Dr.Öğr.Üyesi Furkan Göz

2025

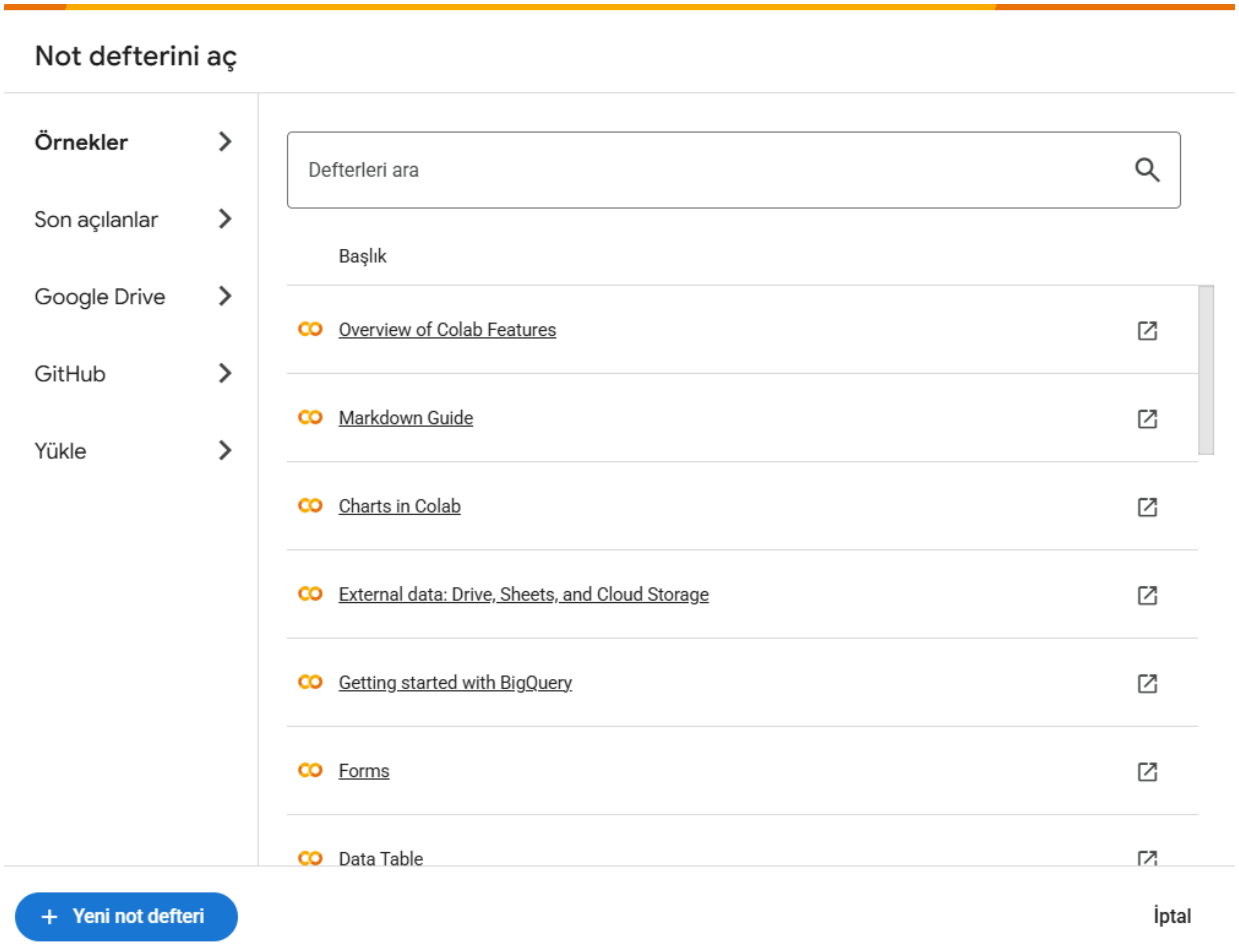
Atölyeye Hoş Geldiniz

- Python ile temel kodlama
- Veri türleri ve tip dönüşümleri
- Koşullar ve döngüler
- Basit yapay zeka projelerinin temelleri
- Çalışma ortamımız: Google Colab

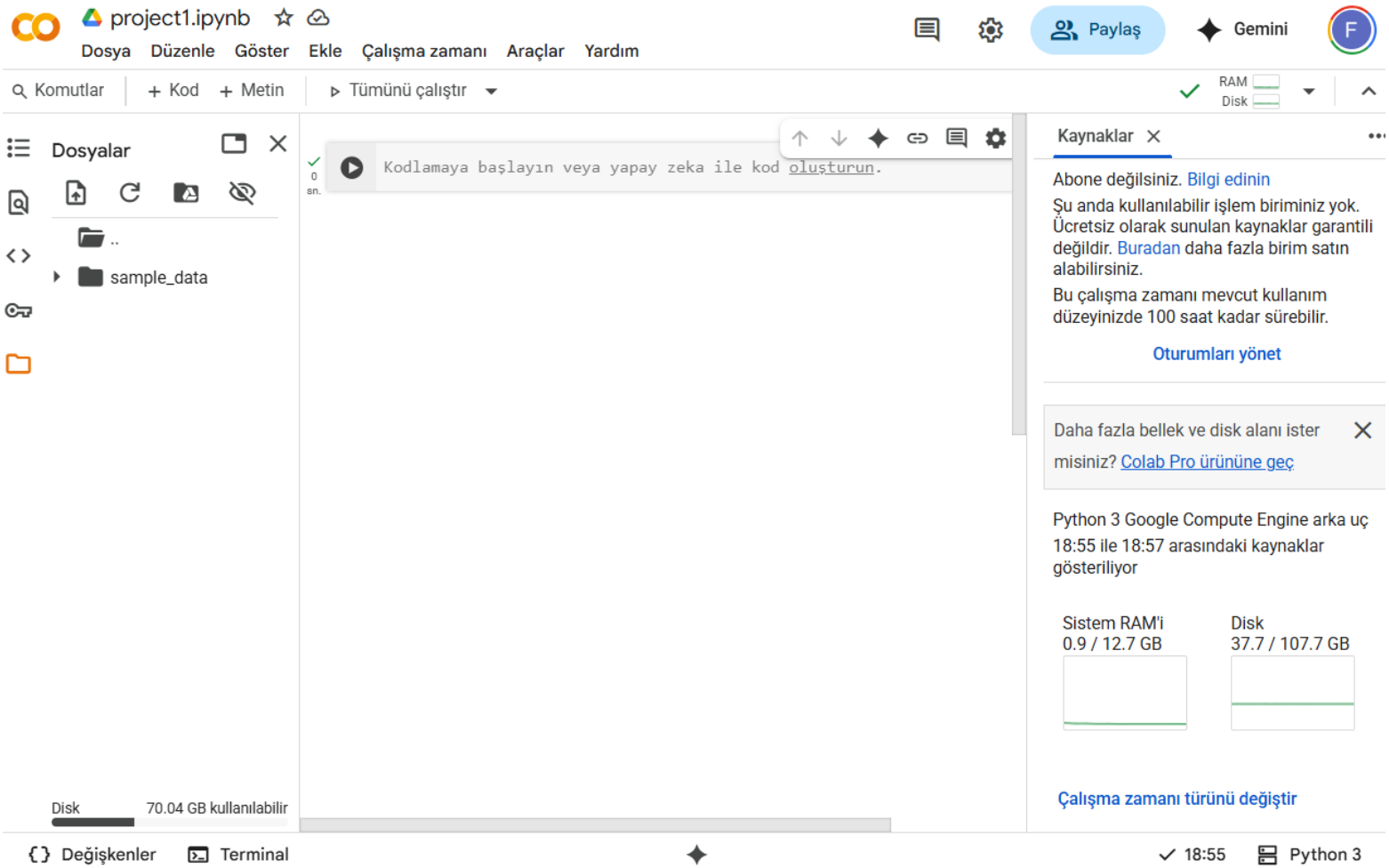
Google Colab Nedir?

- Tarayıcıda çalışan bir Python editörüdür
  → Bilgisayar farkı gözetmeksizin kullanılabilir
- Bilgisayara kurulum gerekmez
  → Hemen çalışmaya başlayabilirsiniz
- Kod hücreleriyle çalışılır (Shift + Enter)
  → Kod ve metin içeriği aynı yerde yazılır
- Google Drive ile entegre çalışır
  → Dosyalarınızı kaydedebilir, paylaşabilirsiniz
- colab.research.google.com

Google Colab

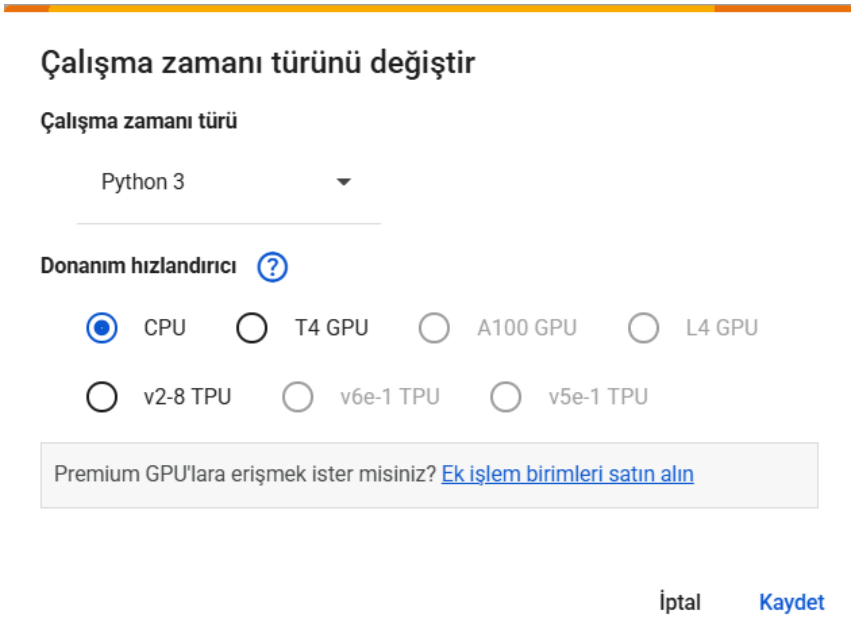


Google Colab



Çalışma Zamanı Türünü Değiştirme

- Donanım hızlandırıcısı (GPU/TPU) kullanmak için:
  - Menüden `Çalışma Zamanı > Çalışma Zamanı Türünü Değiştir` yolunu izleyin
  - Python sürümünü ve hızlandırıcıyı seçin (CPU, GPU, TPU)
- Derin öğrenme gibi yoğun işlemler için GPU önerilir
- Ayarlar sonrası yeni çalışma zamanı başlatılır



İlk Python Kodu

```
1 print("Merhaba, Yapay Zeka!")
```

- `print()` komutu ekrana çıktı verir.
- Python'da ilk çalışan satır genellikle budur.
- Shift + Enter ile çalıştırabilirsiniz.

Değişkenler ve Veri Tipleri

```python
sayi = 42          # int
oran = 3.14        # float
isim = "Ayse"      # str
aktif = True       # bool
liste = [1, 2, 3] # list
```

- Python değişkenlerde tür belirtmez, otomatik algılar.
- Temel veri tipleri: `int`, `float`, `str`, `bool`, `list`
- Yorum satırı: `#` ile başlar.

Değişken Tanımlama

```
# Bir sehir adi (str)
# Dogum yili (int)
# Ortalama puan (float)
# Uye mi? (bool)
# Ilgi alanlari listesi (list)
```

str → metin, int → tam sayı, float → ondalık, bool → True/False, list → köşeli parantezle diziler

Tip Dönüşümleri

```python
x = "5"
y = int(x)          # → 5
z = float("2.5")    # → 2.5
isim = str(42)      # → "42"
```

- Metin ⇄ sayı dönüşümleri mümkündür
- `int("abc")` → Hata verir
- Dönüştürmeden önce veri tipi kontrol edilmeli

Koşullu İfadeler

```
1 temperature = 32
2 if temperature > 30:
3     print("Too hot!")
4 elif temperature > 20:
5     print("Warm weather")
6 else:
7     print("Cool or cold")
```

- `if / elif / else` yapısı
- Duruma göre farklı çıktılar üretir

Kullanıcıdan Veri Almak

```
1 giris = input("Sayilari girin: ")
2 liste = giris.split(",")
3 sayilar = [float(x) for x in liste]
4 print("Ortalama:", sum(sayilar)/len(sayilar))
```

- `input()` ile kullanıcıdan veri alınır
- `split(",")` ile listeye dönüştürülür
- Liste üzerinde matematiksel işlem yapılabilir

Sıcaklık Yorumu

```
1 temperature = float(input("Enter temperature: "))
2
3 if temperature < 0:
4     print("Below freezing!")
5 elif temperature < 20:
6     print("Cool day")
7 else:
8     print("Warm or hot")
```

BMI Hesaplama

```
1  boy = float(input("Boy (metre): "))
2  kilo = float(input("Kilo (kg): "))
3  bmi = kilo / (boy ** 2)
4
5  if bmi < 18.5:
6      print("Zayif")
7  elif bmi < 25:
8      print("Normal")
9  elif bmi < 30:
10     print("Fazla kilolu")
11 else:
12     print("Obez")
```

for Döngüsü

```
1 fruits = ["apple", "banana", "strawberry"]
2 for fruit in fruits:
3     print(fruit)
```

- Listedeki her eleman için işlem yapılır
- Dizi, liste, aralık vb. üzerinde çalışır

Uygulama: Çift / Tek Sayı Kontrolü

```python
for i in range(11):
    if i % 2 == 0:
        print(i, "is even")
    else:
        print(i, "is odd")
```

- `range(11)` → 0'dan 10'a kadar sayılar üretir
- `%` operatörü: mod alma, yani kalanı bulma işlemidir
- `i % 2 == 0` → sayı çiftse doğru olur

while Döngüsü

```
1 i = 0
2 while i < 3:
3     print(i)
4     i += 1
```

- Koşul doğruysa döngü devam eder
- Sayaç değişkeni dikkatle yönetilmeli

Fonksiyon Tanımlama

- Kodu tekrar yazmak yerine, fonksiyonlarla modülerleştiririz.
- Girdi alabilir, çıktı döndürebilir.
- `def` anahtar kelimesi ile tanımlanır.

Basit Bir Fonksiyon

```
1 def selamla(isim):
2     print("Merhaba,", isim)
3
4 selamla("Ayse")
```

- Fonksiyon çağrıldığında belirli bir işi yapar.
- Parametre alabilir (örnek: `isim`).

Çıktı Döndüren Fonksiyon

```
1 def ortalama(liste):
2     return sum(liste) / len(liste)
3
4 puanlar = [80, 90, 75]
5 print("Ortalama:", ortalama(puanlar))
```

- `return` ile çıktı geri döndürülür.
- Fonksiyon başka işlemler için tekrar kullanılabilir.

Hatalı Girdiyle Başa Çıkma

```python
try:
    sayi = int(input("Bir ısay girin: "))
    print("Karesi:", sayi ** 2)
except ValueError:
    print("Lütfen geçerli bir tam ısay girin!")
```

- `try` bloğu: Hata oluşabilecek kod.
- `except` bloğu: Hata oluşursa ne yapılacağını belirtir.

Dosya Okuma Hataları

```python
import pandas as pd
try:
    df = pd.read_csv("veri.csv")
    print(df.head())
except FileNotFoundError:
    print("Dosya bulunamadi. Lutfen yolunu kontrol edin.")
```

- Dosya işlemleri sırasında sıkça kullanılır.
- Programın durması engellenir, kullanıcı bilgilendirilir.

- Python'da 4 temel koleksiyon tipi vardır:
  - List – sıralı, değiştirilebilir
  - Tuple – sıralı, değiştirilemez
  - Set – benzersiz elemanlar, sırasız
  - Dict – anahtar-değer çiftleri
- Veri işleme ve analizde bu yapılar sıkça kullanılır

List, Tuple, Set, Dict Örnekleri

```python
meyveler = ["elma", "muz", "çilek"]  # list
koordinat = (10, 20)                 # tuple
renkler = {"kirmizi", "mavi"}        # set
ogrenci = {"ad": "Ali", "yas": 21}   # dict
```

- `list`: Değiştirilebilir, sıralı
- `tuple`: Sabit yapılar
- `set`: Tekrarsız veri
- `dict`: Etiketlenmiş veri (anahtar/değer)

Sözlük (Dictionary) Kullanımı

```python
ogrenci = {"ad": "Zeynep", "not": 85}

print(ogrenci["ad"])        # Zeynep
ogrenci["not"] += 5         # Notu ıartr
print(ogrenci["not"])       # 90
```

- Anahtarlar string veya sayısal olabilir.
- Veri analizi sırasında JSON ve tablo verileri sözlüklerle temsil edilir.

- NumPy dizileri hızlı ve verimli hesaplamalar sağlar.
- Vektör, matris ve çok boyutlu dizilerle çalışılır.
- Rastgele sayı üretimi, istatistiksel işlemler ve matris çarpımı yapılabilir.

NumPy ile Dizi Oluşturma

```
1 import numpy as np
2
3 dizi = np.array([1, 2, 3, 4])
4 print(dizi)
5
6 aralik = np.arange(0, 10, 2)
7 print(aralik)
```

- `np.array()` ile liste dizisine dönüştürme
- `np.arange(start, stop, step)` ile sayı üretme

NumPy ile Sayısal İşlemler

```
1 a = np.array([1, 2, 3])
2 b = np.array([4, 5, 6])
3
4 toplam = a + b
5 ortalama = np.mean(toplam)
6 print(ortalama)
```

- Diziler arasında doğrudan matematiksel işlem yapılabilir
- `np.mean()` ile ortalama alınabilir

Rastgele Sayılar: Temel Üretim

```python
import numpy as np

print("Tam sayi:", np.random.randint(0, 10, size=3))  # 0-9 arasi 3 adet
print("0-1 arasi:", np.random.rand(3))                # [0,1) arasi
print("Normal dagilim:", np.random.randn(3))
```

- `randint()` → belirtilen aralıkta tamsayı üretir.
- `rand()` → 0 ile 1 arasında eş dağılımlı sayı üretir.
- `randn()` → normal dağılıma göre rastgele sayı üretir.

Rastgele Seçim ve Karıştırma

```python
import numpy as np

meyveler = ["elma", "armut", "muz"]
print("Rastgele secim:", np.random.choice(meyveler))

a = np.array([1, 2, 3, 4])
np.random.shuffle(a)
print("Karistirilmis:", a)
```

- `choice()` → bir listeden rastgele seçim yapar.
- `shuffle()` → dizinin elemanlarını yerinde karıştırır.
- Simülasyon ve test senaryoları için sıkça kullanılır.

Temel İstatistiksel İşlemler

```
1 veri = np.array([10, 20, 30, 40])
2
3 print("Toplam:", np.sum(veri))
4 print("Ortalama:", np.mean(veri))
5 print("Std Sapma:", np.std(veri))
```

- `sum`, `mean`, `std` gibi fonksiyonlar hızlı hesaplama sağlar.
- İstatistiksel analizlerin temeli atılır.

Matris Çarpımı ve Şekil

```
1 a = np.array([[1, 2], [3, 4]])
2 b = np.array([[5, 6], [7, 8]])
3
4 carpim = np.dot(a, b)
5 print(carpim)
```

- `dot()` ile matris çarpımı yapılır.
- Derin öğrenme gibi alanlarda temel yapı taşlarındandır.

NumPy: reshape ile Şekil Değiştirme

```
1 import numpy as np
2 a = np.arange(6)
3 print(a.reshape((2, 3)))
```

- `reshape()` diziyi 2 boyutlu hale getirir

Pandas ile Veri Çerçevesi

```python
import pandas as pd

veri = {
  "isim": ["Ayse", "Ali", "Zeynep"],
  "yas": [23, 34, 29]
}
df = pd.DataFrame(veri)
print(df)
```

- `pd.DataFrame` tablo benzeri veri yapısı oluşturur
- Sözlük yapısı → satır/sütun yapısına dönüştürülür

Pandas: Kategorik Sıklık Analizi

```
df = pd.DataFrame({
    "kategori": ["A", "B", "A", "C", "B", "A"]
})

print(df["kategori"].value_counts())
```

- `value_counts()` → Kategorilerin sıklığını verir
- Görselleştirme öncesi anlamlı içgörü sağlar

Pandas ile Filtreleme

```
1 yasli = df[df["yas"] > 30]
2 print(yasli)
```

- Sadece belirli koşulu sağlayan satırlar seçilebilir
- `df[df["sütun"] > değer]` yapısı kullanılır

Grup Bazlı Analiz

```
veri = {
  "cinsiyet": ["K", "E", "K", "E"],
  "puan": [85, 70, 90, 60]
}
df = pd.DataFrame(veri)
grup = df.groupby("cinsiyet")["puan"].mean()
print(grup)
```

- `groupby()` ile veriler gruplanabilir
- Her grup için ortalama gibi işlemler yapılabilir

- Matplotlib ile grafiklerde başlık, etiket, renk, stil ekleyebiliriz.
- Seaborn ile estetik ve anlamlı görseller kolayca oluşturulur.
- Grafikler veriyi daha anlaşılır ve etkili hale getirir.

Matplotlib ile Grafik Özelleştirme

```
1 import matplotlib.pyplot as plt
2 gunler = ["Pzt", "Sal", "Car"]
3 satis = [100, 150, 90]
4 plt.plot(gunler, satis, marker='o', color='green',
      linestyle='--')
5 plt.title("Gunlük ışSatlar")
6 plt.xlabel("Gun")
7 plt.ylabel("Satis")
8 plt.grid(True)
9 plt.show()
```

- `marker`, `color`, `linestyle` ile stil verilebilir. Başlık ve eksen etiketleri okunabilirliği artırır.

```python
import seaborn as sns
import pandas as pd

df = pd.DataFrame({
    "kategori": ["A", "B", "A", "C", "B", "A"]
})

sns.countplot(x="kategori", data=df)
```

- `countplot` → kategorik verilerin sıklığını görselleştirir.

Satış Verisi Analizi

```
data = {
  "urun": ["Kalem", "Defter", "Kalem", "Silgi"],
  "adet": [10, 5, 8, 3],
  "fiyat": [2.5, 5.0, 2.5, 1.0]
}


df = pd.DataFrame(data)
df["toplam"] = df["adet"] * df["fiyat"]

print(df.groupby("urun")["toplam"].sum())
```

- Veri çerçevesi oluşturma, Yeni sütun hesaplama, Ürün bazında toplam satış analizi

Eksik Verilerle Çalışmak: Giriş

- Gerçek veri setlerinde eksik gözlemler çok yaygındır.
- Nedenleri:
  - Katılımcı bazı soruları boş bırakmış olabilir.
  - Sensörler hata verip ölçüm yapmamış olabilir.
  - Veri aktarımı sırasında bilgi kaybolmuş olabilir.
- Eksik veriler analiz sonuçlarını bozabilir, dikkatle ele alınmalıdır.

Eksik Verilerle Çalışmak

- Gerçek veri setlerinde eksik değerler kaçınılmazdır.
- `isna()`, `fillna()`, `dropna()` ile eksik veriler analiz edilir.
- Eksik verileri silmek yerine, ortalama veya medyan gibi stratejilerle doldurmak daha etkilidir.
- Görsel analiz için ısı haritası (heatmap) kullanılabilir.

Eksik Verileri Bulma ve Saymak

```python
import pandas as pd

df = pd.DataFrame({
    "ad": ["Ali", "Ayse", "Mehmet", "Zeynep"],
    "yas": [25, None, 40, None],
    "puan": [85, 90, None, 75]
})

print(df.isna())
print(df.isna().sum())
```

- `isna()` → Boolean tablo: True = eksik
- `sum()` → eksik değer sayısını verir

Eksik Değeri Ortalama ile Doldurma

```
1 ortalama = df["yas"].mean()
2 df["yas"] = df["yas"].fillna(ortalama)
3 print(df)
```

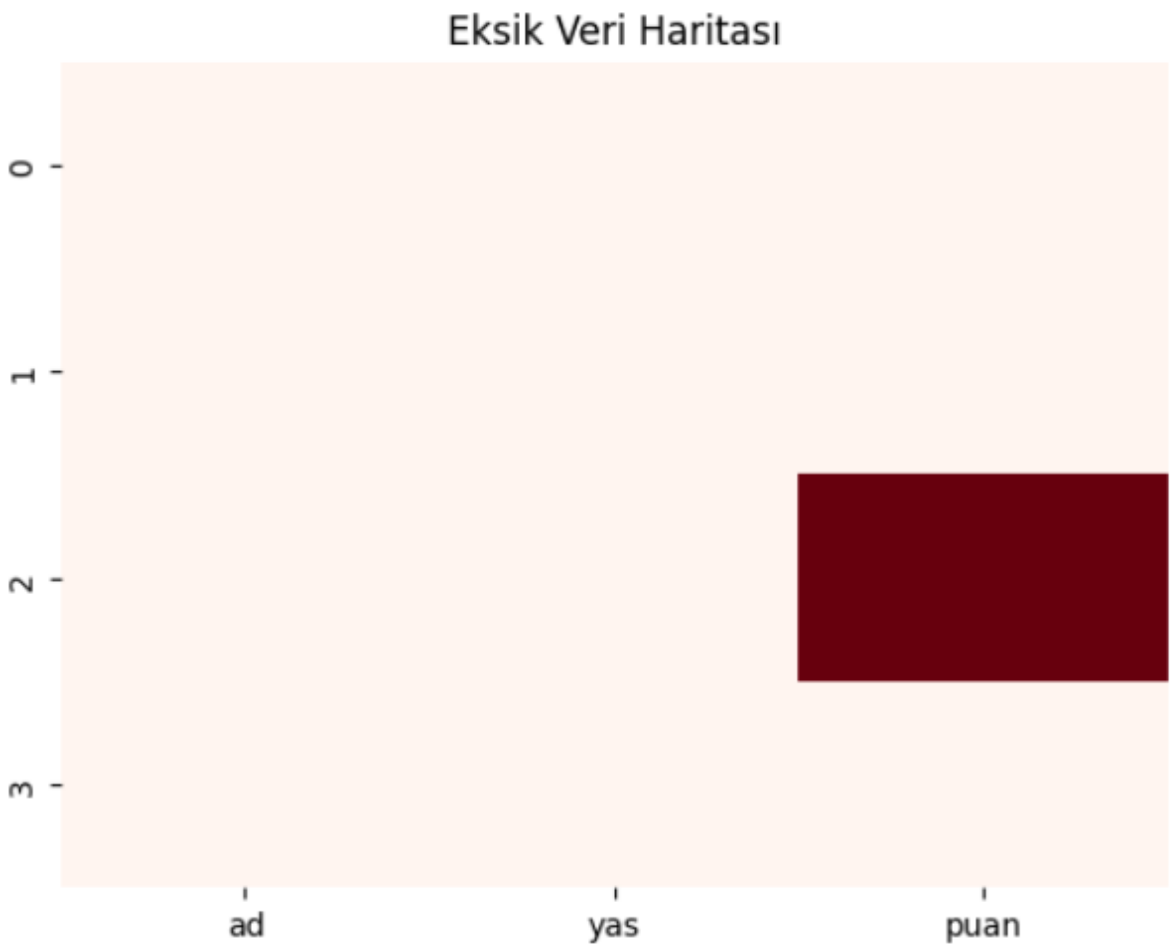
- `fillna()` ile boş hücreler doldurulur.
- Ortalama, medyan veya sabit bir değer kullanılabilir.

Eksik Verilerin Görselleştirilmesi

```
1 import seaborn as sns
2 import matplotlib.pyplot as plt
3
4 sns.heatmap(df.isna(), cbar=False, cmap="Reds")
5 plt.title("Eksik Veri Haritasi")
6 plt.show()
```

- Isı haritası ile hangi satır/sütunlarda eksik veri olduğu kolayca görülür.
- Özellikle büyük veri setlerinde çok faydalıdır.

Eksik Verilerin Görselleştirilmesi



Koşullu Filtreleme

```
df = pd.DataFrame({
  "ad": ["Ali", "Zeynep", "Ayse"],
  "yas": [25, 22, 30],
  "puan": [80, 90, 75]
})

secim = df[(df["yas"] > 23) & (df["puan"] > 70)]
print(secim)
```

- Birden fazla koşul kullanılabilir: `&`, `|`
- Parantez kullanımı zorunludur!

Grafik Türlerini Seçmek

- Bar (çubuk): Kategorik veriler (ürün satışları)
- Line (çizgi): Zaman serileri (günlük sıcaklık)
- Histogram (hist): Sayısal dağılım (puan dağılımı)

Bar ve Çizgi Grafiği Karşılaştırması

```
df = pd.DataFrame({
  "gun": ["Pzt", "Sal", "Çar"],
  "satis": [100, 150, 90]
})
df.plot(kind="bar", x="gun", y="satis")
df.plot(kind="line", x="gun", y="satis")
```

Bar Grafiği

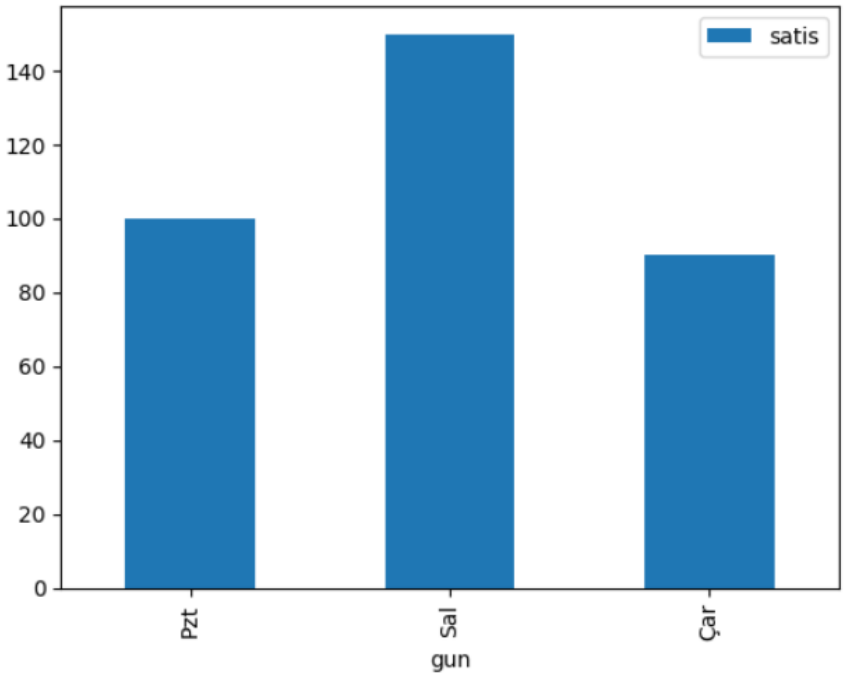
- Kategorik veriler için uygundur
- Ürün, şehir, gün gibi sınıflar
- Değerler ayrı ayrı çubukla

Çizgi Grafiği

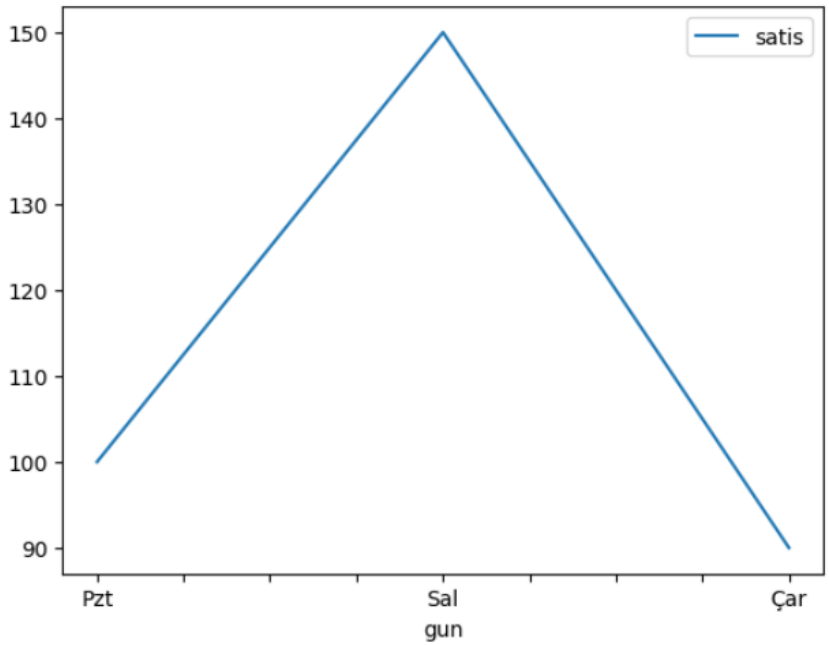
- Zamanla değişimi gösterir
- Trend ve artış/azalış analizleri
- Noktalar bir çizgiyle birleştirilir

Bar ve Çizgi Grafik Çıktıları

Bar Grafiği



Çizgi Grafiği

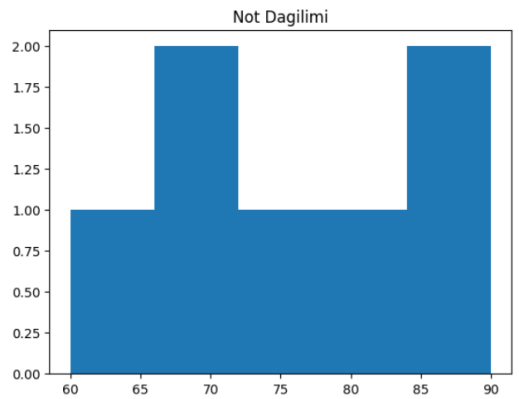


- Bar grafikte her günün satış değeri ayrı çubukla gösterilir
- Çizgi grafik, satışların trendini daha net ortaya koyar

Histogram ile Dağılım Analizi

```python
import matplotlib.pyplot as plt
veriler = [60, 70, 80, 85, 90, 70, 75]
plt.hist(veriler, bins=5)
plt.title("Not Dagilimi")
plt.show()
```

- Histogram → Sayısal verilerin dağılımını gösterir
- `bins`: Kaç aralıkta gruplandırılacağı



Veri Formatlarını Anlamak

- CSV ve JSON dosyaları veri bilimi projelerinde en sık kullanılan formatlardır.
- Gerçek projelerde karşılaşılan zorluklar:
  - Farklı ayırıcılar (örneğin: ; veya |)
  - Dosya karakter kodlamaları (UTF-8, ISO-8859-9)
  - JSON dosyalarında iç içe geçmiş yapılar (nested)

CSV Dosyasında Farklı Ayırıcı ve Encoding

```python
df = pd.read_csv("veri.csv", sep=";", encoding="utf-8")
print(df.head())
```

- `sep=";"` → Noktalı virgülle ayrılmış dosyalar için
- `encoding="utf-8"` → Türkçe karakterler için yaygın

CSV Dosyası Okuma

```
1 df = pd.read_csv("veri.csv")
2 print(df.head())
```

- CSV: Virgülle ayrılmış tablo dosyası
- `read_csv()` ile kolayca yüklenir
- İlk satırlar için `head()` fonksiyonu kullanılır

İç İçe JSON Yapısı

```
1 json_data = [
2   {"ad": "Ali", "adres": {"sehir": "Ankara", "posta":
      6000}},
3   {"ad": "Zeynep", "adres": {"sehir": "İzmir", "posta":
      3500}}
4 ]
```

- Bu tür veri `pd.read_json()` ile doğrudan düzgün yapılamaz.
- Nested yapılar `json_normalize()` ile düzenlenmelidir.

Nested JSON Düzleştirme

```
1 import pandas as pd
2 from pandas import json_normalize
3
4 df = json_normalize(json_data)
5 print(df)
```

- `json_normalize()` → iç içe geçmiş JSON'u tablolara dönüştürür.
- Kolon isimleri: `adres.sehir`, `adres.posta` gibi olur.

JSON Verisi ile Çalışmak

```
1 df = pd.read_json("veri.json")
2 print(df)
```

- JSON: Sözlük benzeri veri formatı (anahtar-değer)
- Genellikle API'lerden veri çekerken kullanılır
- `read_json()` ile yüklenebilir

JSON Yapısı Örneği

```
[
  {"ad": "Ayse", "yas": 25},
  {"ad": "Ali", "yas": 30}
]
```

- JSON yapısı: Liste içinde sözlükler
- Her sözlük bir satırı temsil eder

Analitik Düşünme

- Veriyi sadece görselleştirmek değil, anlamlandırmak da önemlidir.
- Satış, kategori ya da zaman trendleri üzerine yorum yapılmalı.
- Öğrenci çıkarım yapabilmeli: ”Neden bu gün daha çok satıldı?”
- Bu beceri veri okuryazarlığı ve yapay zeka için kritik önemdedir.

Veriyi Gözlemleyip Soru Sorma

```python
import pandas as pd

df = pd.read_csv("satislar.csv")
df["toplam"] = df["miktar"] * df["birim_fiyat"]

kategori_sat = df.groupby("kategori")["toplam"].sum()
print(kategori_sat)
```

- Soru: En çok satış yapılan kategori neden bu kadar yüksek?
- O kategoriye özel kampanya mı vardı? Daha pahalı ürünler mi?

Günlük Satış Trendine Yorum Katma

```python
df["tarih"] = pd.to_datetime(df["tarih"])
gunluk = df.groupby("tarih")["toplam"].sum()

print(gunluk.idxmax(), "gunu en cok satis yapildi.")
```

- Soru: En çok satış yapılan gün hangisi?
- Cevap: `idxmax()` → En yüksek değerin tarihi
- Yorum: Hafta sonu mu? Kampanya dönemi mi?

Veriden Hikaye Çıkarmak

- “Satışlar hafta ortasında düşüyor, cuma tekrar artıyor.”
- “Elektronik kategorisi yüksek, çünkü ürün başı fiyatlar yüksek.”
- “Hafta sonu satış artışı $\rightarrow$ müşterilerin alışveriş alışkanlığı”
- Yorum yapabilmek = veriden değer üretmek

Analitik düşünme yetkinliği = Veri bilimi ve yapay zeka için temel beceridir.

Veri Ön İşleme Neden Gerekli?

- Makine öğrenmesi modelleri sayısal verilerle çalışır.
- Kategorik veriler sayıya çevrilmeli, sayısal veriler ölçeklenmelidir.
- Bu adımlar model başarısını ve yorumlanabilirliği artırır.

Kategorik Veriyi Sayısallaştırma

```
1 import pandas as pd
2
3 df = pd.DataFrame({
4     "cinsiyet": ["E", "K", "K", "E"]
5 })
6
7 df_kodlu = pd.get_dummies(df, columns=["cinsiyet"],, dtype=
      int)
8 print(df_kodlu)
```

- `get_dummies()` → kategorik sütunları 0/1 vektörlerine dönüştürür.
- Özellikle ”erkek/kadın” gibi sınıflarda kullanılır.

Sayısal Veriyi Neden Normalleştiriyoruz?

- Farklı ölçeklerdeki veriler model öğrenmesini olumsuz etkiler.
  - Örnek: `gelir` (1000– 100000) ve `yaş` (18–65)
- Büyük değerler, küçük değerleri baskılayabilir.
- Özellikle mesafe tabanlı algoritmalarda (kNN, K-Means) bu durum çok önemlidir.
- Normalleştirme sayesinde:
  - Tüm değişkenler benzer ölçeğe gelir.
  - Model daha hızlı ve dengeli öğrenir.
  - Eğitim süresi kısalır, ağırlık öğrenimi istikrarlı olur.

Sayısal Veriyi Normalleştirme

```python
from sklearn.preprocessing import MinMaxScaler

veri = [[50], [80], [100]]
scaler = MinMaxScaler()
normal = scaler.fit_transform(veri)

print(normal)
```

- `MinMaxScaler` verileri 0-1 aralığına çeker.
- Modelin daha hızlı ve kararlı öğrenmesini sağlar.

Standardizasyon (Z-Skoru)

```python
from sklearn.preprocessing import StandardScaler

veri = [[60], [70], [80]]
scaler = StandardScaler()
zskor = scaler.fit_transform(veri)

print(zskor)
```

- `StandardScaler` → ortalaması 0, std sapması 1 olacak şekilde dönüştürür.
- Özellikle mesafe tabanlı algoritmalarda önemlidir.

- İstatistiksel özetler, verinin yapısını anlamamızı sağlar.
- Ortalama, medyan, varyans, standart sapma, korelasyon gibi metrikler incelenmelidir.
- Bu bilgiler, model seçimi ve yorumlamada yol gösterir.

Medyan, Varyans, Standart Sapma

```python
import numpy as np

veri = [50, 55, 60, 100]

print("Medyan:", np.median(veri))
print("Varyans:", np.var(veri))
print("Standart Sapma:", np.std(veri))
```

- `median()` → ortadaki değer
- `var()`, `std()` → veri yayılımını ölçer

Pandas ile İstatistiksel Özet

```python
import pandas as pd

df = pd.DataFrame({
    "yas": [20, 30, 40, 35],
    "puan": [80, 70, 90, 85]
})

print(df.describe())
```

- `describe()` → sayısal sütunların genel özetini verir.
- Ort., min, max, çeyrek değerler gibi birçok ölçütü otomatik verir.

Colab'de Sistem Komutları

- `!` ile Linux terminal komutları çalıştırılabilir
  - `!pwd` → Mevcut klasörü gösterir
  - `!ls /content/` → Dosyaları listeler
  - `!nvidia-smi` → GPU varsa özelliklerini gösterir
- Ortamı tanımak ve kontrol etmek için kullanışlıdır

Google Drive Bağlama ve Klasör Gezinme

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3
4 !ls /content/drive/MyDrive/
5 %cd /content/drive/MyDrive/projeler/
```

- `drive.mount()` ile Drive bağlanır
- `!ls` → Klasör içeriği listelenir
- `%cd` → İlgili klasöre geçiş yapılır

Veri Seti

```python
import pandas as pd

df = pd.read_csv("titanic.csv")
print(df.head())
```

- Titanic yolcu verisi: yaş, cinsiyet, sınıf, hayatta kalma
- `read_csv()` ile dosya yüklenir
- `head()` ile ilk 5 satır görüntülenir

Eksik Veriler

```
1 print(df.isna().sum())
```

- Bazı sütunlarda eksik değer olabilir
- Bu veriler doldurulabilir, silinebilir ya da göz ardı edilebilir
- `isna().sum()` → kaç eksik olduğunu gösterir

Filtreleme: Örnek Seçimler

```
1 kadinlar = df[df["Sex"] == "female"]
2 yetiskinler = df[df["Age"] > 30]
3 birinci_sinif = df[df["Pclass"] == 1]
```

- Cinsiyet, yaş ve sınıfa göre filtreleme örnekleri
- Bu filtrelerle veri segmentasyonu yapılabilir

Hayatta Kalma Oranı

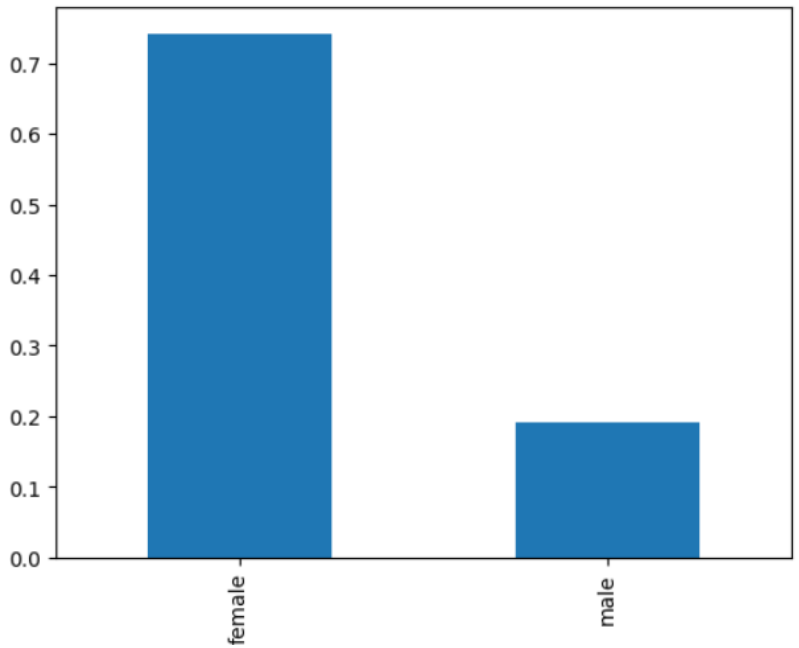
```python
oran = df["Survived"].mean()
print("Genel hayatta kalma orani:", oran)

#float((df["Survived"] == 1).sum()/df["Survived"].size)
```

- `Survived`: 0 = öldü, 1 = hayatta kaldı
- Ortalama hayatta kalma oranını verir
- Grup bazlı oranlar da yapılabilir

Görselleştirme: Cinsiyet ve Hayatta Kalma

```
1 df.groupby("Sex")["Survived"].mean().plot(kind="bar")
```

- Cinsiyete göre hayatta kalma oranı
- Bar grafikle fark kolayca görülür



Ne Yaptık?

- Gerçek bir veri setini inceledik (titanic.csv)
- Eksik verileri kontrol ettik
- Koşullu filtreleme uyguladık
- Ortalama hayatta kalma oranını hesapladık
- Grup bazlı grafiklerle sonuçları görselleştirdik

Uygulama Görevi: Online Satış Verisi Analizi

Veri Seti: `satislar.csv` dosyasını kullanarak aşağıdaki adımları takip ediniz.

1. Veriyi oku: `pandas` ile dosyayı içe aktar.
2. İlk inceleme: `head()`, `info()`, `describe()` ile yapıyı tanı.
3. Eksik değerleri tespit et: Hangi sütunlarda eksik var? Kaç adet?
4. Eksik değerleri doldur: `adet` ve `fiyat` sütunlarını ortalama ile doldur.
5. Yeni sütun oluştur: `toplam = adet * fiyat`
6. Kategori bazlı analiz: `groupby()` ile toplam satışları hesapla, bar grafik çiz.
7. Günlük satış trendi: `tarih` sütununu dönüştür ve çizgi grafikle göster.
8. Şehir bazlı analiz: Hangi şehirde satış daha fazla? Bar grafikle sun.
9. Ödeme tipi analizi: Hangi ödeme türü daha yaygın? `countplot` çiz.

Öğrenci Başarı Analizi

Veri Seti: `ogrenci_notlari.csv` dosyasını analiz ederek aşağıdaki işlemleri uygulayınız:

1. Veriyi oku ve ilk 5 satırı görüntüle.
2. Eksik notları tespit et ve ortalama ile doldur.
3. Başarı notu hesapla: Vize %40, Final %60.
4. Geçti/Kaldı sınıflandırması yap. Notu 50 ve üzeri olanlar geçti.
5. Her ders için ortalama başarıyı hesapla ve bar grafikle göster.
6. Devamsızlık dağılımını histogram grafiği ile çiz.
7. Yorum yap:
   - En düşük başarı hangi derste?
   - Devamsızlık başarıyı etkiliyor mu?
   - Sınıfta kaç kişi geçti/kaldı?