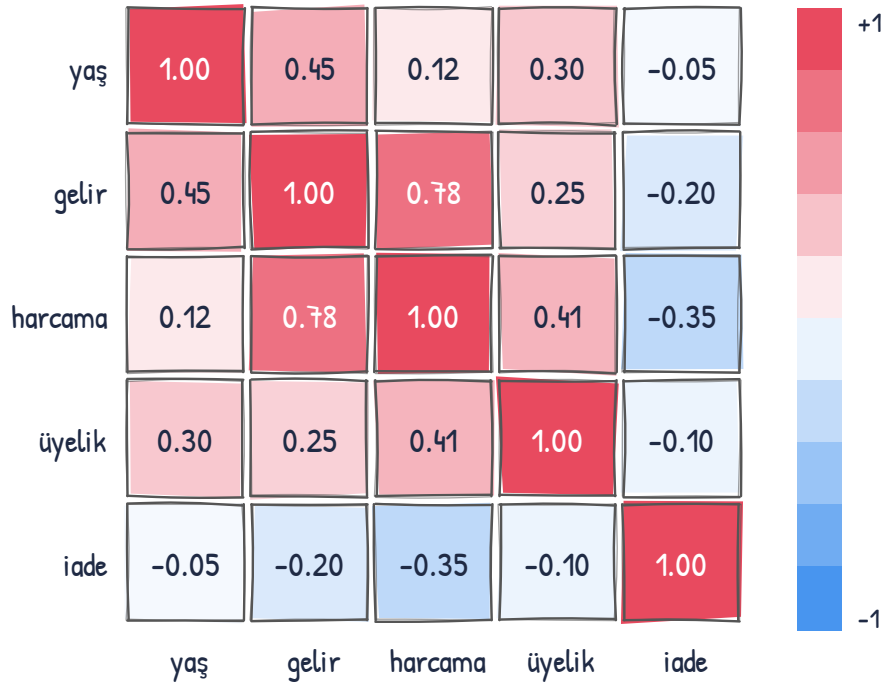
Sayıya güvenmeden önce grafiğe bak! (Anscombe dördlüsü: aynı r değerine sahip, tamamen farklı görünen 4 veri seti.)



8.3 Isı haritası (Heatmap)

Tanım

Isı haritası, bir matrisin değerlerini renklerle gösteren grafikdir. Veri ön işlemede en çok korelasyon matrisini ve eksik değerleri görselleştirmek için kullanılır.



kırmızı = pozitif, mavi = negatif korelasyon

Nasıl okunur?

- ★ Köşegen her zaman 1'dir (bir değişkenin kendisiyle korelasyonu).
- ★ Matris simetriktir: üst üçgen ile alt üçgen aynıdır.
- ★ Koyu kırmızı → güçlü pozitif, koyu mavi → güçlü negatif, beyaz → ilişki yok.
- ★ Bu örnekte en güçlü ilişki: gelir - harcama (0.78)



8.3 Isı haritası (Heatmap) (devam)

seaborn ile korelasyon ısı haritası

Python

```
kor = df.corr(numeric_only=True)
plt.figure(figsize=(8, 6))
sns.heatmap(kor,
             annot=True,           # hücelere sayıları yaz
             fmt=".2f",           # 2 ondalık basamak
             cmap="coolwarm",     # mavi-beyaz-kırmızı
             vmin=-1, vmax=1,     # renk ölçeğini sabitle
             center=0)
plt.title("Korelasyon Matrisi")
plt.show()
```

Sadece yarısını göstermek

Matris simetrik olduğu için üst üçgeni gizlemek okumayı kolaylaştırır:

Python

```
maske = np.triu(np.ones_like(kor, dtype=bool))
sns.heatmap(kor, mask=maske, annot=True, cmap="coolwarm", vmin=-1,
            vmax=1)
```

İpucu

`vmin=-1, vmax=1` yazmazsan renk skalası verideki en küçük ve en büyük değere göre ayarlanır; zayıf korelasyonlar bile **koyu** renkte görünüp yanıltabilir.



8.3 Isı haritası (Heatmap) (devam)

Eksik değer ısı haritası

Python

```
sns.heatmap(df.isnull(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("Eksik değerler (sarı = eksik)")
```

Sütunlarda dikey sarı şeritler görürsen o sütunda çok eksik vardır; farklı sütunlarda aynı satırlarda şeritler varsa eksiklikler birbiriyle ilişkilidir.

Pivot tablo ısı haritası

Isı haritası sadece korelasyon için değil, iki kategorik değişkene göre özetlenmiş herhangi bir sayısal değer için de kullanılır:

Python

```
tablo = df.pivot_table(values="satis", index="gun", columns="saat",
aggfunc="mean")
sns.heatmap(tablo, cmap="YlOrRd")
plt.title("Gün ve saate göre ortalama satış")
```

↳ "Cuma akşamları satış patlıyor!" gibi desenleri anında görürsün.

Pairplot: tüm ikili ilişkiler tek seferde

Python

```
sns.pairplot(df[["yas", "gelir", "harcama", "segment"]], hue="segment")
```

Not

Pairplot çok değişkenli verilerde çok yavaşlar. 4-6 önemli sütunla sınırlı tut.



8.4 Uygulama örnekleri

Örnek 1: Hızlı EDA paneli

Python

```
fig, ax = plt.subplots(2, 2, figsize=(12, 9))
sns.histplot(df["gelir"], kde=True, ax=ax[0, 0])
ax[0, 0].set_title("Gelir dağılımı")
sns.boxplot(data=df, x="sehir", y="harcama", ax=ax[0, 1])
ax[0, 1].set_title("Şehirlere göre harcama")
sns.scatterplot(data=df, x="gelir", y="harcama", ax=ax[1, 0])
ax[1, 0].set_title("Gelir - harcama ilişkisi")
sns.countplot(data=df, x="uyelik", ax=ax[1, 1])
ax[1, 1].set_title("Üyelik türleri")
plt.tight_layout(); plt.show()
```

Bu panelden ne öğrenebiliriz?

- ★ Gelir **sağa çarpık** → log dönüşümü düşünülmeli (Bölüm 5)
- ★ Bazı şehirlerde harcamada **aykırıları** var (Bölüm 4)
- ★ Gelir ve harcama arasında **pozitif ilişki** var ($r \approx 0.78$)
- ★ Üyelik türleri **dengesiz** dağılmış → train/test bölmede **stratify** (Bölüm 9)

İpucu

`plt.subplots` ile birden fazla grafiği tek görselde toplamak, raporlarda ve sunumlarda çok profesyonel görünür.

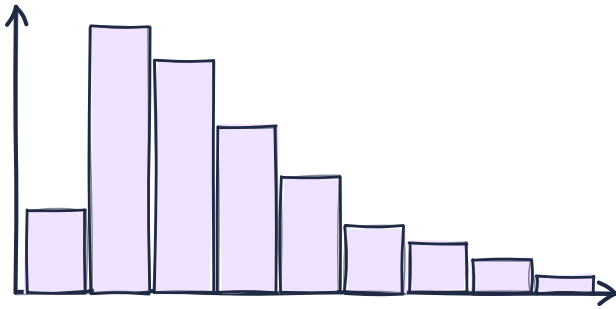


8.4 Uygulama örnekleri (devam)

Örnek 2: Ön işleme öncesi / sonrası karşılaştırması

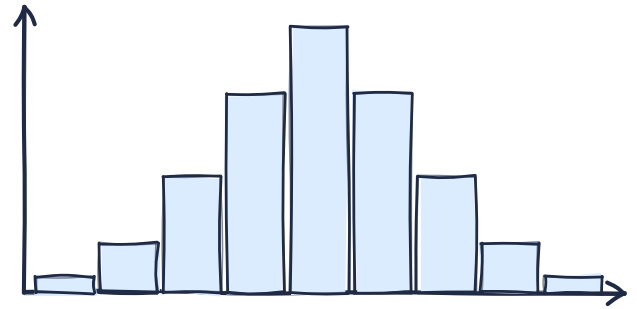
Python

```
fig, ax = plt.subplots(1, 2, figsize=(11, 4))
sns.histplot(df["gelir"], ax=ax[0]).set_title("Önce: ham gelir")
sns.histplot(np.log1p(df["gelir"]), ax=ax[1]).set_title("Sonra: log(gelir)")
plt.show()
```



Sağa çarpık

önce



Normal (simetrik)

sonra

Örnek 3: Hedefe göre özellik dağılımı

Bir özelliğin hedefi ayırt etme gücünü görmek için:

Python

```
sns.kdeplot(data=df, x="uyelik_gun", hue="ayrildi_mi", fill=True)
plt.title("Ayrılan ve kalan müşterilerde üyelik süresi")
```

↳ İki eğri birbirinden ne kadar ayırıksa, özellik o kadar ayırt edici!

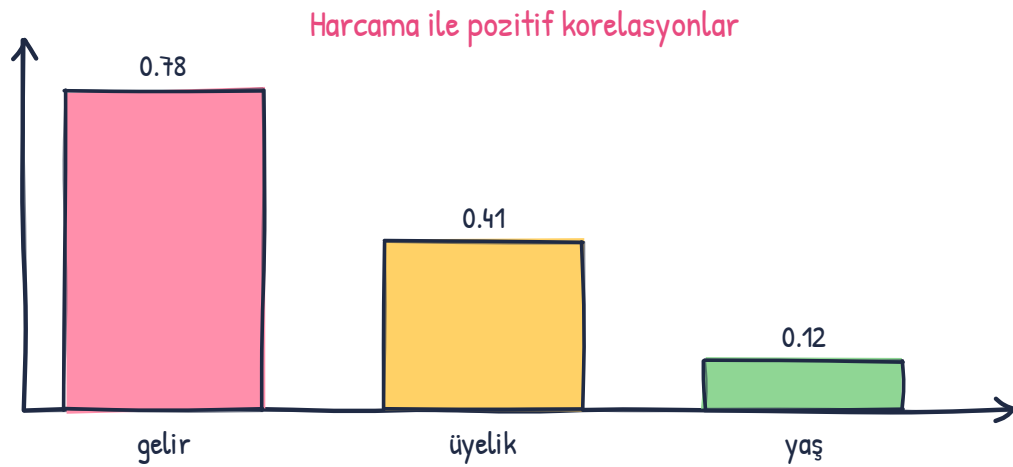


8.4 Uygulama örnekleri (devam)

Örnek 4: Hedefle en ilişkili özellikler

Python

```
hedef_kor = df.corr(numeric_only=True)["harcama"].drop("harcama")
hedef_kor.sort_values().plot(kind="barh", color="orchid")
plt.title("Harcama ile korelasyon"); plt.axvline(0, color="black")
plt.show()
```

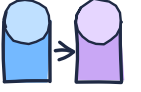


Görselleştirme için altın kurallar

1. Bir grafik = bir mesaj. Grafik neyi göstermeli?
2. Eksenleri, başlığı ve birimleri etiketle.
3. Renkleri anlamlı kullan; gereksiz renk karmaşasından kaçın.
4. Eksenleri keserek farkları abartma.
5. Çıkardığın sonucu grafiğin altına bir cümleyle yaz.

Özet

Histogram → dağılım · Boxplot → aykırı ve karşılaştırma · Scatter → ilişki · Heatmap → korelasyon ve eksikler. Korelasyon nedensellik değildir; sayıya güvenmeden önce grafiğe bak!



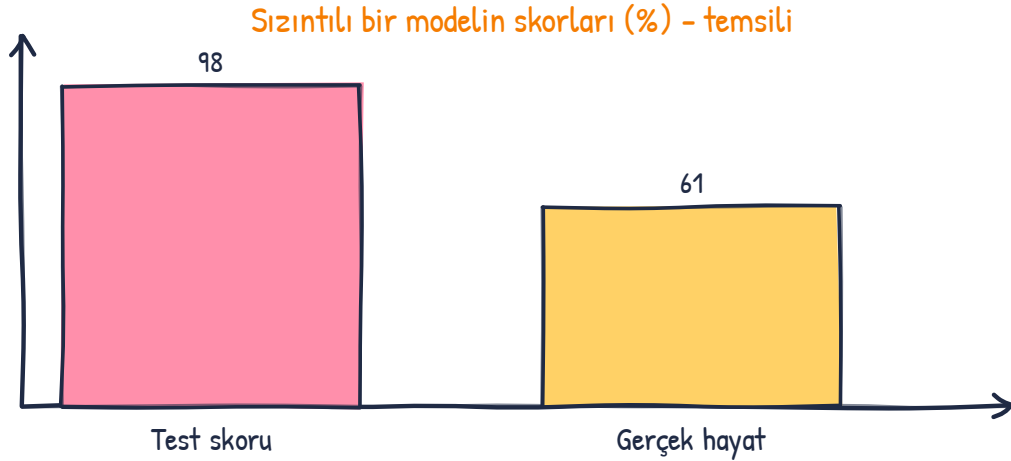
9.1 Veri sızıntısı nedir?

Tanım

Veri sızıntısı (data leakage), model eğitilirken tahmin anında gerçekte erişilemeyecek bilgilerin modele sızmasıdır. Sonuç: eğitimde muhteşem, gerçek hayatta hayal kırıklığı yaratan bir model.

✏ Bir benzetme: kopya çeken öğrenci

Sınavdan önce cevap anahtarını gören öğrenci 100 alır. Ama konuyu gerçekten öğrenmemiştir; bir sonraki sınavda başarısız olur. Sızıntılı model de cevap anahtarını görmüş bir öğrencidir.

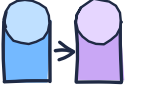


✏ Neden bu kadar tehlikeli?

- ★ Fark etmesi çok zordur: kod hata vermez, skor mükemmel görünür.
- ★ Yanlış kararlar alınır: "Modelimiz %98 başarılı!" diye canlıya alınır.
- ★ Zaman, para ve güven kaybına yol açar.

Unutma!

Test sonucun "gerçek olamayacak kadar iyi" görünüyorsa, ilk şüphelen gereken şey veri sızıntısıdır!



9.1 Veri sızıntısı nedir? (devam)

İki ana sızıntı türü

Hedef sızıntısı (target leakage)

Özelliklerin arasında hedefin kendisini veya sonucunu içeren bir sütun vardır.

↳ "Kredi ödenmedi mi?" tahmininde
"icra_takibi_başladı" sütunu

Train-test kirlenmesi

Test verisinin bilgisi, ön işleme sırasında eğitime karışır.

↳ Ölçekleyicinin tüm veri üzerinde fit edilmesi

Hedef sızıntısına örnek

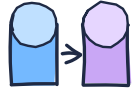
Bir hastanede "hasta zatürre mi?" sorusunu tahmin eden bir model:

yas	ates	antibiyotik_aldi	zature
65	38.9	1	1
34	36.8	0	0
71	39.2	1	1

antibiyotik_aldi sütunu hedefi neredeyse mükemmel tahmin eder. Ama antibiyotik, teşhis konulduktan sonra verilir! Yeni gelen bir hastada bu bilgi henüz yoktur.

Mini Soru

Bir özelliğin sızıntı olup olmadığını anlamak için en önemli soru nedir? → "Bu bilgi, tahmin yapacağım anda elimde olacak mı?"



9.2 Nerelerde oluşur?

Veri sızıntısı genellikle iyi niyetli ama yanlış sırada yapılan işlemlerden kaynaklanır.

1) Bölmeden önce ön işleme yapmak

En yaygın hata! Ortalama, medyan, min-max, standart sapma gibi değerler **tüm veri** üzerinden hesaplanırsa, test setinin bilgisi eğitime sızar.

Python

```
# YANLIŞ ✗
X_s = StandardScaler().fit_transform(X)           # tüm veri
X_train, X_test = train_test_split(X_s, ...)

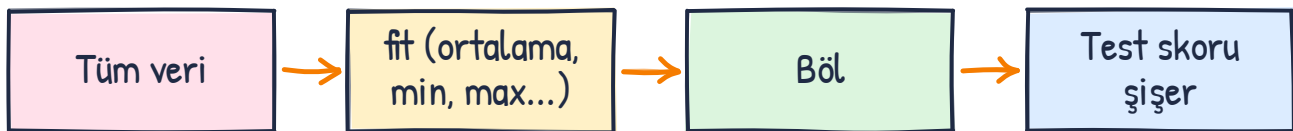
# DOĞRU ✓
X_train, X_test = train_test_split(X, ...)
sc = StandardScaler().fit(X_train)                 # sadece train
X_train_s, X_test_s = sc.transform(X_train), sc.transform(X_test)
```

2) Eksik veri doldurma ve aykırı sınırları

Doldurma ortalaması veya IQR sınırları tüm veriden hesaplanırsa aynı sorun oluşur.

3) Özellik seçimini tüm veride yapmak

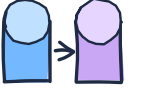
Hedefle korelasyona bakarak özellik seçimi yapıp sonra bölmek de sızıntıdır; seçim test verisini "görmüştür."



yanlış sıra: test bilgisi eğitime sızar

Unutma!

Doğru sıra her zaman: **Böl → train'de fit → train ve test'te transform.**



9.2 Nerelerde oluşur? (devam)

4) Target encoding'i tüm veride yapmak

Kategorinin hedef ortalaması test satırlarının hedefini de içerir → model testte hile yapar.

5) Zamanı karıştırmak

Zaman serisinde veriyi rastgele bölmek, modelin gelecekte öğrenip geçmişi tahmin etmesine yol açar. Ayrıca shift kullanılmadan hesaplanan rolling özellikler de bugünü içerir.

6) Tekrarlı ve ilişkili kayıtlar

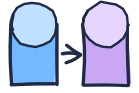
Aynı hastanın birden fazla ölçümü hem train'de hem test'te varsa, model hastayı ezberler. Test başarısı yapay olarak yükselir.

7) Gelecek bilgisi içeren sütunlar

Tahmin edilen	Sızıntılı özellik
Müşteri ayrılacak mı?	iptal_tarihi, son_fatura_iadesi
Kredi ödenecek mi?	gecikme_faizi, icra_durumu
Öğrenci geçecek mi?	final_notu, harf_notu
Satış miktarı	toplam_ciro (satışın kendisinden hesaplanmış)

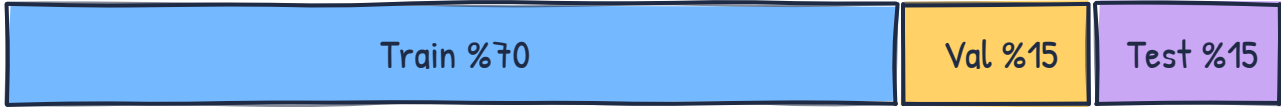
Dikkat!

Bir özelliğin önem skoru diğerlerinden çok yüksekse (ör. 0.95), o sütunu dikkatle incele. Çoğu zaman bir sızıntı sütunudur!



9.3 Doğru bölme stratejileri

Modeli, hiç görmediği veride test etmeliyiz. Bunun için veri setini bölüyoruz.



← tüm veri seti →

Val = doğrulama (validation) kümesi

Küme	Görevi
Eğitim (train)	Model burada öğrenir; ön işleme de burada fit edilir
Doğrulama (validation)	Model ve ayar (hiperparametre) seçimi
Test	Sadece en sonda, <u>bir kez</u> : gerçek performans tahmini

Basit bölme: train_test_split

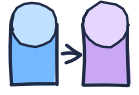
Python

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,           # %20 test
    random_state=42,         # tekrarlanabilir sonuç
    shuffle=True)
```

İpucu

`random_state` sabitlenmezse kod her çalıştığında farklı bir bölme olur ve sonuçlar değişir. Tekrarlanabilirlik için hep bir sayı ver!

↳ Yaygın oranlar: 80/20, 70/30, 70/15/15



9.3 Doğru bölme stratejileri (devam)

📌 Tabakalı (stratified) bölme

Sınıflar **dengesizse** (ör. %95 sağlıklı, %5 hasta), rastgele bölme sonucunda test setinde hiç hasta kalmayabilir! **stratify** her kümede sınıf oranlarını korur.

Python

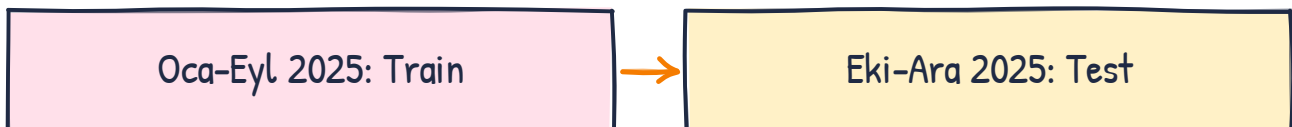
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
print(y_train.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))    # oranlar aynı!
```

📌 Zaman serisi bölme

Geçmiş → eğitim, gelecek → test. **Asla karıştırma!**

Python

```
df = df.sort_values("tarih")
sinir = int(len(df) * 0.8)
train, test = df.iloc[:sinir], df.iloc[sinir:]
```

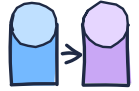


📌 Grup bazlı bölme

Aynı kişiye/kaynağa ait kayıtların hepsi **ya train'de ya test'te** olmalı:

Python

```
from sklearn.model_selection import GroupShuffleSplit
gss = GroupShuffleSplit(test_size=0.2, random_state=42)
tr_idx, te_idx = next(gss.split(X, y, groups=df["hasta_id"]))
```



9.3 Doğru bölme stratejileri (devam)

Çapraz doğrulama (cross-validation)

Tek bir bölme şansa bağlı olabilir. **K-katlı çapraz doğrulama**, veriyi k parçaya böler; her seferinde bir parçayı test, diğerlerini eğitim olarak kullanır ve k skorun ortalamasını alır.

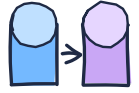
Tur	Kat 1	Kat 2	Kat 3	Kat 4	Kat 5
1	Test	Train	Train	Train	Train
2	Train	Test	Train	Train	Train
3	Train	Train	Test	Train	Train
4	Train	Train	Train	Test	Train
5	Train	Train	Train	Train	Test

Python

```
from sklearn.model_selection import cross_val_score, StratifiedKFold,
TimeSeriesSplit
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
skorlar = cross_val_score(model, X_train, y_train, cv=cv)
print(skorlar.mean(), skorlar.std())
# Zaman serisi için: cv = TimeSeriesSplit(n_splits=5)
```

Dikkat!

Çapraz doğrulamada ön işleme her katta **ayrı ayrı** fit edilmelidir. Bunu elle yapmak zordur; çözüm: **Pipeline!** (9.4)



9.4 Uygulama örnekleri

Örnek 1: Sızıntılı ve sızıntısız kod

Python

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# X YANLIŞ: önce ön işleme, sonra bölme
X_dolu = SimpleImputer().fit_transform(X)
X_olcek = StandardScaler().fit_transform(X_dolu)
X_tr, X_te, y_tr, y_te = train_test_split(X_olcek, y, random_state=42)

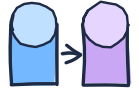
# ✓ DOĞRU: önce bölme, sonra sadece train'den öğrenme
X_tr, X_te, y_tr, y_te = train_test_split(X, y, random_state=42)
imp = SimpleImputer().fit(X_tr)
sc = StandardScaler().fit(imp.transform(X_tr))
X_tr_h = sc.transform(imp.transform(X_tr))
X_te_h = sc.transform(imp.transform(X_te))
```

Not

Basit ölçekleme gibi işlemlerde sızıntının etkisi küçük olabilir. Ama target encoding, özellik seçimi ve hedef sızıntısında fark **çok büyük** olur. Doğru alışkanlığı baştan kazan!

Neden fark eder?

	Yanlış yol	Doğru yol
Imputer ortalaması	Tüm veriden	Sadece X_tr'den
Scaler μ ve σ	Test'i de içerir	Test'i hiç görmez
Test skoru	İyimser (yanıltıcı)	Gerçekçi



9.4 Uygulama örnekleri (devam)

🍷 Örnek 2: Pipeline + ColumnTransformer = sızıntıya karşı kalkan

Python

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier

sayisal = ["yas", "gelir", "harcama"]
kategorik = ["sehir", "uyelik"]

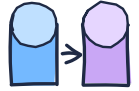
sayi_hatti = Pipeline([("doldur", SimpleImputer(strategy="median")),
                        ("olcek", StandardScaler())])
kat_hatti = Pipeline([("doldur", SimpleImputer(strategy="most_frequent")),
                        ("kodla", OneHotEncoder(handle_unknown="ignore"))])

on_isleme = ColumnTransformer([("num", sayi_hatti, sayisal),
                                ("cat", kat_hatti, kategorik)])

model = Pipeline([("on_isleme", on_isleme),
                   ("model", RandomForestClassifier(random_state=42))])
```



Pipeline içinde her adım sadece train'den öğrenir



9.4 Uygulama örnekleri (devam)

Örnek 2 (devam): eğitim ve değerlendirme

Python

```
X_tr, X_te, y_tr, y_te = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)

skorlar = cross_val_score(model, X_tr, y_tr, cv=5)  # her katta ayrı
fit!
print("CV ortalaması:", skorlar.mean().round(3))

model.fit(X_tr, y_tr)
print("Test skoru   :", round(model.score(X_te, y_te), 3))
```

Çıktı

```
CV ortalaması: 0.862
Test skoru   : 0.857
```

↳ CV ve test skorlarının birbirine yakın olması, sağlıklı bir modelin işaretidir.

Veri sızıntısı kontrol listesi

- ★ Veriyi ön işlemeden önce böldüm mü?
- ★ Tüm fit işlemleri sadece train üzerinde mi?
- ★ Tahmin anında elimde olmayacak sütunları çıkardım mı?
- ★ Zaman serisinde kronolojik bölme yaptım mı?
- ★ Aynı kişi/grup hem train'de hem test'te var mı?
- ★ Test setine sadece en sonda, bir kez baktım mı?

Özet

Önce böl, sonra öğren! Test seti kasadaki altın gibidir: en sona kadar dokunma. Pipeline kullanmak en güvenli yoldur.



10.1 Gerçek veri seti ile çalışma

Öğrendiğimiz her şeyi gerçek bir veri setinde birleştirme zamanı! Veri bilimin en ünlü "ilk projesi": **Titanic yolcu verisi**.

Örnek

Görev: 1912'de batan Titanic gemisindeki bir yolcunun bilgilerine (yaş, cinsiyet, bilet sınıfı...) bakarak **hayatta kalıp kalmadığını** tahmin etmek. Hedef değişken: **survived** (1 = kurtuldu, 0 = kurtulamadı).

Python

```
import pandas as pd, numpy as np
import seaborn as sns
df = sns.load_dataset("titanic")
print(df.shape)
df.head()
```

Çıktı

(891, 15)

Sütunlar

Sütun	Anlamı	Tür
survived	Hayatta kaldı mı? (hedef)	İkili
pclass	Bilet sınıfı (1, 2, 3)	Ordinal
sex	Cinsiyet	Nominal / ikili
age	Yaş	Sayısal (sürekli)
sibsp / parch	Gemideki kardeş-eş / ebeveyn-çocuk sayısı	Sayısal (kesikli)
fare	Bilet ücreti	Sayısal (sürekli)
embarked	Biniş limanı (S, C, Q)	Nominal
deck	Kabin güvertesi	Nominal
alive, class, who, embark_town...	Diğer sütunlar	? (dikkat!)



10.1 Gerçek veri seti ile çalışma (devam)

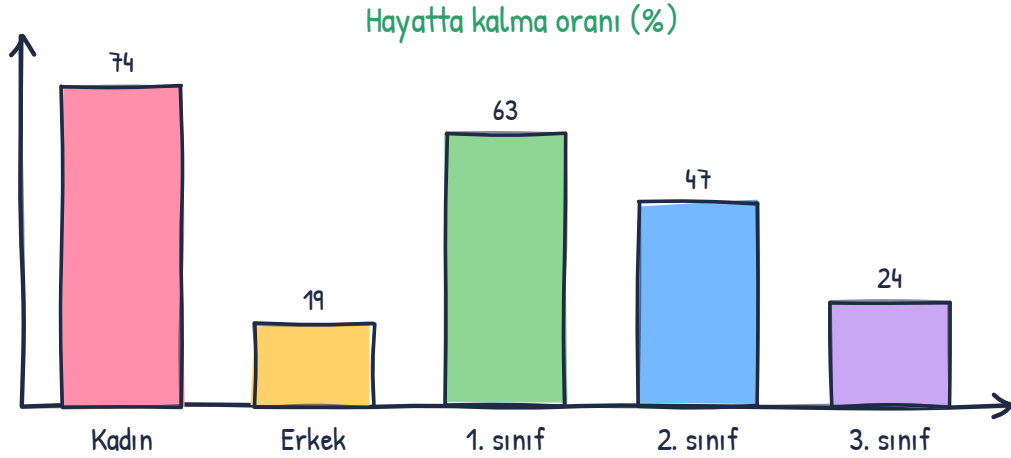
EDA: Veriyi tanıyalım (Bölüm 2)

Python

```
df.isnull().sum()[lambda s: s > 0]
df["survived"].mean()
df.groupby("sex")["survived"].mean()
df.groupby("pclass")["survived"].mean()
```

Çıktı

```
age          177      # %19.9 eksik
embarked      2
deck         688      # %77.2 eksik!
embark_town    2
survived oranı: 0.384
sex:   female 0.74 | male 0.19
pclass: 1 → 0.63 | 2 → 0.47 | 3 → 0.24
```



İlk gözlemler

- ★ Yolcuların sadece %38'i hayatta kalmış → sınıflar dengesiz, **stratify** kullanmalıyız.
- ★ **Cinsiyet** ve **bilet sınıfı** çok güçlü tahminçiler ("Önce kadınlar ve çocuklar!").
- ★ **deck** sütununun %77'si boş; **age** sütununun %20'si boş.

Mini Soru

`df.duplicated().sum()` 107 sonucunu veriyor. Bunları silmeli miyiz? → Bu veri setinde isim/ID sütunu yok; aynı özelliklere sahip **farklı yolcular** olabilir. Kimlik bilgisi olmadan silmek, gerçek insanları veriden atmak olur!



10.2 Adım adım veri ön işleme

Adım 1: Sızıntı ve tekrar sütunlarını temizle (Bölüm 9)

Sütun	Sorun	Karar
alive	"yes/no" → hedefin kopyası!	Sil (sızıntı)
class	pclass ile aynı bilgi	Sil
embark_town	embarked ile aynı bilgi	Sil
who, adult_male	sex + age'den türetilmiş	Sil (kendimiz üreteceğiz)
deck	%77 eksik	Sil

Python

```
df = df.drop(columns=["alive", "class", "embark_town", "who",
"adult_male", "deck"])
```

Dikkat!

alive sütunu modelde kalsaydı ne olurdu? Test doğruluğu **%100** çıkar! Ama bu model yeni bir yolcu için hiçbir işe yaramaz, çünkü "hayatta mı?" bilgisi zaten tahmin etmeye çalıştığımız şeydir.

Python

```
# Denedik: alive sütunu ile lojistik regresyon
print(model_sizintili.score(X_test, y_test))
```

Çıktı

```
1.0      # "Gerçek olamayacak kadar iyi" = sızıntı!
```



10.2 Adım adım veri ön işleme (devam)

Adım 2: Özellik mühendisliği (Bölüm 7)

Python

```
df["aile"] = df["sibsp"] + df["parch"] + 1      # kendisi dahil aile büyüklüğü
df["yalniz"] = (df["aile"] == 1).astype(int)     # yalnız ile aynı bilgi
df = df.drop(columns=["alone"])
```

Adım 3: Aykırı değerleri incele (Bölüm 4)

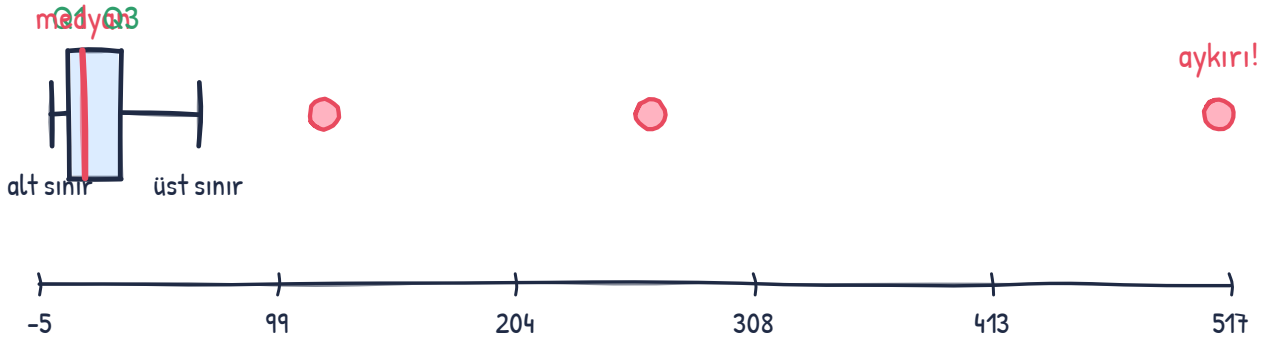
Python

```
df["fare"].describe()[["min", "50%", "max"]]
```

Çıktı

```
min      0.00
50%     14.45
max     512.33
```

Medyan bilet 14.45 iken 512.33 gerçekten çok uç bir değer. Ama bu **gerçek** bir lüks bilet! Silmiyoruz; ağaç modelleri buna dayanıklıdır. Doğrusal model için log dönüşümü seçeneği not edilir.



fare sütunu: sağa çarpık, çok sayıda gerçek aykırı

Adım 4: Bölme - her şeyden önce! (Bölüm 9)

Python

```
from sklearn.model_selection import train_test_split
X = df.drop(columns="survived")
y = df["survived"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```



10.2 Adım adım veri ön işleme (devam)

Adım 5: Ön işleme hattını kur (Bölüm 3, 5, 6)

Sütunlar	Eksik veri	Dönüşüm
age, fare, aile, sibsp, parch	Medyan ile doldur	StandardScaler
sex, embarked, pclass, yalnız	En sık değerle doldur	OneHotEncoder

Python

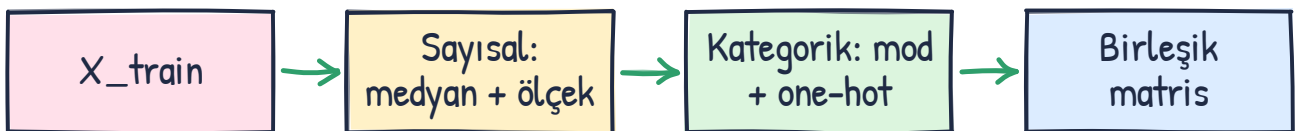
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

sayisal = ["age", "fare", "aile", "sibsp", "parch"]
kategorik = ["sex", "embarked", "pclass", "yalniz"]

on_isleme = ColumnTransformer([
    ("num", Pipeline([("doldur", SimpleImputer(strategy="median")),
                     ("olcek", StandardScaler())])), sayisal),
    ("cat", Pipeline([
        ("doldur", SimpleImputer(strategy="most_frequent")),
        ("kodla", OneHotEncoder(handle_unknown="ignore"))]), kategorik)])
```

Not

`pclass` sayısal görünse de (1, 2, 3) aslında bir **kategoridir**. Burada one-hot kodladık; ordinal olarak bırakmak da savunulabilir bir seçimdir. İkisini de dene!



ColumnTransformer iki hattı paralel çalıştırıp sonuçları yan yana birleştirir

İpucu

`on_isleme.fit_transform(X_train).shape` ile ön işleme sonrası kaç sütun olduğunu kontrol edebilirsiniz.



10.2 Adım adım veri ön işleme (devam)

Adım 6: Model, çapraz doğrulama ve test

Python

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

model = Pipeline([("on_isleme", on_isleme),
                  ("model", LogisticRegression(max_iter=1000))])

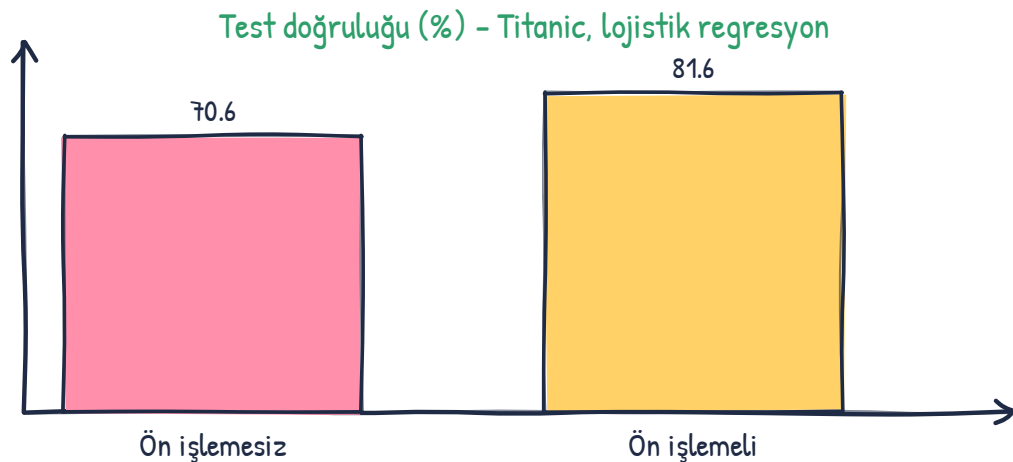
cv = cross_val_score(model, X_train, y_train, cv=5)
model.fit(X_train, y_train)
print("CV   :", cv.mean().round(3))
print("Test :", round(model.score(X_test, y_test), 3))
```

Çıktı

CV : 0.794
Test : 0.816

Ön işlemenin katkısı

Karşılaştırma için sadece sayısal sütunları alıp eksik yaşa sahip satırları sildiğimiz **ön işlemesiz** bir model kurduk (891 → 714 satır):



↳ Aynı algoritma, sadece veriyi doğru hazırlayarak ~11 puan daha iyi!

İpucu

Sonraki adımlar: **fare** için log dönüşümü, isimden unvan çıkarma (Mr, Mrs, Miss), yaş grupları... Her yeni fikri çapraz doğrulama ile **ölçerek** dene.



10.3 Sonuç ve anahtar kavramlar



veri ön işlemenin büyük resmi

Kavram	Tek cümlede
EDA	Veriye dokunmadan önce onu tanımak
Eksik veri	Tespit et, nedenini düşün, sil ya da doldur (+ bayrak)
Aykırı değer	Hata mı gerçek mi? Sil, düzelt, baskıla ya da koru
Normalizasyon	Min-Max ile [0, 1] aralığı
Standardizasyon	Z-score ile ortalama 0, std 1
One-Hot / Label	Nominal → One-Hot, hedef → Label, sıralı → Ordinal
Özellik mühendisliği	Alan bilgisiyle yeni ve anlamlı sütunlar üret
Korelasyon	-1 ile +1 arası doğrusal ilişki; nedensellik değil!
Veri sızıntısı	Tahmin anında olmayacak bilginin modele girmesi
Pipeline	Tüm adımları sızıntısız, tekrarlanabilir hâle getirir

Özet

Temiz veri = doğru sonuç. Önce veriyi anla, sonra böl, sadece train'den öğren ve her kararını ölçerek ver. Tebrikler, artık veri ön işlemenin temellerini biliyorsun!

↳ Veri + Doğru İşlem = Daha İyi Sonuçlar! 😊