AÇIK ERİŞİMLİ DERS KİTABI

Veri Analizi için Python Kütüphaneleri

Caner Erden

Sakarya Uygulamalı Bilimler Üniversitesi, Bilgisayar Mühendisliği Bölümü

Yıl	2023
Sürüm	Çevrimiçi baskı
Çevrimiçi	https://canererden.com/veri-analizi-python/
Lisans	Bkz. depo LICENSE dosyası

Özet

Bu kitap, Python programlama dili kullanarak veri analizi yapmak isteyenler için hazırlanmış açık erişimli bir kaynaktır. NumPy ile çok boyutlu diziler, Pandas ile veri işleme ve dönüştürme, Matplotlib ile görselleştirme ve Scikit-Learn ile makine öğrenmesinin temelleri uygulamalı örneklerle ele alınır. Her bölüm çalıştırılabilir kod hücreleri ve alıştırmalar içerir. Hedef okuyucu kitlesi lisans ve lisansüstü öğrenciler, araştırmacılar ve veri analizi yapan uygulayıcılardır.

Anahtar kelimeler: veri analizi, Python, NumPy, Pandas, Matplotlib, Scikit-Learn, makine öğrenmesi, veri görselleştirme.

İçindekiler

Veri Analizi için Python Kütüphaneleri

Bu kitap, Python programlama dili kullanarak veri analizi yapmak isteyen herkes için tasarlanmış bir kaynaktır. NumPy, Pandas ve Matplotlib kütüphanelerini kullanarak, temel veri analizi becerilerini kazanmanıza yardımcı olacak kapsamlı bir rehberdir.

Kitap, Python programlama dilini bilmeyenlerin bile anlayabileceği şekilde yazılmıştır. Okuyuculara, veri analizi yapmak için gerekli olan temel Python becerilerini öğretecektir. Daha sonra, NumPy kütüphanesinin kullanımı ile sayısal verileri manipüle etmeyi öğreneceksiniz. Ardından, Pandas kütüphanesini kullanarak verileri işlemenin yanı sıra kesikli hale getirme, normalizasyon, standardizasyon ve one-hot encoding gibi dönüşümleri uygulayabileceksiniz. Son olarak, Matplotlib kütüphanesi ile verileri görselleştirecek ve grafikler oluşturacaksınız.

Kitap, her bölümün sonunda uygulama örnekleri ve egzersizler içermektedir. Bu sayede okuyucular, öğrendiklerini pratik yaparak pekiştirebilirler.

Bu kitap, veri analizi yapmak isteyen öğrenciler, araştırmacılar, veri bilimcileri ve endüstriyel uygulayıcılar için ideal bir kaynak olacaktır. Okuyucular, bu kaynak sayesinde, verileri doğru şekilde manipüle edip analiz edebilecek, sonuçlarını görselleştirebilecek ve bu sonuçları yorumlayabileceklerdir.

NumPy ile Çok Boyutlu Diziler

NumPy, Python programlama dilinde sayısal işlemler için kullanılan açık kaynaklı bir kütüphanedir. Bu kütüphane, büyük veri setleri üzerinde hızlı ve etkili hesaplamalar yapmak için tasarlanmıştır ve bilimsel, mühendislik ve veri analizi uygulamaları için sıkça kullanılır. NumPy, matematiksel ve istatistiksel hesaplamalar yapmak, çok boyutlu diziler ve matrislerle çalışmak ve lineer cebir işlemlerini gerçekleştirmek için bir dizi fonksiyon ve araç sağlar. Bu bölümde, NumPy'nin temel özelliklerini ve kullanımını daha ayrıntılı olarak inceleyeceğiz.

NumPy Kütüphanesine Giriş

NumPy, Python programlama dilinde sayısal işlemleri gerçekleştirmek için kullanılan bir kütüphanedir. Bu kütüphane, büyük veri setleri üzerinde hızlı ve etkili hesaplamalar yapmak için özel olarak tasarlanmıştır. Ayrıca, bilimsel ve matematiksel hesaplamalar yapmak için bir dizi fonksiyon ve araç sağlar. Bu nedenle, birçok veri bilimi, mühendislik ve bilimsel uygulama, NumPy kullanarak hesaplama yapar. NumPy, çok boyutlu dizileri ve matrisleri işlemek için optimize edilmiştir ve bu nedenle de büyük veri setleri üzerinde hızlı işlemler yapmak için idealdir.

NumPy kütüphanesi, birçok matematiksel fonksiyonu içerir. Örneğin, trigonometrik fonksiyonlar (sinüs, kosinüs, tanjant, vb.), logaritmalar, üs alma, karekök alma, vb. gibi temel matematiksel işlemleri gerçekleştirmek için kullanılabilir. Ayrıca, NumPy, polinomlar, fonksiyonlar ve veriler üzerinde çeşitli istatistiksel işlemleri gerçekleştirmek için de bir dizi fonksiyon sağlar.

NumPy kütüphanesi, verileri diziler halinde saklar. Bu diziler, farklı boyutlara sahip çok boyutlu diziler olarak tanımlanabilir. NumPy dizileri, Python listelerinden daha hızlı ve daha verimlidir ve bu nedenle büyük veri setleri üzerinde çalışmak için idealdir. NumPy, ayrıca bu diziler üzerinde çeşitli işlemler yapmak için bir dizi fonksiyon sağlar. Örneğin, dizileri birleştirmek, yeniden şekillendirmek, kesmek, vb.

NumPy kütüphanesi ayrıca, çeşitli lineer cebir işlemlerini de gerçekleştirmek için kullanılabilir. Örneğin, matris çarpımı, matris tersi alma, determinant hesaplama, vb. gibi işlemler yapmak için NumPy fonksiyonları kullanılabilir.

Sonuç olarak, NumPy, Python programlama dilinde sayısal işlemleri gerçekleştirmek için kullanılan bir kütüphanedir. Büyük veri setleri üzerinde hızlı ve etkili hesaplamalar yapmak için özel olarak tasarlanmıştır. NumPy, bilimsel ve matematiksel hesaplamaları yapmak için birçok fonksiyon ve araç sağlar ve çok boyutlu dizileri ve matrisleri işlemek için optimize edilmiştir. NumPy, verileri diziler halinde saklar ve büyük veri setleri üzerinde çalışmak için idealdir.

Çağırma ve import etme

NumPy'yı kullanmak için öncelikle kütüphaneyi çağırmalıyız. Bu, genellikle aşağıdaki şekilde yapılır:

```
import numpy as np
```

Bu, NumPy kütüphanesini Python programına dahil eder ve `np` kısaltması ile çağırmayı sağlar. Ardından, NumPy dizileri oluşturmak ve temel işlemler yapmak için aşağıdaki gibi kodlar yazılabilir:

```
import numpy as np

# 1D NumPy dizi oluşturma
a = np.array([1, 2, 3])
print(a)

# 2D NumPy dizi oluşturma
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b)

# NumPy dizileri üzerinde temel işlemler yapma
c = np.array([1, 2, 3])
d = np.array([4, 5, 6])

print(c + d) # toplama
print(c - d) # çıkarma
print(c * d) # çarpma
print(c / d) # bölme
```

```
[1 2 3]
[[1 2 3]
 [4 5 6]]
[5 7 9]
[-3 -3 -3]
[ 4 10 18]
[0.25 0.4  0.5 ]
```

Bu kodlar, 1D ve 2D NumPy dizileri oluşturmak için `np.array()` fonksiyonunu kullanır ve temel matematiksel işlemleri gerçekleştirmek için NumPy dizileri üzerinde `+`, `-`, `*`, `/` operatörlerini kullanır.

NumPy versiyonu

NumPy'nin sürümü, kütüphanenin özelliklerinde ve işlevlerindeki değişiklikleri yansıtır ve kullanıcıların hangi sürümün yüklü olduğunu ve hangi özellikleri kullanabileceklerini bilmesini sağlar.

NumPy sürümünü kontrol etmek için, aşağıdaki gibi bir kod kullanabilirsiniz:

```
np.__version__
```

```
'1.20.3'
```

NumPy sürümü, kütüphanenin yüklendiği Python sürümüne göre değişebilir. Bu nedenle, farklı Python sürümleri için farklı NumPy sürümleri mevcut olabilir. NumPy'nin web sitesinde, hangi Python sürümlerinin hangi NumPy sürümleriyle uyumlu olduğuna dair bir uyumluluk tablosu mevcuttur.

Çok Boyutlu Diziler

NumPy'nin en önemli özelliklerinden biri, çok boyutlu dizileri desteklemesidir. Çok boyutlu diziler, farklı boyutlardaki verileri saklamak için kullanılan veri yapılarıdır. NumPy, iki veya daha fazla boyutta veri depolamak için kullanılan çok boyutlu dizilere olanak tanır. Çok boyutlu diziler, matrisler olarak da bilinir ve 2D NumPy dizileri matrislerin bir örneğidir.

NumPy'da çok boyutlu dizileri oluşturmak için, `np.array()` fonksiyonuna bir liste veya bir dizi veri girilir. Örneğin, aşağıdaki kod, 2 satır ve 3 sütun içeren bir matris oluşturur:

```
# 2D NumPy dizi oluşturma
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

NumPy, aynı boyutta olmayan çok sayıda boyutlu diziyi destekler. Örneğin, aşağıdaki kod, 2 matris ve her matris 2 satır ve 3 sütundan oluşan 3 boyutlu bir dizi oluşturur:

```
# 3D NumPy dizi oluşturma
b = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(b)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

Çok boyutlu diziler üzerinde farklı matematiksel işlemler yapılabilir ve NumPy, çok boyutlu diziler üzerinde işlemleri hızlı bir şekilde gerçekleştirebilir.

Matematiksel işlemler

NumPy dizileri, farklı matematiksel işlemleri gerçekleştirmek için bir dizi yönteme (metot) sahiptir. Bazı yaygın kullanılan NumPy dizisi yöntemleri aşağıda açıklanmıştır:

- `ndarray.ndim`: Dizi boyutunu döndürür.
- `ndarray.shape`: Dizi boyutlarını tuple olarak döndürür.
- `ndarray.size`: Dizideki elemanların toplam sayısını döndürür.
- `ndarray.dtype`: Dizinin veri tipini döndürür.
- `ndarray.reshape()`: Dizi şeklini değiştirir.
- `ndarray.min()`: Dizideki en küçük elemanı döndürür.
- `ndarray.max()`: Dizideki en büyük elemanı döndürür.
- `ndarray.sum()`: Dizideki tüm elemanların toplamını döndürür.
- `ndarray.mean()`: Dizideki tüm elemanların ortalamasını döndürür.
- `ndarray.std()`: Dizideki tüm elemanların standart sapmasını döndürür.
- `ndarray.transpose()`: Dizinin transpozunu döndürür.

Örneğin, aşağıdaki kod, bir NumPy dizisi oluşturur ve `shape` ve `size` yöntemlerini kullanarak dizinin boyutunu ve eleman sayısını gösterir:

```
# NumPy dizi oluşturma
a = np.array([[1, 2, 3], [4, 5, 6]])
print("Shape of array:", a.shape) # çıktı: (2, 3)
print("Size of array:", a.size) # çıktı: 6
```

```
Shape of array: (2, 3)
Size of array: 6
```

Ayrıca, aşağıdaki kod, `reshape()` ve `transpose()` yöntemlerini kullanarak dizi şeklini değiştirir ve transpozunu alır:

```
# Dizi şeklini değiştirme
b = a.reshape((3, 2))
print("Reshaped array:", b) # çıktı: [[1 2] [3 4] [5 6]]

# Dizinin transpozunu alma
c = a.transpose()
print("Transposed array:", c) # çıktı: [[1 4] [2 5] [3 6]]
```

```
Reshaped array: [[1 2]
 [3 4]
 [5 6]]
Transposed array: [[1 4]
 [2 5]
 [3 6]]
```

3 boyutlu diziler

NumPy ile 3 boyutlu diziler, 2 boyutlu dizilerde olduğu gibi çok sayıda matematiksel işlem için kullanılabilir. 3 boyutlu diziler, veri depolama, işleme ve görselleştirme gibi uygulamalarda kullanılır. 3 boyutlu bir NumPy dizisi oluşturmak için, `numpy.array()` fonksiyonuna 3 boyutlu bir liste vermek yeterlidir.

Aşağıda, bir 3 boyutlu dizi oluşturmanın ve farklı işlemler yapmanın örnekleri verilmiştir:

```
# 3 boyutlu dizi oluşturma
a = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("Shape of array:", a.shape) # çıktı: (2, 2, 2)

# Dizinin elemanlarına erişme
print("Element at [0, 1, 0]:", a[0, 1, 0]) # çıktı: 3

# Dizinin şeklini değiştirme
b = a.reshape((2, 4))
print("Reshaped array:", b) # çıktı: [[1 2 3 4] [5 6 7 8]]

# Dizinin maksimum değerini bulma
max_value = a.max()
print("Max value of array:", max_value) # çıktı: 8

# Dizinin elemanlarının toplamını bulma
sum_value = a.sum()
print("Sum of array:", sum_value) # çıktı: 36

# Dizinin transpozunu alma
c = a.transpose()
print("Transposed array:", c) # çıktı: [[[1 5] [3 7]] [[2 6] [4 8]]]
```

```
Shape of array: (2, 2, 2)
Element at [0, 1, 0]: 3
Reshaped array: [[1 2 3 4]
 [5 6 7 8]]
Max value of array: 8
Sum of array: 36
Transposed array: [[[1 5]
  [3 7]]

 [[2 6]
  [4 8]]]
```

Yukarıdaki örnekte, `a` adlı bir 3 boyutlu dizi oluşturulmuş, sonra dizinin şekli (`shape`) ve boyutu (`size`) gösterilmiştir. Daha sonra, dizinin elemanlarına erişmek için indeksleme yapılmış, şekli değiştirme, maksimum değeri bulma, elemanlarının toplamını bulma ve transpozunu alma gibi işlemler yapılmıştır.

3 boyutlu diziler, matrislerin aksine, üçüncü bir boyutu da içerdiğinden, farklı bir gösterime ihtiyaç duyarlar. Bir 3 boyutlu dizi, üç farklı boyutta bulunan bir dizi noktadan oluşur. Örneğin, bir RGB resmin 3 boyutlu bir dizi olarak düşünülebilir, bu durumda resimdeki her piksel üç boyutlu bir nokta olacaktır. Aşağıdaki örnek, bir 3 boyutlu dizi gösterimini göstermektedir:

```
[[[ 1  2]
  [ 3  4]]

 [[ 5  6]
  [ 7  8]]]
```

Bu örnekte, 2x2 boyutlu ve 2 adet 2x2 matris içeren bir 3 boyutlu dizi gösterilmiştir. İlk matris şöyle gösterilir:

```
[[ 1  2]
 [ 3  4]]
```

İkinci matris de şöyle gösterilir:

```
[[ 5  6]
 [ 7  8]]
```

Bu matrisler, birbirleriyle aynı düzlemde bulunur, ancak farklı birer “katmanda” yer alırlar. Böylece, bu matrislerin üst üste yığılıp 3 boyutlu bir dizi oluşturdukları görülür.

3 boyutlu dizilerin gösteriminde, matrislerin gösterimindeki gibi her bir eleman arasında bir boşluk yerine, elemanları ayırmak için virgül kullanılır. Ayrıca, farklı katmanlardaki matrisler, aralarına bir boşluk bırakılarak gösterilirler.

Elemanlara ulaşma

Numpy dizilerinde, elemanlara erişmek için indeksleme ve dilimleme işlemleri kullanılır. İndeksleme, dizinin belirli bir elemanına doğrudan erişmek için kullanılırken, dilimleme işlemi, dizinin belirli bir alt kümesine erişmek için kullanılır.

Indeksleme

Indeksleme, belirli bir elemana doğrudan erişmek için kullanılır. Dizilerin indeksleri 0’dan başlar. Örneğin, aşağıdaki kodda, 0, 1 ve 2. elemanlara sırasıyla erişilmiştir.

```
arr = np.array([1, 2, 3])

print(arr[0]) # 1
print(arr[1]) # 2
print(arr[2]) # 3
```

```
1
2
3
```

3 boyutlu bir dizi üzerinde çalışırken, elemanlara erişmek için üç boyutlu bir indeks kullanmanız gerekir. Örneğin, aşağıdaki kodda, 1. katmanın 0, 0 ve 1. elemanına erişilmiştir:

```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print(arr[1,0,0]) # 5
print(arr[1,0,1]) # 6
```

```
5
6
```

Dilimleme

Dilimleme, dizinin belirli bir alt kümesine erişmek için kullanılır. Dilimleme işlemi, iki nokta arasındaki dizinin bütün elemanlarını alır. Örneğin, aşağıdaki kodda, 1. ve 2. elemanlar arasındaki elemanlar alınmıştır.

```
arr = np.array([1, 2, 3, 4, 5])

print(arr[1:3]) # [2, 3]
```

```
[2 3]
```

3 boyutlu bir dizi üzerinde çalışırken, dilimleme işlemi için üç boyutlu bir dilimleme işlemi yapmanız gerekir. Örneğin, aşağıdaki kodda, 1. katmanın tüm elemanları alınmıştır:

```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print(arr[1,:,:]) # [[5, 6], [7, 8]]
```

```
[[5 6]
 [7 8]]
```

Ayrıca, dilimleme işleminde iki nokta yerine üç nokta kullanarak da belirli bir adımda elemanları seçebilirsiniz. Örneğin, aşağıdaki kodda, 1. katmanın 0. ve 1. sıralarındaki elemanlar, ikişer ikişer atlanarak alınmıştır:

```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(arr[1::2, 0::2,:]) # [[[5, 6]]]
```

```
[[[5 6]]]
```

Burada, `1::2` ifadesi, 1. katmandan başlayarak her iki elemanı alacağını belirtirken, `0::2` ifadesi, 0. ve 1. sıralardan başlayarak her iki elemanı alacağını belirtir. Son olarak `:` ifadesi, tüm sütunlardaki elemanları alacağını belirtir.

Numpy dizilerinde indeksleme ve dilimleme işlemleri oldukça güçlüdür ve verilerin büyük ölçekli işlemlerini yapmak için önemlidir.

Mantıksal dilimleme

Mantıksal dilimleme (boolean indexing) yöntemi, numpy dizilerinde belirli koşulları sağlayan verileri seçmek için kullanılır. Bu işlem, belirli koşulları sağlayan verileri filtrelemek için çok kullanışlıdır.

Örneğin, aşağıdaki numpy dizisinde 10'dan küçük olan tüm elemanları seçmek isteyelim:

```
a = np.array([1, 4, 10, 2, 8, 15, 3, 9])
a[a < 10] # array([1, 4, 2, 8, 3, 9])
```

```
array([1, 4, 2, 8, 3, 9])
```

Burada, `a < 10` ifadesi, `a` dizisindeki her elemanın 10'dan küçük olup olmadığını kontrol eder ve sonuç olarak bir bool numpy dizisi döndürür. Bu bool dizisindeki `True` değerleri, ana dizideki o konumdaki elemanın seçileceğini belirtir. Sonuç olarak, `a[a < 10]` ifadesi ile 10'dan küçük olan tüm elemanlar seçilir.

Mantıksal dilimleme yöntemi, çok boyutlu dizilerde de kullanılabilir. Örneğin, aşağıdaki numpy dizisinde 3'ten büyük olan tüm elemanları seçmek isteyelim:

```
b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b[b > 3] # array([4, 5, 6, 7, 8, 9])
```

```
array([4, 5, 6, 7, 8, 9])
```

Burada, `b > 3` ifadesi, `b` dizisindeki her elemanın 3'ten büyük olup olmadığını kontrol eder ve sonuç olarak bir bool numpy dizisi döndürür. Bu bool dizisindeki `True` değerleri, ana dizideki o konumdaki elemanın seçileceğini belirtir. Sonuç olarak, `b[b > 3]` ifadesi ile 3'ten büyük olan tüm elemanlar seçilir.

Mantıksal dilimleme yöntemi, numpy dizilerindeki veri filtrelemesi yaparken çok kullanışlı bir yöntemdir.

Dizilerde aritmetik işlemler

Numpy dizilerinde aritmetik işlemler, çok hızlı bir şekilde yapılabilir. Temel aritmetik işlemleri (+, -, *, /, **) numpy dizilerinde doğrudan kullanabilirsiniz. Bu işlemler, her bir öğeye ayrı ayrı uygulanır.

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

c = a + b  # [6 8 10 12]
d = a - b  # [-4 -4 -4 -4]
e = a * b  # [ 5 12 21 32]
f = b / a  # [5.         3.         2.33333333 2.        ]
g = a ** 2 # [ 1  4  9 16]
```

Ayrıca, numpy dizileri arasında işlemler de yapabilirsiniz. Bunun için, iki numpy dizisinin şekillerinin uyumlu olması gerekir.

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

d = a + c  # ValueError: operands could not be broadcast together with shapes (4,) (2,2)
e = c + b  # [[ 6  8]
           #  [10 12]]
f = a * c  # [[ 1  4]
           #  [ 9 16]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15032/4240253268.py in <module>
      3 c = np.array([[1, 2], [3, 4]])
      4
----> 5 d = a + c  # ValueError: operands could not be broadcast together with shapes
(4,) (2,2)
      6 e = c + b  # [[ 6  8]
      7            #  [10 12]]

ValueError: operands could not be broadcast together with shapes (4,) (2,2)
```

Burada, `d = a + c` işleminde, `a` dizisi tek boyutlu iken `c` dizisi 2 boyutlu olduğundan işlem yapılamaz ve hata verir. `e = c + b` işleminde, `b` dizisi 1 boyutlu ve `c` dizisi 2 boyutlu olduğundan, `b` dizisi otomatik olarak `[[5, 6], [5, 6]]` şekline yayılır (broadcast) ve işlem

yapılabilir. `f = a * c` işleminde, `a` dizisi ve `c` dizisi boyutları uyumlu olduğundan doğrudan işlem yapılabilir.

Numpy dizilerinde aritmetik işlemler yaparken, dizilerin şekillerine ve veri tiplerine dikkat etmek önemlidir.

Matris işlemleri

Numpy, matris işlemleri yapmak için kullanışlı bir kütüphanedir. Matrisler, numpy dizileri olarak temsil edilir ve numpy, matris işlemlerini yapmak için uygun fonksiyonları sağlar.

Matrislerin çarpımı, özellikle makine öğrenmesi ve veri analizi gibi birçok alanda kullanılan temel bir matematiksel işlemdir. Numpy ile matris çarpımı yapmak için `dot` fonksiyonu kullanılır. Bu fonksiyon, iki matrisi çarpar ve sonucu yeni bir matris olarak döndürür.

```
# 2x3'lük bir matris
a = np.array([[1, 2, 3], [4, 5, 6]])

# 3x2'lik bir matris
b = np.array([[7, 8], [9, 10], [11, 12]])

# matris çarpımı
c = np.dot(a, b)

print(c)
```

```
[[ 58  64]
 [139 154]]
```

Yukarıdaki örnekte, `a` ve `b` matrisleri `dot` fonksiyonu ile çarpılır ve sonuç olarak yeni bir matris olan `c` oluşur. `c` matrisi, `a` matrisinin satırları ile `b` matrisinin sütunları arasındaki iç çarpımları temsil eder.

Numpy ayrıca, matrisin tersini, determinantını, izini ve diğer matris işlemlerini yapmak için de fonksiyonlar sağlar. Örneğin, bir matrisin tersi `inv` fonksiyonu kullanılarak bulunabilir.

```
a = np.array([[1, 2], [3, 4]])
b = np.linalg.inv(a)

print(b)
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Yukarıdaki örnekte, `a` matrisinin tersi `inv` fonksiyonu ile hesaplanır ve `b` matrisine atılır. Matrisin tersi, matrisin boyutlarına ve özelliklerine bağlı olarak bulunmayabilir veya tanımlı olmayabilir. Bu nedenle, matris işlemleri yaparken matrisin özelliklerine ve boyutlarına dikkat etmek önemlidir.

Sıralı Dizi Oluşturma

Numpy, sıralı dizi oluşturma yöntemleri için birkaç farklı fonksiyon sağlar. Bu fonksiyonlar, belirli bir aralıkta belirli bir adımda artan, azalan veya rastgele değerler üretmek için kullanılabilir.

1. `arange`: Belirli bir aralıkta belirli bir adımda artan değerler üretir. `arange` fonksiyonunun kullanımı şu şekildedir:

```
# 0'dan 10'a kadar (10 hariç) 2 adımda artan değerler
a = np.arange(0, 10, 2)

print(a)
```

```
[0 2 4 6 8]
```

Yukarıdaki örnekte, `arange` fonksiyonu ile 0 ile 10 arasında (10 hariç) 2 adımda artan bir dizi oluşturulur ve `a` değişkenine atılır.

2. `linspace`: Belirli bir aralıkta belirli bir sayıda eşit aralıklı değerler üretir. `linspace` fonksiyonunun kullanımı şu şekildedir:

```
# 0 ile 1 arasında 5 adet eşit aralıklı değer
a = np.linspace(0, 1, 5)

print(a)
```

```
[0.   0.25 0.5  0.75 1.  ]
```

Yukarıdaki örnekte, `linspace` fonksiyonu ile 0 ile 1 arasında 5 adet eşit aralıklı bir dizi oluşturulur ve `a` değişkenine atılır.

3. `logspace`: Belirli bir aralıkta belirli bir sayıda eşit logaritmik aralıklı değerler üretir. `logspace` fonksiyonunun kullanımı şu şekildedir:

```
# 10 üzeri 0'dan 10 üzeri 3'e kadar (10 üzeri 3 dahil) 4 adet logaritmik aralıklı değer
a = np.logspace(0, 3, 4)

print(a)
```

```
[   1.   10.  100. 1000.]
```

Yukarıdaki örnekte, `logspace` fonksiyonu ile 10 üzeri 0'dan 10 üzeri 3'e kadar (10 üzeri 3 dahil) 4 adet eşit logaritmik aralıklı bir dizi oluşturulur ve `a` değişkenine atılır.

4. `zeros` ve `ones`: Sıfırlardan veya birlerden oluşan bir dizi oluşturur. `zeros` ve `ones` fonksiyonlarının kullanımı şu şekildedir:

```
# 2x3'lük bir sıfır matrisi
a = np.zeros((2, 3))

# 3x3'lük bir birler matrisi
b = np.ones((3, 3))

print(a)
print(b)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Dizilerin kaydedilmesi ve kopyalanması

Numpy dizileri, Python listelerinden farklı olarak verileri doğrudan bellekte tutar. Bu nedenle, bir dizi oluşturulduktan sonra veriler üzerinde değişiklik yapmak mümkündür ancak bu verilerin orijinal halini kaybetme riski vardır. Bu nedenle, Numpy dizilerinin kopyalanması ve kaydedilmesi işlemleri dikkatli bir şekilde yapılmalıdır.

Dizilerin Kaydedilmesi:

Numpy dizileri, `.npy` uzantılı dosyalara kaydedilebilir ve daha sonra `load` fonksiyonuyla tekrar yüklenebilir. Dizi kaydetmek için `save` fonksiyonu kullanılır:

```
# 1x4'lük bir dizi oluştur
a = np.array([1, 2, 3, 4])

# Diziyi kaydet
np.save("dizi.npy", a)

# Kaydedilmiş diziyi yükle
b = np.load("dizi.npy")

print(b)
```

```
[1 2 3 4]
```

Yukarıdaki örnekte, `save` fonksiyonu ile `a` dizisi "dizi.npy" adlı dosyaya kaydedilir ve `load` fonksiyonu ile tekrar yüklenerek `b` değişkenine atanır.

Dizilerin Kopyalanması:

Numpy dizilerinin kopyalanması iki şekilde yapılabilir: "shallow copy" (yüzeysel kopya) ve "deep copy" (derin kopya).

Yüzeysel kopya (`view`) fonksiyonu, dizinin verilerini doğrudan kopyalamaz. Bunun yerine, orijinal dizi ile yeni dizi arasında bir referans bağı oluşturur. Bu nedenle, bir dizi üzerinde yapılan değişiklikler, yüzeysel bir kopyasını alınan dizi üzerinde de yapılır:

```
# 2x2'lik bir dizi oluştur
a = np.array([[1, 2], [3, 4]])

# Yüzeysel bir kopya oluştur
b = a.view()

# a dizisindeki verileri değiştir
a[0][0] = 0

print(a)
print(b)
```

```
[[0 2]
 [3 4]]
[[0 2]
 [3 4]]
```

Yukarıdaki örnekte, `view` fonksiyonu ile `a` dizisinin yüzeysel bir kopyası oluşturulur ve `b` değişkenine atanır. `a` dizisindeki veriler değiştirildiğinde, `b` dizisi de etkilenir.

Derin kopya (`copy`) fonksiyonu, bir dizinin tamamen yeni bir kopyasını oluşturur. Bu nedenle, orijinal dizi üzerinde yapılan değişiklikler, derin bir kopyasını alınan dizi üzerinde etkilemez:

```
# 2x2'lik bir dizi oluştur
a = np.array([[1, 2], [3, 4]])

# Derin bir kopya oluştur
b = a.copy()

# a dizisindeki verileri değiştir
a[0][0] = 0

print(a)
print(b)
```

```

```

```
[[0 2]
 [3 4]]
[[1 2]
 [3 4]]
```

Yukarıdaki örnekte, `copy` fonksiyonu ile `a` dizisinin derin bir kopyası oluşturulur ve `b` değişkenine atanır. `a` dizisindeki veriler değiştirildiğinde, `b` dizisi bundan etkilenmez.

Özetle, Numpy dizileri `.npy` dosyalarına kaydedilebilir ve yüzeysel kopya ve derin kopya yöntemleri kullanılarak kopyalanabilir. Ancak, verilerin doğrudan bellekte tutulması nedeniyle, diziler üzerinde yapılan değişiklikler orijinal verileri etkileyebilir. Bu nedenle, Numpy dizileri üzerinde işlem yaparken dikkatli olunmalı ve kaydetme ve kopyalama işlemleri dikkatle yapılmalıdır.

Sayı formatlama

`fmt` (format) parametresi, Numpy dizilerini yazdırırken kullanılan bir biçimlendirme parametresidir. Bu parametre, yazdırılan dizinin elemanlarının nasıl biçimlendirileceğini belirler.

`fmt` parametresi, standart Python biçimlendirme dizelerini kullanır. Örneğin, `"%d"` bir tamsayı olarak biçimlendirirken, `"%f"` bir ondalık sayı olarak biçimlendirir. Aşağıdaki örnek, `fmt` parametresi kullanarak bir Numpy dizisini yazdırmak için `np.savetxt()` fonksiyonunu kullanır:

```
# 3x3 boyutunda bir dizi oluştur
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Diziyi bir metin dosyasına yazdır
np.savetxt('my_array.txt', a, fmt='%d')

# Dosyayı oku ve yazdır
with open('my_array.txt', 'r') as f:
    print(f.read())
```

```
1 2 3
4 5 6
7 8 9
```

Bu örnekte, `fmt='%d'` parametresi, `a` dizisindeki elemanları tam sayı olarak biçimlendirir ve `my_array.txt` adlı bir dosyaya yazdırır. Dosya, aşağıdaki gibi görünecektir:

```
1 2 3
4 5 6
7 8 9
```

`fmt` parametresi, aynı zamanda farklı biçimlendirmeleri bir arada kullanmak için de kullanılabilir. Örneğin, aşağıdaki örnek, bir Numpy dizisindeki sayıları tam sayı ve ondalık olarak bir arada kullanarak yazdırır:

```
# 3x3 boyutunda bir dizi oluştur
a = np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]])

# Diziyi bir metin dosyasına yazdır
np.savetxt('my_array.txt', a, fmt='%.2f')

# Dosyayı oku ve yazdır
with open('my_array.txt', 'r') as f:
    print(f.read())
```

```
1.00 2.00 3.00
4.00 5.00 6.00
7.00 8.00 9.00
```

Bu örnekte, `fmt='%d %.2f'` parametresi, dizideki tam sayıları tam sayı olarak ve ondalık sayıları iki ondalık basamaklı olarak biçimlendirir. Dosya aşağıdaki gibi görünecektir:

```
1.00 2.00 3.00
4.00 5.00 6.00
7.00 8.00 9.00
```

Formatlama ile ilgili daha detaylı örneklere şu adresten ulaşılabilir.

Sonuç

Bu bölümde, Numpy kütüphanesiyle ilgili çeşitli konular ele alındı. Öncelikle, Numpy kütüphanesini nasıl kullanacağımızı, dizilerin nasıl oluşturulacağını ve 3 boyutlu dizilerin nasıl kullanılacağını öğrendik. Daha sonra, dizilerdeki indeksleme, dilimleme ve mantıksal dilimleme yöntemlerini inceledik. Ayrıca, Numpy dizileri üzerinde aritmetik ve matris işlemleri yapmanın yanı sıra, sıralı diziler oluşturma yöntemlerini de öğrendik. Son olarak, dizilerin kaydedilmesi, kopyalanması ve fmt parametresi hakkında bilgi edindik. Numpy kütüphanesi, bilimsel hesaplama ve veri analizi için güçlü bir araçtır ve bu bölümde ele aldığımız konuların bir kısmı, Numpy ile yapabileceğimiz işlemlerin sadece bir örneğidir.

Pandas ile Veri Analizi ve İşleme

Bu bölümde, temel pandas veri yapılarını oluşturmadan verileri pandas ile analiz etmeye kadar, pandas'ın veri işleme ve analizi alanındaki yeteneklerini uygulama örnekleri ve pratik egzersizlerle göstererek size yardımcı olacak bir kısa bir kurs olarak verilmiştir. Bu bölümün sonunda, veri okuma ve yazma, veri çerçeveleri birleştirme, veri ön işleme, veri görselleştirme vb. temel deneyimleri kazanmış olacaksınız. Daha sonraki bölümlerinde daha ayrıntılı bir şekilde ele alacağımız kütüphanenin çoğu temel özelliklerini bu bölümde belirteceğiz.

Python Pandas kütüphanesi, veri analizi ve işleme için kullanılan açık kaynaklı bir kütüphanedir. Bu kütüphane, veri manipülasyonu, filtreleme, görselleştirme ve veri analizi gibi birçok işlemi kolaylaştırır. Python programlama dilinde Pandas kütüphanesi kullanarak veri analizi yapmak oldukça kolaydır.

Bu yazıda, Python Pandas kütüphanesi hakkında genel bir bilgi edineceğiz ve kütüphane ile nasıl veri analizi yapabileceğimizi öğreneceğiz.

pandas Kütüphanesi

Pandas, Python programlama dilinde veri analizi için kullanılan bir kütüphanedir. Veri manipülasyonu, veri filtreleme, veri birleştirme, veri işleme, veri analizi ve veri görselleştirme gibi birçok işlemi yapmamıza yardımcı olur. Pandas, verileri düzenlemek, analiz etmek ve modellemek için yaygın olarak kullanılan bir araçtır.

Pandas kütüphanesi, iki temel veri yapıları olan DataFrame ve Series sınıflarına dayanmaktadır. DataFrame, iki boyutlu bir veri yapısıdır ve sütunlar ve satırlar şeklinde organize edilir. Series, bir boyutlu bir veri yapısıdır ve sadece bir sütun içerir.

Nasıl yüklenir?

Pandas kütüphanesi, Python programlama dilinin bir varsayılan kütüphanesi değildir ve sonradan yüklenmesi gerekmektedir. Pandas kütüphanesini yüklemek için aşağıdaki komutu kullanabilirsiniz:

`pip install pandas`

```
import pandas as pd
import numpy as np
```

```
pd.__version__  # pandas versiyonu: '1.3.4'
```

```
'1.3.4'
```

pandas kullanım alanları

Pandas, Python programlama dilinde yüksek performanslı ve kullanımı kolay veri analizi ve manipülasyonu kütüphanesidir. Pandas, özellikle tablo benzeri yapıdaki veriler üzerinde çalışmak için tasarlanmıştır ve sütun ve satırların adlandırılmasına ve indekslenmesine izin verir. Pandas ayrıca, veri okuma ve yazma işlemleri için de birçok araç sağlar.

Pandas kütüphanesi ile yapılabilecek işlemler şu şekilde özetlenebilir:

- Veri Okuma ve Yazma
- Veri Seçme ve Sıralama
- Veri Dönüştürme ve Temizleme
- Veri Gruplama ve Birleştirm
- Zaman Serisi Analizi
- Veri Görselleştirme
- Veri Modelleme ve Karşılaştırma

Veri Yapıları

Pandas kütüphanesi, üç temel veri yapısı olan DataFrame, İndeks ve Series sınıflarına dayanmaktadır: Seriler (Series), Indeksler (Indexes) ve Veri Çerçeveleri (DataFrames).

Seriler (Series)

Seriler, tek boyutlu bir veri yapısıdır ve birbirinden farklı veri tiplerini içerebilirler. Seriler, indeks ve değerlerden oluşur. Indeksler varsayılan olarak 0'dan başlayan tam sayılar olarak atanabilir veya belirtilen etiketleri kullanarak özelleştirilebilirler.

Seriler, bir boyutlu bir veri yapısıdır ve sadece bir sütun içerir. Seriler, bir liste, dizi veya sözlük gibi bir veri yapısından oluşturulabilir. Seriler aslında veri çerçevesinin sütununu temsil eder.

Aşağıdaki örnekte, bir Seriler nesnesi oluşturulup nesne üzerinde işlemler gösterilmiştir. Örnekler Jupyter Notebook üzerinde gerçekleştirilmiştir.

```
import pandas as pd

veriler = [1, 2, 5, 8]
seri_ornegi = pd.Series(veriler)

print(seri_ornegi)
```

```
0    1
1    2
2    5
3    8
dtype: int64
```

Aşağıdaki kod parçacıklarında Seriler hakkında çeşitli metotlara örnekler verilmiştir.

```
print(seri_ornegi.dtype)  # int64
print(seri_ornegi.values)  # [1 2 5 8]
print(seri_ornegi.name)  # seri_ornegi
print(seri_ornegi.shape)  # (4,)

seri_ornegi2 = pd.Series(
    [2, 3, 4], index=['a', 'b', 'c'],
    name='seri_ornegi')  # indeks değerlerinin değiştirilmesi
print(seri_ornegi2.index)  # Index(['a', 'b', 'c'], dtype='object')

print(seri_ornegi[seri_ornegi > 2])  # 2den büyük veriler

sayilar = np.linspace(0, 10, num=5)
print(sayilar)  # [ 0.   2.5  5.   7.5 10. ]

x = pd.Series(sayilar)  # indeksler [0, 1, 2, 3, 4]
y = pd.Series(sayilar, index=pd.Index([1, 2, 3, 4, 5]))

print(x + y)  # indekslere göre toplama yapılır.

# numpy da iki diziyi toplayınca
print(np.array([1, 1, 1]) + np.array([-1, 0, 1]))
```

```
int64
[1 2 5 8]
None
(4,)
Index(['a', 'b', 'c'], dtype='object')
2    5
3    8
dtype: int64
[ 0.   2.5  5.   7.5 10. ]
0     NaN
1     2.5
2     7.5
3    12.5
4    17.5
5     NaN
dtype: float64
[0 1 2]
```

```
print(2 in seri_ornegi)  # seri örneğinde 2 değeri var mı?
print(10 in seri_ornegi)  # False
```

```
True
False
```

Görüldüğü gibi Seriler tek boyutlu veri yapılarını temsil etmektedir. İki veya daha fazla boyuttaki veri tipleri ile ilgili ise Veri Çerçeveleri kullanılmaktadır.

Veri Çerçeveleri (DataFrames)

Veri Çerçeveleri, iki boyutlu bir veri yapısıdır ve birden çok Seriden oluşur. Veri Çerçeveleri, sütunlara ve satırlara göre indekslenir ve her sütun bir Seri nesnesi temsil eder. Her sütun farklı bir veri türüne sahip olabilir. Veri Çerçeveleri, birçok veri kaynağından okunabilir ve birçok farklı şekilde dönüştürülebilir.

DataFrame, iki boyutlu bir veri yapısıdır ve sütunlar ve satırlar şeklinde organize edilir. DataFrame, tablo benzeri bir yapıya sahiptir ve her sütun bir Series nesnesidir. DataFrame'in her bir satırı, bir gözlem veya bir veri noktasını temsil eder. DataFrame, verileri işlemek ve manipüle etmek için birçok yöntem sağlar.

Aşağıdaki örnekte, "isim", "yas" ve "sehir" sütunları olan bir DataFrame nesnesi oluşturur.

```
# Veri çerçevesi yardımı
pd.DataFrame?
```

```
data = {'isim': ['Ahmet', 'Mehmet', 'Ali', 'Ayşe'],
        'yas': [28, 22, 30, 25],
        'sehir': ['İstanbul', 'Ankara', 'İzmir', 'Bursa']}

df = pd.DataFrame(data)
df
```

	isim	yas	sehir
0	Ahmet	28	İstanbul
1	Mehmet	22	Ankara
2	Ali	30	İzmir
3	Ayşe	25	Bursa

Veri çerçevesi oluşturma yöntemleri

Veri Çerçevelerini oluşturmak için yukarıdaki örnekte olduğu gibi tüm veriler listeler içerisine yazılabilir. Bununla birlikte birkaç yöntemden daha bahsetmemiz gerekirse aşağıdaki yöntemleri söyleyebiliriz.

```
# Veri Çerçevesi oluşturma 1. yöntem
veri_cercevesi = pd.DataFrame(np.random.rand(3, 2),
                              columns=['Sütun 1', 'Sütun 2'],
                              index=['a', 'b', 'c'])
veri_cercevesi
```

	Sütun 1	Sütun 2
a	0.470738	0.198176
b	0.372200	0.376816
c	0.769168	0.094494

Veri çerçevesi oluşturmada 2. yöntem olarak Python veri tiplerinden `sözlükler (dictionary)` kullanılabilir. Sözlükler key:value tiplerinden oluşur.

```
plakalar = {'istanbul':'34','sakarya':'54','kmaraş':'46'}
nufus_sayilari = {'istanbul':20000000,'sakarya':1000000,'kmaraş':900000}
sehirler = pd.DataFrame({'plaka kodları':plakalar,'nufuslar':nufus_sayilari})
print(sehirler)
```

```
         plaka kodları  nufuslar
istanbul            34  20000000
sakarya             54   1000000
kmaraş              46    900000
```

Eğer index ve sütun değerleri eşleşirse toplama gerçekleşir. Bu durumda `sehirler + sehirler` komutu aşağıdaki çıktıyı verir.

```
sehirler + sehirler
```

	plaka kodları	nufuslar
istanbul	3434	40000000
sakarya	5454	2000000
kmaraş	4646	1800000

sehirler iki boyutlu bir veri yapısına sahiptir. Bu veri yapısı Numpy nesnesine dönüştürülebilir. Bunun için kullanılması gereken komut `to_numpy` komutudur.

```
print(sehirler.columns) # Index(['plaka kodları', 'nufuslar'], dtype='object')
print(sehirler.values)
print(sehirler.values)
sehirler_numpy = sehirler.to_numpy()
```

```
Index(['plaka kodları', 'nufuslar'], dtype='object')
[['34' 20000000]
 ['54' 1000000]
 ['46' 900000]]
[['34' 20000000]
 ['54' 1000000]
 ['46' 900000]]
```

Veri Çerçevesi oluşturmanın üçüncü yöntemi de liste fonksiyonları kullanmaktır. Bu duruma örnek aşağıdaki kod parçacıklarında verilmiştir.

```
veri_listesi = [(n, n**2, n**3) for n in range(5)] # list comprehensions
pd.DataFrame(veri_listesi, columns=['n', 'karesi', 'kubu'])
```

	n	karesi	kubu
0	0	0	0
1	1	1	1
2	2	4	8
3	3	9	27
4	4	16	64

Son olarak Veri Çerçeveleri `random` modülü ile oluşturulabilir.

```
np.random.seed(12345)
pd.Series(np.random.rand(5), name='rassal')
```

```
0    0.929616
1    0.316376
2    0.183919
3    0.204560
4    0.567725
Name: rassal, dtype: float64
```

```
np.random.seed(12345)
disari_cikma = pd.DataFrame(
    {
        'rassal': np.random.rand(5),
        'hava durumu': ['sicak', 'ilik', 'soguk', 'yagmurlu', 'karli'],
        'karar': [np.random.choice(['Dışarı Çık', 'Dışarı Çıkma']) for _ in range(5)]
    },
    index=pd.date_range(start='1/1/2018',
                        freq='1D',
                        periods=5,
                        name='tarih'))

disari_cikma
```

tarih	rassal	hava durumu	karar
2018-01-01	0.929616	sicak	Dışarı Çıkma
2018-01-02	0.316376	ilik	Dışarı Çıkma
2018-01-03	0.183919	soguk	Dışarı Çık
2018-01-04	0.204560	yagmurlu	Dışarı Çıkma
2018-01-05	0.567725	karli	Dışarı Çıkma

Veri Çerçevesindeki herhangi bir sütuna ulaşmak için iki yöntem kullanılabilir. Birincisinde köşeli parantez içerisinde sütun ismi verilir. Bu durumda sütun isminde boşluk gibi durumların olması çağırmayı etkilemez. İkinci yöntem olan nokta kullanımında ise boşluk gibi özel karakterlerin kullanılması mümkün değildir. Bu kullanımlara örnekler aşağıda verilmiştir.

```
print(disari_cikma["karar"]) # dışarı çıkma kararı series oldu
print(disari_cikma.karar)# yukarıdaki ile aynı kullanıma sahiptir.
```

```
tarih
2018-01-01    Dışarı Çıkma
2018-01-02    Dışarı Çıkma
2018-01-03      Dışarı Çık
2018-01-04    Dışarı Çıkma
2018-01-05    Dışarı Çıkma
Freq: D, Name: karar, dtype: object
tarih
2018-01-01    Dışarı Çıkma
2018-01-02    Dışarı Çıkma
2018-01-03      Dışarı Çık
2018-01-04    Dışarı Çıkma
2018-01-05    Dışarı Çıkma
Freq: D, Name: karar, dtype: object
```

Indeksler (Index)

Pandas Serilerini numpy tek boyutlu dizilerden ayıran nokta indeks değerleridir. Indeks veri tipi sayesinde satır kayıtlarına etiketler ile ulaşılabilir.

```
indexler = seri_ornegi.index
print(indexler) # Index(['a', 'b', 'c'], dtype='object')
print(type(indexler)) # <class 'pandas.core.indexes.base.Index'>
print(indexler.is_unique) # indeksler tekil verilerden mi oluşuyor mu? True

# indeksler: Değiştirilemez listelerdir.

indeks_1 = pd.Index([1,2,3,4])
indeks_2 = pd.Index([3,4,5])

print(indeks_1.intersection(indeks_2)) # Kesişim: Int64Index([3, 4], dtype='int64')
print(indeks_1.union(indeks_2)) # Birleşim Int64Index([1, 2, 3, 4, 5], dtype='int64')
print(indeks_2.difference(indeks_1)) # Fark Int64Index([5], dtype='int64')
```

```
RangeIndex(start=0, stop=4, step=1)
<class 'pandas.core.indexes.range.RangeIndex'>
True
Int64Index([3, 4], dtype='int64')
Int64Index([1, 2, 3, 4, 5], dtype='int64')
Int64Index([5], dtype='int64')
```

Veri Giriş / Çıkış İşlemleri

Pandas ile işlem yaparken çoğu kez farklı kaynak dosyalarından veri alma ve çıktıları farklı formatlarda yazmak gerekecektir. Bu işlemler, veri ile çalışırken vazgeçilmez işlemlerdir. Aşağıdaki komutlar ile, farklı dosya tipleri ile işlemler gerçekleştirilebilir.

- text dosyası `genfromtxt`
- csv dosyası `read_csv`
- Excel dosyası `read_excel`
- json dosyası `read_json()`
- Veritabanından `read_sql()`

Verilerin Okunması

Burada bir uygulama üzerinde veri giriş çıkış ve kayıt işlemleri gerçekleştirilecektir. Uygulama için bu adreste yer alan veri seti kullanılacaktır.

```
# CSV dosyasının okunması
calisan_verileri = pd.read_csv("https://figshare.com/ndownloader/files/40286887",
                               skiprows=0,
                               sep=',',
                               header=0,
                               index_col=0)
```

```
calisan_verileri
```

Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...
NaN	11/23/2014	6:09 AM	132483	16.655	False	Distribution
Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
Male	5/20/2013	12:39 PM	96914	1.421	False	Product
Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 6 columns

Pandas içerisindeki önemli fonksiyonlardan birisi `read_csv` fonksiyonudur. Bu fonksiyon CSV dosyalarını pandas DataFrame veri yapısına dönüştürmek için kullanılır. Bu yöntem aşağıdaki parametrelerle çağrılabilir:

- `filepath_or_buffer`: Okunacak CSV dosyasının yolu ya da URL'si.
- `sep`: Dosya içerisindeki sütunları ayırmak için kullanılacak ayırıcı karakteri belirtir. Varsayılan ayırıcı virgüldür.
- `delimiter`: Dosya içerisindeki sütunları ayırmak için kullanılacak ayırıcı karakteri belirtir. sep parametresiyle aynı işlevi görür.
- `header`: Başlık satırının (sütun adları) var olup olmadığını belirtir. Varsayılan değer infer ile, ilk satırdaki değerlere göre otomatik olarak belirlenir.
- `index_col`: Veri setinde hangi sütunun index olarak kullanılacağını belirtir.
- `usecols`: Hangi sütunların alınacağını belirtir. Varsayılan olarak tüm sütunlar alınır.
- `dtype`: Sütunların veri tiplerini belirtir.
- `skiprows`: Dosyadaki belirtilen satır sayısı kadar okunmadan atlanır.
- `skipfooter`: Dosyanın sonundan belirtilen satır sayısı kadar okunmadan atlanır.
- `nrows`: Okunacak maksimum satır sayısını belirtir.
- `na_values`: Boş veya eksik verilerin nasıl işleneceğini belirtir. Varsayılan olarak, herhangi bir boş veya eksik veri NAN olarak belirlenir.

- `encoding`: Dosyanın karakter kodlamasını belirtir.

Online kaynaktan veri okunması

Pandas, web sayfalarından veya online kaynaklardan veri okumak için birkaç farklı yöntem sunar. Bunlar arasında `read_html()`, `read_json()`, `read_csv()` ve `read_excel()` gibi popüler yöntemler yer alır. Bu yöntemler, farklı dosya türlerini veya veri kaynaklarını okumak için kullanılabilirler.

Aşağıdaki örnekte, bir web sayfasından veri okunur ve DataFrame nesnesi olarak döndürülür:

```
# Web sayfasındaki tablo verisini okuma
url = 'https://en.wikipedia.org/wiki/List_of_countries_by_population_(United_Nations)'

data = pd.read_html(url)

# İlk tabloyu alalım
df = data[0]

# DataFrame'i göster
df.head()
```

	Country / Area	UN continental region[4]	UN statistical subregion[4]	Population(1 July 2022)	Population(1 April 2023)	Change
0	India	Asia	Southern Asia	1417173173	1428627663	+0.81%
1	China[a]	Asia	Eastern Asia	1425887337	1425671352	−0.02%
2	United States	Americas	Northern America	338289857	339996564	+0.50%
3	Indonesia	Asia	South-eastern Asia	275501339	277534123	+0.74%
4	Pakistan	Asia	Southern Asia	235824863	240485658	+1.98%

Bu örnekte, `pd.read_html()` yöntemi, bir URL'yi parametre olarak alır ve bu URL'deki sayfada yer alan tüm tabloları içeren bir liste döndürür. Bizim örneğimizde, birinci tabloyu seçtik ve bir DataFrame nesnesi olarak döndürdük. Sonuç olarak, df değişkeni, web sayfasındaki ilk tabloyu içeren bir DataFrame nesnesi olarak oluşturuldu.

Ayrıca, `pd.read_json()` gibi diğer yöntemler de benzer şekilde kullanılabilir. Bu yöntemler, farklı veri türleri için özelleştirilmiş seçenekler ve parametreler sunarlar.

Verilerin Kaydedilmesi

pandas kütüphanesi, farklı dosya biçimleri için çeşitli yazdırma metotları sunar. Bazı örnekler aşağıdaki gibidir:

to_csv: Bu metot, verileri bir CSV dosyasına yazdırmak için kullanılır.

```
# DataFrame'i bir CSV dosyasına yazdır
calisan_verileri.to_csv('calisan_verileri.csv', index=False)
```

Bu örnekte, `to_csv()` metodu kullanılarak bu veriler calisan_verileri.csv dosyasına yazdırılır. index=False parametresi, verilerin index sütununu dahil etmemeyi sağlar.

to_csv() yöntemi, verileri CSV dosyasına yazarken belirli bir klasöre kaydetmek için `path_or_buf` parametresini kullanır. Bu parametreye tam bir dosya yolu veya dosya adı verilebilir. Ayrıca, alt dizinleri de dahil ederek farklı klasörlere kaydetmek için dosya yolu belirtirken dizin adı da kullanılabilir.

- normal şekilde kayıt

```
calisan_verileri.to_csv('./data/calisan_verileri.csv', index=False)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4196/25051184.py in <module>
----> 1 calisan_verileri.to_csv('./data/calisan_verileri.csv', index=False)

D:\Anaconda3\lib\site-packages\pandas\core\generic.py in to_csv(self, path_or_buf, sep,
na_rep, float_format, columns, header, index, index_label, mode, encoding, compression,
quoting, quotechar, line_terminator, chunksize, date_format, doublequote, escapechar,
decimal, errors, storage_options)
   3464         )
   3465
-> 3466         return DataFrameRenderer(formatter).to_csv(
   3467             path_or_buf,
   3468             line_terminator=line_terminator,

D:\Anaconda3\lib\site-packages\pandas\io\formats\format.py in to_csv(self, path_or_buf,
encoding, sep, columns, index_label, mode, compression, quoting, quotechar,
line_terminator, chunksize, date_format, doublequote, escapechar, errors,
storage_options)
   1103             formatter=self.fmt,
   1104         )
-> 1105         csv_formatter.save()
   1106
   1107         if created_buffer:

D:\Anaconda3\lib\site-packages\pandas\io\formats\csvs.py in save(self)
    235         """
    236         # apply compression and byte/text conversion
--> 237         with get_handle(
    238             self.filepath_or_buffer,
    239             self.mode,

D:\Anaconda3\lib\site-packages\pandas\io\common.py in get_handle(path_or_buf, mode,
encoding, compression, memory_map, is_text, errors, storage_options)
    700         if ioargs.encoding and "b" not in ioargs.mode:
    701             # Encoding
--> 702             handle = open(
    703                 handle,
    704                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: './data/calisan_verileri.csv'
```

Bu örnekte, ./data yolunu kullanarak calisan_verileri.csv dosyasını data klasörü altında oluşturduk. Bu örneğin çalışabilmesi için çalışma klasörününün içerisine data adında bir klasör oluşturmak gerekmektedir. index=False parametresi, DataFrame'in satır dizinlerinin CSV dosyasına yazılmamasını sağlar. Eğer index=True olarak ayarlanırsa, satır dizinleri CSV dosyasına yazdırılacaktır.

Klasörlerin konumları ile ilgili şunları bilmek gereklidir.

'./' dizini, bulunduğumuz çalışma dizinini temsil eder. Bu nedenle, './data' ifadesi, bulunduğunuz dizin içindeki 'data' adlı bir alt dizine işaret eder. '.' karakteri, mevcut dizini ifade etmek için kullanılır.

'..' ise bir üst dizini ifade eder. Örneğin, '../data' ifadesi, mevcut dizinin bir üstündeki dizindeki 'data' adlı bir alt dizine işaret eder.

‘’ karakteri, Windows işletim sistemlerinde dosya yollarında kullanılan bir kaçış karakteridir. Bu karakter, dosya yolu bölümlerini ayırmak için kullanılır. Örneğin, “C:\Users\kullanici_adi\Desktop” ifadesinde, ‘’ karakterleri “C:”, “Users”, “kullanici_adi” ve “Desktop” gibi dosya yolu bölümlerini ayırmak için kullanılır. Unix tabanlı işletim sistemlerinde ise ‘/’ karakteri kullanılır.

Python’da da ‘’ karakteri kaçış karakteri olarak kullanılır. Ancak, Windows işletim sistemlerinde olduğu gibi dosya yollarında kullanılan ‘’ karakterleri yerine, Python’da Unix tabanlı işletim sistemlerinde olduğu gibi ‘/’ karakterleri kullanılır. Bunun nedeni, Windows işletim sisteminde ‘’ karakterinin kaçış karakteri olarak kullanılmasıdır ve dosya yollarında kullanılan ‘’ karakteri tek başına bir kaçış karakteri olarak algılandığından, dosya yolu yazımında hatalara yol açabilmektedir.

Bununla birlikte Python’da iki tane arka arkaya ‘’ karakteri, bir tanesi normal bir karakter olarak kabul edilip diğerinin kaçış karakteri olarak kabul edilmesini sağlamak için kullanılır. Örneğin, “\server\path\filename” ifadesinde, iki tane ‘’ karakteri, dosya yolunda kullanılan normal ‘’ karakterleri ve kaçış karakterleri arasında bir ayırıcı olarak işlev görür. Bu ifadede, ilk iki ‘’ karakteri, ‘\server’ dizini için normal ‘’ karakterleridir. Sonraki iki ‘’ karakteri, ‘\path’ dizini için normal ‘’ karakterleridir. Ve son olarak, son ‘’ karakteri, dosya adı olan ‘filename’ için normal ‘’ karakteridir.

- `os` modülü kullanarak kayıt

`os` modülü kullanarak da dosyaları farklı klasörlere kaydedebilirsiniz. Eğer data klasörü yoksa oluştur komutunu da burada verebilirsiniz. Örneğin:

```
import os

folder_path = './data'

if not os.path.exists(folder_path):
    os.makedirs(folder_path)

file_path = os.path.join(folder_path, 'calisan_verileri.csv')
calisan_verileri.to_csv(file_path, index=False)
```

Bu örnekte, os modülü kullanılarak öncelikle ./data klasörü oluşturulur. Daha sonra, os.path.join() yöntemi kullanarak calisan_verileri.csv dosyasının tam yolu oluşturulur ve DataFrame to_csv() yöntemi kullanılarak bu dosyaya yazdırılır.

`to_excel`: Bu metot, verileri bir Excel dosyasına yazdırmak için kullanılır. Örnek kullanımı:

```
# DataFrame'i bir Excel dosyasına yazdır
calisan_verileri.to_excel('calisan_verileri.xlsx', index=False)
```

Elde edilen Excel dosyası açıldığında aşağıdaki gibi düzgün bir çıktı alınacaktır.

Şekil 6 - Excel Ekran Görüntüsü

`to_sql`: Bu metot, verileri bir SQL veritabanına yazdırmak için kullanılır.

pandas Veri Tipleri

Pandas veri yapıları, aşağıdaki veri tiplerini içerebilir:

- float: Ondalık sayıları temsil eder. Örneğin, 3.14 gibi.
- int: Tam sayıları temsil eder. Örneğin, 42 gibi.
- bool: True ve False mantıksal değerlerini temsil eder.
- datetime: Tarih ve zaman değerlerini temsil eder. Örneğin, "2022-01-01 12:00:00" gibi.
- timedelta: İki tarih veya zaman arasındaki farkı temsil eder.
- object: Stringler, listeler, sözlükler, vb. nesneleri temsil eder.

Pandas, bu temel veri yapılarına ek olarak, Categorical ve Sparse veri yapıları da sunar. Categorical veri yapısı, kategorik verileri temsil etmek için kullanılır ve Sparse veri yapısı, verilerin seyrek veya büyük olduğu durumlarda bellek kullanımını optimize etmek için kullanılır.

`dtypes`, bir veri çerçevesinin tüm sütunlarının veri tiplerini gösteren bir özelliktir. Pandas DataFrame'i, farklı veri tiplerini (int, float, bool, string, datetime vb.) içeren sütunlardan oluşabilir. Her bir sütunun veri tipi, o sütundaki verilerin nasıl işleneceğini belirler.

`dtypes`, DataFrame'deki sütunların veri tiplerini gösteren bir Seri nesnesi döndürür. Bu özellik, DataFrame'deki her sütunun veri tipini belirlemek ve herhangi bir sütunun veri tipini değiştirmek gibi işlemler yapmak için kullanılabilir. Ayrıca, farklı veri tiplerine sahip sütunların DataFrame'deki verileri nasıl işlediğini anlamak için de kullanılabilir. Örneğin, bir sütunun tamsayı değerleri taşıması ve daha sonra ondalık sayılara dönüştürülmesi isteniyorsa, bu özellik, ilgili sütunun veri tipini değiştirmek için kullanılabilir.

```
calisan_verileri.dtypes
```

`info()` metodu ise, DataFrame'in bellekteki boyutunu, sütunların isimlerini, dolu hücrelerin sayısını ve her sütundaki veri tiplerinin sayısını görüntüler. Bu metot ayrıca bellekteki DataFrame'in bellek kullanımı ve veri tipleri hakkında da bilgi sağlar.

Ayrıca, her bir sütundaki eksik değerlerin sayısını ve veri tiplerinin bellek boyutunu da verir. Bu bilgi, veri seti üzerinde eksik değerlerin olduğu veya yanlış veri tipleri kullanıldığı durumlarda hataların nedenini anlamak için kullanılabilir.

```
calisan_verileri.info()  # non-null sayılarını görmek için
```

Bununla birlikte bazı durumlarda veri tipleri arasında değişiklik yapılmak istenebilir. Bunun için `astype()` metodu kullanılabilir. Bu metot, Pandas DataFrame sütunlarının veri tiplerini değiştirmek için kullanılır. Bu yöntem, mevcut bir DataFrame'deki bir veya daha fazla sütunun veri tipini değiştirerek, veri analizi veya veri manipülasyonu için daha uygun bir formata dönüştürmeye olanak tanır.

Örneğin, bir sütunun veri tipini sayısal bir formattan tarih veya saat formatına dönüştürmek isteyebilirsiniz. Bu durumda, `astype()` yöntemini kullanarak sütunun veri tipini değiştirebilirsiniz.

Ayrıca, `astype()`, sayısal sütunları float veya integer veri tiplerine dönüştürerek hesaplama işlemlerinin hızını arttırmaya da yardımcı olabilir.

```
calisan_verileri['Salary'] = calisan_verileri['Salary'].astype(float)
```

Yukarıdaki örnekte, `astype()` yöntemi kullanılarak 'Salary' sütununun veri tipi integer'dan float'a değiştirildi.

```
calisan_verileri.dtypes
```

Veriler arasındaki dönüşümler

Pandas, bir veri tipinden diğerine dönüştürmek için birkaç farklı seçenek sunar. Bazı yaygın dönüşüm işlemleri şunları içerir:

- `astype()`: Bu yöntem, bir DataFrame sütununu veya Serisi'ni başka bir veri tipine dönüştürmek için kullanılır. Örneğin, bir sayısal sütunu metinsel bir sütuna dönüştürebilirsiniz. Bu yöntem, belirli bir veri tipi için bir seçenek belirleyebilir veya geniş bir yelpaze sunan `category`, `datetime64`, `timedelta` vb. gibi özel veri tipleri için kullanılabilir.

`to_numeric()`: Bu yöntem, bir seriyi sayısal veri tipine dönüştürmek için kullanılır. Örneğin, bir serinin `object` veri tipinden `float` veya `int` veri tipine dönüştürebilirsiniz.

`to_datetime()`: Bu yöntem, bir Serisi'ni tarih / saat veri tipine dönüştürmek için kullanılır. Örneğin, bir seriyi `object` veya `string` veri tipinden `datetime64` veri tipine dönüştürebilirsiniz.

`to_timedelta()`: Bu yöntem, bir seriyi zaman aralığı (timedelta) veri tipine dönüştürmek için kullanılır. Örneğin, bir seriyi "object" veya "string" veri tipinden "timedelta" veri tipine dönüştürebilirsiniz.

Aşağıdaki örnekler, bir veri tipinin diğerine nasıl dönüştürüleceğini gösterir:

```
# astype() yöntemi
df = pd.DataFrame({'A': [1, 2, 3], 'B': ['x', 'y', 'z']})
df['A'] = df['A'].astype(str)   # 'A' sütununu str veri tipine dönüştürür
df['B'] = df['B'].astype('category')  # 'B' sütununu category veri tipine dönüştürür

# to_numeric() yöntemi
s = pd.Series(['1', '2', '3'])
s = pd.to_numeric(s)   # Serisi'ni sayısal veri tipine dönüştürür

# to_datetime() yöntemi
s = pd.Series(['2021-01-01', '2021-02-01', '2021-03-01'])
s = pd.to_datetime(s)  # Serisi'ni datetime64 veri tipine dönüştürür

# to_timedelta() yöntemi
s = pd.Series(['1 days', '2 days', '3 days'])
s = pd.to_timedelta(s)   # Serisi'ni timedelta veri tipine dönüştürür
```

Bu örneklerde, `astype()`, `to_numeric()`, `to_datetime()` ve `to_timedelta()` yöntemleri kullanılarak farklı veri tiplerinin dönüşümü, verilerin analizi sırasında sık sık gereklidir. Pandas, çeşitli veri tipleri arasında dönüşüm yapabilen esnek bir kütüphanedir.

Bununla birlikte, dönüştürme işlemlerinin her zaman otomatik olarak başarılı olması garanti edilemez. Verilerin doğru şekilde dönüştürülüp dönüştürülmediğini kontrol etmek için, dönüştürme işlemi sonrası verilerin tekrar kontrol edilmesi önemlidir. Ayrıca, verileri dönüştürmek, verilerin boyutlarını da değiştirebilir. Bu nedenle, dönüştürme işlemi sırasında oluşabilecek olası veri kaybı veya hataların farkında olmak da önemlidir.

Veri Seti Üzerinde İşlemler

Veri seti çok sayıda veriden oluşmaktadır. Bu nedenle veri setinin doğru şekilde alındığını göstermek açısından ilk satırlarının yazdırılması faydalı olacaktır. Bunun için `head()` metodu kullanılabilir.

```
calisan_verileri.head()
```

```
calisan_verileri.head(2)  # ilk 2 veri
print(calisan_verileri.empty)  # False
print(calisan_verileri.shape)  # (1000, 7),
print(type(calisan_verileri))  # <class 'pandas.core.frame.DataFrame'>
print(calisan_verileri.columns)
```

```
print(
    "bu veri setinde {} adet veri vardır. Ayrıca özellik sayısı {}'dir.".
    format(calisan_verileri.shape[0], calisan_verileri.shape[1]))
```

Pandas kütüphanesi DataFrame ve Series veri tiplerinde kullanılan `describe()` fonksiyonu, temel istatistiksel bilgileri özetleyen bir tablo oluşturur. Bu istatistiksel bilgiler arasında ortalama, standart sapma, minimum değer, maksimum değer, medyan, çeyreklikler ve veri sayısı yer alır.

`describe()` fonksiyonu, özellikle veri setinin özetini hızlı bir şekilde elde etmek istediğinizde kullanışlıdır. Fonksiyon, sayısal verilerin yanı sıra kategorik veriler hakkında da bazı temel istatistiksel bilgiler sağlar.

```
calisan_verileri.describe() # veri özetleme
```

Bu örnekte, `describe()` fonksiyonu calisan_verileri DataFrame'ine uygulanmış ve istatistiksel bilgileri özetleyen bir tablo oluşturulmuştur.

Bu tablo, maaş ve Bonus sütunları için temel istatistiksel bilgileri özetlemektedir. Çıktı aşağıdaki gibi görünecektir:

`describe()` fonksiyonu, varsayılan olarak sadece sayısal sütunlar için çalışır. Bununla birlikte, include parametresi kullanılarak belirli sütun türleri de tanımlanabilir. Örneğin, tüm sütunlar için istatistiksel özet bilgileri almak için `include='all'` parametresi kullanılabilir.

`describe()` fonksiyonunun ayrıca percentiles parametresi de vardır. Bu parametre, özet istatistiklerindeki çeyreklikleri belirlemek için kullanılan yüzde değerleri tanımlamak için kullanılabilir. Varsayılan olarak, `percentiles` parametresi [0.05, 0.95] olarak ayarlanmıştır.

Ayrıca, bu fonksiyonu varsayılan olarak tüm sütunlardaki eksik değerleri atlar. Bu davranış, `include='all'` parametresiyle aşılabileceği gibi, ayrıca dropna=False parametresi kullanılarak da değiştirilebilir. Bu, eksik değerlerin de dahil edilmesini sağlar ve eksik değerlerin sayısını özet istatistiklerinde gösterir.

```
calisan_verileri.describe(percentiles=[0.05, 0.95]) # diğer dilimleri de görüntülemek
için
```

Veri çerçevesinde veri manipülasyonu, analizi ve temizliği için kullanılan pandas kütüphanesi, veri çerçevesi objeleriyle ilgili birçok metot içermektedir. Bu metotlar, veri işleme sürecinde oldukça yararlıdır. Aşağıda, bazı önemli veri çerçevesi metotları ve kısa açıklamaları yer almaktadır:

- `head()`: Veri çerçevesinin ilk birkaç satırını görüntüler.
- `tail()`: Veri çerçevesinin son birkaç satırını görüntüler.
- `drop()`: Veri çerçevesinden belirtilen sütun veya satırları çıkarır.

- `groupby()`: Belirli bir sütuna göre verileri gruplar.
- `merge()`: İki veya daha fazla veri çerçevesini belirtilen sütuna göre birleştirir.
- `sort_values()`: Belirli bir sütuna göre verileri sıralar.
- `pivot_table()`: Veri çerçevesindeki verileri bir tablo halinde yeniden düzenler.
- `isna() / isnull()`: Veri çerçevesindeki boş değerleri (NaN) tespit eder.
- `fillna()`: Veri çerçevesindeki boş değerleri belirli bir değerle doldurur.
- `drop_duplicates()`: Veri çerçevesindeki tekrarlayan satırları çıkarır.
- `apply()`: Veri çerçevesinde belirtilen işlevi sütunlar veya satırlar üzerinde uygular.
- `map()`: Belirtilen bir işlevi veri çerçevesinin belirli bir sütununa veya serisine uygular.
- `replace()`: Belirli bir değeri başka bir değerle değiştirir.
- `count()`: Her sütundaki non-null (boş olmayan) veri sayısını verir.
- `nunique()`: Her sütundaki benzersiz (tekrar etmeyen) veri sayısını verir.
- `sum()`: Her sütundaki sayısal verilerin toplamını verir. Eğer sütunda sayısal veri yoksa o sütun atlanır.
- `corr()`: Sütunlar arasındaki korelasyon matrisini oluşturur. Korelasyon, iki değişken arasındaki ilişkinin gücünü ve yönünü ölçer. Değerler -1 ile 1 arasındadır. -1 mükemmel negatif korelasyonu, 1 mükemmel pozitif korelasyonu ve 0 korelasyon olmadığını gösterir.

```
calisan_verileri.count()
```

`count()`, `sum()`, `mean()`, `min()`, `max()`, `std()` ve `corr()` fonksiyonları hem Series hem de DataFrame veri yapıları için kullanılabilirken `nunique()`, `value_counts()` gibi fonksiyonlar ise yalnızca Series veri yapısı için kullanılabilirdir.

```
calisan_verileri.Team.value_counts() # hangi takımda kaç kişi var?
```

```
# Takımlara ait istatistikler
calisan_verileri.groupby('Team').mean()
```

```
calisan_verileri.groupby('Team').Team.count()
```

```
calisan_verileri.Team.unique() # Tekil verileri getirir.
```

`None` ve `NaN` veri tipleri

Python'da eksik verileri temsil etmek için genellikle iki ana veri tipi kullanılır: `None` ve `NaN`.

`None`, bir değişkenin değerinin bilinmediği veya tanımlanmadığı durumlarda kullanılır. Örneğin, bir fonksiyonun dönüş değerinin tanımsız olduğu durumlarda None kullanılır.

`NaN (Not a Number)` ise matematiksel işlemler sırasında ortaya çıkan veya eksik sayısal verileri temsil etmek için kullanılır. `NaN`, float veri tipi içinde kullanılır ve matematiksel işlemler sırasında oluşan sayısal hataları temsil eder.

Pandas kütüphanesi de eksik verileri temsil etmek için `NaN` değerini kullanır. `NaN` değeri, bir DataFrame veya Series içindeki belirli bir hücrenin değerinin eksik olduğunu belirtir. Pandas, `NaN` değerlerini otomatik olarak tespit eder ve bu değerleri işlem yaparken dikkate alır.

Eksik verilerin doğru şekilde yönetilmesi, veri analizi ve modelleme sırasında önemlidir. Pandas, eksik verileri ele almak için çeşitli yöntemler sunar, bunlar arasında eksik verileri silmek, eksik verileri belirli bir değer ile doldurmak veya eksik verileri interpolate etmek yer alır.

```
print(type(None)) # NoneType
print(type(np.nan)) # <class 'float'>
```

Yukarıdaki kodda görüldüğü gibi `None` ve `NaN` arasındaki fark, temsil ettikleri veri tiplerinden kaynaklanır.

`None` bir Python nesnesidir ve bir değişkenin değerinin bilinmediği veya tanımlanmadığı durumlarda kullanılır. `None`, herhangi bir veri tipine ait olmayan özel bir değerdir ve yalnızca Python programlama dilinde kullanılır.

`NaN (Not a Number)` ise float veri tipinde kullanılan bir değerdir. Matematiksel işlemler sırasında oluşan veya eksik sayısal verileri temsil etmek için kullanılır. `NaN`, sayısal bir değerin tanımsız veya hesaplanamaz olduğunu belirtir.

Yani, `None` bir nesne veya değişkenin tanımlanmadığı veya bilinmediği durumlarda kullanılırken, `NaN` sayısal değerlerde oluşan eksik veya tanımsız durumlar için kullanılır.

```
print(np.nan + 1) # nan
print(None + 1) # unsupported operand type(s) for +: 'NoneType' and 'int'
```

Öncelikle, bir eksik veri serisi oluşturmak için `None` ve `NaN` değerlerini kullanabiliriz.

```
eksik_veri_serisi = np.array([None, np.nan, 1,2,3,"Merhaba"])
eksik_veri_df = pd.DataFrame(eksik_veri_serisi)
print(eksik_veri_df)
```

`isnull()`, `isna()`, `notnull()`, `notna()` fonksiyonları

Bu seri, 2 tane eksik veri içermektedir. Bu eksik verileri tespit etmek için `isnull()` fonksiyonu kullanılabilir.

```
eksik_veri_df.isnull()
```

```
print(eksik_veri_df.isnull().sum().sum()) # toplam eksik veri sayısı 2
print(eksik_veri_df.notna().sum().sum()) # eksik olmayan veri sayısı 4
```

`fillna()`

Eksik verileri doldurmak için, `fillna()` fonksiyonunu kullanabiliriz. Bu fonksiyon, belirli bir değerle eksik verileri doldurur. Örneğin, aşağıdaki örnekte eksik verileri 0 ve -999 ile dolduruyoruz:

```
print(eksik_veri_df.fillna(value=0))
print(eksik_veri_df.fillna(value=-999))
```

Başka bir seçenek olarak, `ffill` ve `bfill` yöntemlerini kullanarak eksik verileri önceki veya sonraki değerlerle doldurabiliriz.

```
seri_ffill = eksik_veri_df.fillna(method='ffill')  # önceki değerle doldur
print(seri_ffill)

seri_bfill = eksik_veri_df.fillna(method='bfill')  # sonraki değerle doldur
print(seri_bfill)
```

`ffill` yöntemi, eksik verileri önceki değerle doldururken, `bfill` yöntemi sonraki değerle doldurur.

Ayrıca, `interpolate` yöntemini kullanarak, eksik verileri verilerin aritmetik ortalaması veya doğrusal bir model kullanarak interpolate edebiliriz. Bu yöntemi kullanmadan önce serinin nümerik değerlerden oluştuğundan emin olmak gereklidir.

```
seri_interpolate = pd.Series([1, 2, np.nan, 4, np.nan, 6]).interpolate()
print(seri_interpolate)
```

Yukarıdaki örnekte, eksik verileri doğrusal bir model kullanarak interpolate ettik. Eksik veriler, verilerin aritmetik ortalaması veya başka bir yöntemle de doldurulabilir.

dropna() fonksiyonu, eksik verileri içeren satırları veya sütunları veri kümesinden kaldırmak için kullanılır. Bu fonksiyon, NaN değerlerini içeren satırları veya sütunları veri kümesinden çıkararak yeni bir veri kümesi oluşturur.

İlk olarak, bir örnek veri kümesi oluşturalım:

`dropna()`

`dropna()` fonksiyonu, eksik verileri içeren satırları veya sütunları veri kümesinden kaldırmak için kullanılır. Bu fonksiyon, NaN değerlerini içeren satırları veya sütunları veri kümesinden çıkararak yeni bir veri kümesi oluşturur.

İlk olarak, bir örnek veri kümesi oluşturalım:

```
df = pd.DataFrame({'A': [1, 2, np.nan, 4],
                   'B': [5, np.nan, np.nan, 8],
                   'C': [9, 10, 11, 12]})
print(df)
```

Bu veri kümesinde, NaN değerleri içeren satırlar ve sütunlar var. Bu veri kümesinden `dropna()` fonksiyonunu kullanarak, NaN değerlerini içeren satırları kaldırabiliriz:

```
df_without_nulls = df.dropna()
print(df_without_nulls)
```

`dropna()` fonksiyonu, varsayılan olarak herhangi bir NaN değeri içeren satırı kaldırır ve yeni bir veri kümesi oluşturur. `dropna()` fonksiyonuna axis parametresi ekleyerek, NaN değerleri içeren sütunları kaldırabiliriz:

```
df_without_nulls_col = df.dropna(axis=1)
print(df_without_nulls_col)
```

Bu örnekte, `axis=1` parametresi ile sütunları kaldırdık ve sadece C sütunu kaldı.

`dropna()` fonksiyonu, how parametresi ile daha spesifik davranışlar gösterir. Varsayılan olarak `how='any'` olarak ayarlanmıştır, yani herhangi bir eksik değer içeren satırlar veya sütunlar silinir. Ancak, `how='all'` olarak ayarlandığında, tüm değerleri eksik olan satırlar

veya sütunlar silinir. Örneğin:

```
df_all_null = pd.DataFrame({'A': [np.nan, np.nan, np.nan],
                            'B': [np.nan, np.nan, np.nan],
                            'C': [np.nan, np.nan, np.nan]})
print(df_all_null)
```

Bu örnekte, tüm satırlar veya sütunlar NaN olduğu için, `dropna()` fonksiyonu tüm verileri kaldırdı ve boş bir DataFrame döndürdü.

Ayrıca, `dropna()` fonksiyonu `thresh` parametresi ile belirtilen eşik değerinden daha az sayıda geçerli (yani, eksik olmayan) değere sahip olan satırlar veya sütunlar silinir. Örneğin:

```
df_thresh = pd.DataFrame({'A': [1, 2, np.nan, 4],
                          'B': [5, np.nan, np.nan, 8],
                          'C': [9, 10, np.nan, np.nan]})
print(df_thresh)
```

Bu veri kümesinde, en az 2 geçerli değere sahip satırlar veya sütunlar korunacak:

```
df_thresh_dropped = df_thresh.dropna(thresh=2)
print(df_thresh_dropped)
```

Bu örnekte, sadece 2 veya daha fazla geçerli değere sahip olan satırlar ve sütunlar korundu. thresh parametresi, verilerin çeşitli yönlerde temizlenmesine yardımcı olabilir, özellikle büyük veri kümelerinde, verilerin sadece bir kısmı eksikse.

Aykırı veri analizi

Pandas, aykırı verilerin tespiti ve analizi için kullanılabilen bir dizi fonksiyona sahiptir. Aykırı veriler, genellikle diğer verilerden önemli ölçüde farklı olan ve istatistiksel analizlerde yanıltıcı sonuçlara neden olabilen verilerdir.

Pandas'da aykırı verileri tespit etmek için kullanabileceğiniz birkaç yöntem vardır. Bunlar arasında:

1. `describe()` yöntemi: Bu yöntem, veri setinin istatistiklerini hesaplar ve aykırı değerlerin olup olmadığını gösterir. `describe()` yöntemi, veri setindeki en küçük ve en büyük değerleri, ortalamayı, standart sapmayı ve çeyrekliklerin değerlerini gösterir.
2. `boxplot()` yöntemi: Bu yöntem, kutu grafiği olarak da bilinir ve veri setindeki aykırı değerleri görselleştirir. Kutu grafiği, veri setinin dağılımını gösterir ve aykırı değerlerin kutunun dışında kalan noktalar olarak belirtilir.
3. `quantile()` yöntemi: Bu yöntem, belirli bir yüzdelik aralığında yer alan değerleri hesaplar. Bu yöntem kullanılarak, belirli bir yüzdelik aralığına girmeyen değerler aykırı olarak kabul edilebilir.

4. `isolation forest` yöntemi: Bu yöntem, veri setindeki aykırı değerleri tespit etmek için bir makine öğrenmesi algoritmasıdır. Bu yöntem, veri setindeki aykırı değerleri gruplamak için ağaç yapısı kullanır.

Bunlar, pandas kütüphanesi ile aykırı veri analizi için kullanılabilecek yöntemlerden sadece birkaçıdır. Bu yöntemler, veri setindeki aykırı değerleri tespit etmek ve işlemek için kullanılabilir.

IQR yöntemi

IQR (Interquartile Range) yöntemi, aykırı değerleri tanımlamak için kullanılan bir istatistiksel yöntemdir. Bu yöntem, veri dağılımındaki çeyrekliklerin aralığını kullanarak bir alt ve üst sınır belirler ve bu sınırların dışındaki değerleri aykırı olarak tanımlar.

IQR, bir veri kümesindeki değerlerin ortalamasının yerine veri dağılımının merkezi eğilim ölçüsü olarak kullanılır. Veri kümesindeki değerlerin çoğu IQR içinde kaldığı için, IQR yöntemi aykırı değerleri belirlemek için daha güvenilirdir. IQR, verilerin ortalamasına karşı daha dayanıklıdır çünkü sadece veri kümesinin ortasındaki değerlerin aralığını kullanır.

IQR yöntemi kullanarak aykırı değerleri belirlemek için, verilerin 1. ve 3. çeyrekliklerinin aralığı (yani IQR) hesaplanır. Ardından, alt sınır Q1 - 1.5 x IQR ve üst sınır Q3 + 1.5 x IQR şeklinde hesaplanır. Bu sınırların dışındaki herhangi bir değer aykırı olarak kabul edilir.

Örneğin, bir veri kümesinde Q1 = 20 ve Q3 = 50 olsun. Bu durumda, IQR = Q3 - Q1 = 50 - 20 = 30 olacaktır. Alt sınır, 20 - 1.5 x 30 = -5, üst sınır ise 50 + 1.5 x 30 = 95 olacaktır. Veri kümesindeki herhangi bir değer -5'ten küçük veya 95'ten büyükse aykırı olarak kabul edilecektir.

Aşağıdaki örnekte `titanic` veri seti üzerinde bir aykırı veri analizi gösterilmiştir.

```
veri_seti =
pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.cs
v")
veri_seti.head()
```

```
veri_seti.isnull().sum()
```

```
# veri setinin doldurulması
doldurulmus_veri = veri_seti.fillna(method="bfill").fillna(method="ffill")
```

Aşağıdaki komut, "Age" sütununun istatistiksel özetini oluşturur. İstatistiksel özet, sütunun minimum ve maksimum değerleri, ortalama, standart sapma, medyan ve çeyreklikler gibi önemli özelliklerini içerir. "describe()" fonksiyonu, veri çerçevesindeki sayısal sütunlar için

bu tür istatistiksel özetler oluşturmak için sıklıkla kullanılan bir yöntemdir.

```
doldurulmus_veri.Age.describe()
```

Aşağıdaki komut, “Age” adlı bir değişkenin kutu grafiğini çizmek için Pandas kütüphanesinin “plot” fonksiyonunu kullanır.

Bu komut, bir veri çerçevesindeki “Age” sütununu seçerek, sütundaki sayısal verilerin çeyrekler arası mesafeleri, minimum ve maksimum değerleri, olası aykırı değerleri vb. görselleştiren bir kutu grafiği çizer. Bu grafiği kullanarak, veri setindeki “Age” sütunundaki dağılımı görsel olarak anlayabiliriz.

```
doldurulmus_veri.Age.plot.box()
```

```
Q1 = doldurulmus_veri.Age.quantile(q=0.25)
Q3 = doldurulmus_veri.Age.quantile(q=0.75)
IQR = Q3 - Q1
```

```
print("Q1: {}, Q3: {}, IQR: {}".format(Q1, Q3, IQR)) # Q1: 21.0, Q3: 39.0, IQR: 18.0
```

```
alt_sinir = Q1-1.5*IQR
ust_sinir = Q3 + 1.5*IQR
```

```
alt_sinir, ust_sinir # (-6.0, 66.0)
```

```
# alt sınırın altında veri var mı?
doldurulmus_veri.Age[doldurulmus_veri.Age < alt_sinir]
```

```
# üst sınırın üzerinde veri var mı?
doldurulmus_veri.Age[doldurulmus_veri.Age>ust_sinir]
```

```
# alt sınır ve üst sınırı aşan verileri traşladık
doldurulmus_veri.Age[doldurulmus_veri.Age > ust_sinir] = ust_sinir
doldurulmus_veri.Age[doldurulmus_veri.Age < alt_sinir] = alt_sinir
```

```
# yeniden kontrol edildiğinde
doldurulmus_veri.Age.plot.box()
```

Z-Skoru yöntemi

Z-skoru yöntemi, verilerin ortalama ve standart sapma değerlerini kullanarak aykırı değerleri belirlemek için kullanılan bir yöntemdir. Bu yöntemde, verilerin standart sapma değerlerine göre ne kadar uzağa olduklarına göre bir skor verilir. Skorların dağılımı normal dağılıma göre ayarlanır ve genellikle 3'ün üzerinde veya altında kalan skorlar aykırı değer olarak kabul edilir.

Z-skoru hesaplamak için, her bir veri noktasının ortalama değerden uzaklığının standart sapmaya bölünmesi gerekir. Formül aşağıdaki gibidir:

Z = (x - mean) / std

Burada:

- x, herhangi bir veri noktası
- mean, veri setinin ortalaması
- std, veri setinin standart sapması

Örneğin, bir veri setinde ortalama değer 50 ve standart sapma 10 ise, bir veri noktası 70 olsun. Z-skoru aşağıdaki şekilde hesaplanabilir:

Z = (70 - 50) / 10 = 2

Bu veri noktası, veri setinin normal dağılımına göre 2 standart sapma yukarıdadır. Eğer bu değer 3'ten büyükse veya -3'ten küçükse, bu veri noktası aykırı değer olarak kabul edilir.

Pandas kütüphanesi ile Z-skoru yöntemi kullanılarak aykırı değerleri bulmak için, "zscore" fonksiyonu kullanılabilir. Bu fonksiyon, belirli bir sütundaki her bir veri noktasının Z-skorunu hesaplar ve bunları yeni bir sütuna ekler. Ardından, belirli bir eşik değerine göre aykırı değerleri seçmek için bu sütunu filtrelemek mümkündür. Örneğin:

```
# Verileri yükle
df = doldurulmus_veri

# 'degerler' sütunundaki her bir veri noktasının Z-skorunu hesapla
df['z_score'] = (df['Age'] - df['Age'].mean()) / df['Age'].std()

# Z-skoru eşiği 3 olan aykırı değerleri seç
outliers = df.loc[df['z_score'].abs() > 3, :]
```

```
outliers
```

Veri dönüşüm işlemleri

Pandas kütüphanesi, verileri manipüle etmek için birçok işlevsellik sunar. Bu işlevsellikler arasında veri dönüşümleri, normalizasyon, standardizasyon ve kesikli hale getirme gibi işlemler de bulunur.

- Veri dönüşümleri: Veri dönüşümleri, verilerin farklı birimler veya formatlarla sunulduğu durumlarda kullanışlıdır. Pandas'ta veri dönüşümleri, "apply" ve "map" fonksiyonları gibi farklı yöntemlerle gerçekleştirilebilir.
- Normalizasyon: Normalizasyon, verilerin arasındaki değerleri sınırlandırarak verilerin dağılımını düzenler. Bu işlem sayesinde farklı birimlerdeki veriler aynı ölçekte karşılaştırılabilir hale gelir. Normalizasyon için genellikle Min-Max Normalizasyonu kullanılır.
- Standardizasyon: Standardizasyon, verilerin arasındaki standart sapmaları kullanarak verilerin dağılımını düzenler. Bu işlem sayesinde farklı birimlerdeki verilerin dağılımları daha anlamlı hale gelir. Standardizasyon için genellikle Z-skoru yöntemi kullanılır.
- Kesikli hale getirme: Kesikli hale getirme, sürekli sayısal verileri belirli aralıklarla kesikli hale getirerek verilerin anlamlı hale gelmesini sağlar. Bu işlem, özellikle makine öğrenimi modellerinde kategorik değişkenlerle birlikte kullanılmak üzere faydalıdır.

Örnek olarak, veri setindeki bir sütunu normalizasyon, standardizasyon veya kesikli hale getirme işlemlerinden biri ile işleyebilirsiniz. Örneğin, "Age" sütununu 0 ile 1 arasındaki değerlere normalizasyon yaparak işleyebilirsiniz:

Normalizasyon

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df['Age_Normalized'] = scaler.fit_transform(df[['Age']])
```

Bu kodda, "Age" sütununun değerleri Min-Max Normalizasyon yöntemi ile 0 ile 1 arasındaki değerlere dönüştürülüp "Age_Normalized" adlı yeni bir sütunda saklanır.

Ya da benzer işlem kendi yazacağımız denklem üzerinden de gerçekleştirilebilir. Aşağıdaki kodlar bu amacı yerine getirmektedir.

```
yas = doldurulmus_veri.Age
normalize_edilmis = (yas - yas.min())/(yas.max()-yas.min())
normalize_edilmis.describe()
```

Standardizasyon

Tablodaki ‘Salary’ sütununu standardize etmek için örnek bir pandas kodu aşağıdaki gibi olabilir:

```
from sklearn.preprocessing import StandardScaler

# Örnek bir veri seti
data = {'Name': ['Ali', 'Veli', 'Ahmet', 'Mehmet'],
        'Age': [27, 31, 25, 29],
        'Salary': [5000, 7000, 4500, 6000]}

# Verileri bir pandas DataFrame nesnesine dönüştürme
df = pd.DataFrame(data)

# Verileri standardize etme
scaler = StandardScaler()
df['Standardized_Salary'] = scaler.fit_transform(df[['Salary']])

# Sonuçları yazdırma
print(df)
```

Bu kodda, `StandardScaler` fonksiyonu ile `Salary` sütunu standart normal dağılım (ortalama 0 ve standart sapma 1) haline getiriliyor. Bu işlem, tüm veri setinin aynı ölçekte olmasını sağlamak için yapılır ve makine öğrenmesi modelleri gibi bazı analizlerde gereklidir. Yeni standartlaştırılmış sütun, `Standardized_Salary` olarak adlandırılır ve DataFrame’e eklenir.

Kesikli hale getirme

Kesikli hale getirme (discretization), bir sürekli değişkenin belirli aralıklara ayrılması işlemidir. Bu işlem sayesinde veriler daha kolay yorumlanabilir ve anlaşılabilir hale gelir.

Pandas kütüphanesi ile kesikli hale getirme işlemi `cut()` fonksiyonu ile yapılabilir. Bu fonksiyon, bir aralık serisini kesikli kategorik bir değişkene dönüştürür. Örneğin, bir veri setindeki yaş değişkenini belirli aralıklara bölerek yaş kategorileri elde etmek için kullanılabilir.

Aşağıda örnek bir kesikli hale getirme işlemi bulunmaktadır:

```
# Örnek veri seti
veri = pd.DataFrame({'yaş': [25, 36, 42, 29, 51, 46, 39, 33, 28, 24]})

# Yaş aralıklarını belirleme
araliklar = [0, 30, 40, 60]

# Yaş değişkenini kesikli hale getirme
veri['yaş_kategorisi'] = pd.cut(veri['yaş'], araliklar)

print(veri)
```

Bu kod örneği, `yaş` değişkenindeki her bir değeri belirtilen aralıklara göre kesikli hale getirir ve yeni bir `yaş_kategorisi` değişkeni oluşturur. Çıktı aşağıdaki gibi olacaktır:

```
   yaş yaş_kategorisi
0   25        (0, 30]
1   36       (30, 40]
2   42       (40, 60]
3   29        (0, 30]
4   51       (40, 60]
5   46       (40, 60]
6   39       (30, 40]
7   33       (30, 40]
8   28        (0, 30]
9   24        (0, 30]
```

Görüldüğü gibi, her bir yaş değeri, belirlenen aralıklara göre bir kategoriye atanmıştır. Bu şekilde, `yaş` değişkeni daha anlamlı ve yorumlanabilir bir hale getirilmiştir.

```
yas_kategorileri = ["cocuk", "genc", "yasli"]
pd.cut(x=veri['yaş'], bins=araliklar, labels=yas_kategorileri)
```

One-hot encoding

One-Hot Encoding, kategorik değişkenlerin (categorical variables) makine öğrenmesi algoritmalarında kullanılabilmesi için sayısal veriye dönüştürülmesi işlemidir. Bu işlem, her kategorik değişkenin ayrı bir özellik olarak ele alındığı ve bu özelliklerin sadece 0 veya 1 değerleri alabildiği yeni bir veri kümesi oluşturur. Bu yöntem, sınıflandırma algoritmalarının daha doğru sonuçlar vermesine yardımcı olur.

Örnek olarak, bir “renk” değişkeni düşünelim. Bu değişken “kırmızı”, “yeşil” ve “mavi” gibi 3 farklı kategorik değer alabilsin. One-Hot Encoding ile bu değişken, “renk_kırmızı”, “renk_yeşil” ve “renk_mavi” olmak üzere 3 farklı özellik haline dönüştürülür. Her bir özellik, orijinal “renk” değişkeninin o değerine sahipse 1, diğer durumlarda 0 değerini alır.

Pandas kütüphanesi, One-Hot Encoding işlemi için get_dummies() fonksiyonunu sağlar. Bu fonksiyon, kategorik değişkenleri otomatik olarak bulur ve yeni özellikler haline dönüştürür. Örnek kullanım aşağıdaki gibidir:

```
# Örnek veri seti
data = {'renk': ['kırmızı', 'mavi', 'yeşil', 'mavi', 'kırmızı']}

# Veri setini DataFrame'e dönüştürme
df = pd.DataFrame(data)

# One-Hot Encoding işlemi
one_hot_encoded = pd.get_dummies(df['renk'])

# Yeni özellikleri DataFrame'e ekleme
df = pd.concat([df, one_hot_encoded], axis=1)

# Sonuçları görüntüleme
print(df)
```

Benzer işlem `titanic` veri setine de yapılabilir.

```
veri_seti.Embarked.head()
```

```
veri_seti.Embarked.value_counts()
```

```
# Kukla değişken - one-hot encoding
pd.get_dummies(veri_seti.Embarked).head()
```

Formül Uygulama

Pandas kütüphanesi, `map()` ve `apply()` gibi fonksiyonlar aracılığıyla veri işleme işlevselliği sağlar.

`map()` fonksiyonu, bir pandas Serisi'ndeki tüm öğeleri değiştirmek için kullanılır. Bu işlev, bir sözlük, işlev veya Seri nesnesi alabilir ve Seri öğelerini değiştirmek için bu nesneleri kullanır. Örneğin aşağıdaki kod, Salary Serisi'deki tüm öğelere 200.000 artırarak ve sonucu ekrana yazdıracaktır:

```
calisan_verileri.Salary.map(lambda x: x+200000) # Salary lere toplu artış
```

`apply()` fonksiyonu ise, bir DataFrame'in bir sütununda veya satırında işlevleri uygulamak için kullanılır. `apply()` fonksiyonu, `map()` fonksiyonuna benzer şekilde işlevlerin uygulanmasında kullanılır ancak, DataFrame veri yapısında uygulanır. Aşağıdaki kod, istenilen DataFrame sütununda bir yazılan fonksiyonun işlemini yapacak ve sonucu ekrana yazdıracaktır.

```
def fonksiyon_ismi(x):
    return x + 200000
```

```
calisan_verileri.Salary.apply(fonksiyon_ismi)
```

```
def sayi_formatla(x):
    return f"{x:.1f}"

calisan_verileri["Bonus %"].apply(sayi_formatla)
```

Aşağıdaki örneklerde ise pandas veri çerçevelerine yeni bir sütun eklemek için işlemler gösterilmiştir. Yeni bir sütun oluşturmak için iki aşama gereklidir:

- Yeni bir sütun oluşturmak ve ona bir ad vermek

- Yeni sütunun her bir hücresine değer atamak

Bunun için [] operatörü ve atama işlemi = kullanılır. Örneğin, aşağıdaki kod bir veri çerçevesine yeni bir sütun ekleyecek ve içerisine sabit bir "herhangi bir değer" yazdıracaktır:

```
calisan_verileri['yeni_sutun'] = 'herhangi bir değer'
calisan_verileri.head()
```

```
calisan_verileri['Yeni Sütun'] = np.random.randint(
    low=10, high=20,
    size=calisan_verileri.shape[0])  # yeni sütun rassal verilerden oluşturuldu
```

Örneğin, aşağıdaki kod "Yeni Sütun" sütunundaki her bir değeri 2 ile çarpacaktır:

```
calisan_verileri['Yeni Sütun'] = calisan_verileri['Yeni Sütun'] * 2
```

Burada, calisan_verileri['Yeni Sütun'] * 2 ifadesi, Yeni Sütun sütunundaki her bir değeri 2 ile çarpar ve sonucunu tekrar calisan_verileri['Yeni Sütun'] sütununa atar.

Nüfusu yüksek olan şehirleri bir sütunda tutmak istersek şu şekilde bir kod yazabiliriz.

```
sehirler['nufusu_yuksek'] = sehirler.nufuslar > 1000000
```

`sort_values()` fonksiyonu, bir veri çerçevesindeki sütunlara göre sıralama yapmaya yarayan bir fonksiyondur. Bu fonksiyon, veri çerçevesindeki belirli bir sütuna göre verileri küçükten büyüğe veya büyükten küçüğe doğru sıralama yapar.

```
calisan_verileri.sort_values(by=['Salary', 'Bonus %'], ascending=False)
```

`rename()` fonksiyonu, bir veya daha fazla ekseni, satır veya sütun etiketlerini yeniden adlandırmak için kullanılır.

Fonksiyonun genel kullanımı aşağıdaki gibidir:

```
calisan_verileri.rename(columns={
    'Bonus %': 'Bonus__%',
    'Team': 'Teams'
}).head()
```

Veri Seçme ve Verilere Ulaşma

Dosya yüklendikten ve veri setini anladıktan sonra, bazı verilere erişmek isteyebiliriz. Pandas, veri seçme, indeksleme ve filtreleme işlemleri için birçok farklı yöntem sunar. Bu yöntemlerin en sık kullanılanları şunlardır:

- `İndeksleme operatörü ([])` : Bu operatör ile belirli bir sütun ya da satır seçilebilir. Örneğin, veri çerçevesindeki 'Name' sütununu seçmek için df['Name'] kullanılır.
- `.loc[]` : Bu metot ile satır ve sütun isimleri veya konumlarına göre seçim yapılabilir. Örneğin, veri çerçevesindeki 3. satırı ve 'Age' sütununu seçmek için df.loc[3, 'Age'] kullanılır.
- `.iloc[]` : Bu metot ile satır ve sütun konumlarına göre seçim yapılabilir. Örneğin, veri çerçevesindeki 2. satırı ve 3. sütunu seçmek için df.iloc[1, 2] kullanılır.
- `Boolean indeksleme` : Bu metot ile belirli koşulları sağlayan satırlar seçilebilir. Örneğin, veri çerçevesindeki yaş değeri 30'dan büyük olan satırların seçimi için df[df['Age'] > 30] kullanılır.
- `.query()` : Bu metot ile belirli koşullara göre satırlar seçilebilir. Örneğin, veri çerçevesindeki yaş değeri 30'dan büyük olan satırların seçimi için df.query('Age > 30') kullanılır.
- `.isin()` : Bu metot ile belirli bir değere sahip olan satırlar seçilebilir. Örneğin, veri çerçevesindeki 'Gender' sütununda 'Female' değerine sahip olan satırların seçimi için df[df['Gender'].isin(['Female'])] kullanılır.
- `.between()` : Bu metot ile belirli bir aralıkta olan değerlere sahip olan satırlar seçilebilir. Örneğin, veri çerçevesindeki yaş değeri 20 ile 30 arasında olan satırların seçimi için df[df['Age'].between(20, 30)] kullanılır.

Bu yöntemlerin her biri, farklı durumlar için kullanışlıdır ve veri çerçevelerinde veri seçimi, indeksleme ve filtreleme işlemlerinin kolay ve esnek bir şekilde yapılmasına olanak tanır.

```
print(calisan_verileri["Team"])  # sadece Team sütununu getirir.
calisan_verileri[["Team", "Bonus %"]]  # iki sütunu birlikte getirir
```

Ayrıca, belli bir koşulu sağlayan satırlara erişmek için de `loc[]` fonksiyonunu kullanabiliriz. Bu fonksiyon, koşulu sağlayan satırları getirir.

iloc

`iloc` fonksiyonu, verilerin indeks numaralarına göre erişim sağlar. İndeks numarası, satırın konumunu belirtir ve sütun adı belirtilmezse tüm sütunlar döndürülür. Örneğin, aşağıdaki DataFrame nesnesinde, "isim", "yaş" ve "şehir" sütunlarındaki 1. indeksteki satırlara erişmek istediğimizi varsayalım:

```
data = {
    'isim': ['Ahmet', 'Mehmet', 'Ali', 'Ayşe'],
    'yaş': [28, 22, 30, 25],
    'şehir': ['İstanbul', 'Ankara', 'İzmir', 'Bursa']
}

df = pd.DataFrame(data)

# İlk satırlara erişim
print(df.iloc[1])  # Sadece bir satır döndürür
print(df.iloc[[1]])  # Bir DataFrame nesnesi döndürür
print(df.iloc[[1, 2]])  # İki satırı içeren bir DataFrame nesnesi döndürür

# İlk satırlara ve belirli sütunlara erişim
print(df.iloc[1, 0])  # Sadece bir değer döndürür
print(df.iloc[
    [1, 2],
    0:2])  # İki satırı ve ilk iki sütunu içeren bir DataFrame nesnesi döndürür
```

loc

`loc` fonksiyonu, satırların etiket veya koşullara göre erişim sağlar. Etiket veya koşul belirtilmezse, tüm satırlar döndürülür. Örneğin, aşağıdaki DataFrame nesnesinde, “isim”, “yaş” ve “şehir” sütunlarındaki “Mehmet” ve “Ali” isimli satırlara erişmek istediğimizi varsayalım:

```
data = {
    'isim': ['Ahmet', 'Mehmet', 'Ali', 'Ayşe'],
    'yaş': [28, 22, 30, 25],
    'şehir': ['İstanbul', 'Ankara', 'İzmir', 'Bursa']
}

df = pd.DataFrame(data)

# Etiket veya koşullara göre erişim
print(df.loc[df['isim'] == 'Mehmet'])  # Bir DataFrame nesnesi döndürür
print(df.loc[df['isim'].isin(
    ['Mehmet', 'Ali'])])  # İki satırı içeren bir DataFrame nesnesi döndürür

# Etiket veya koşullara göre erişim ve belirli sütunlar
print(df.loc[df['isim'] == 'Mehmet', 'yaş'])  # Sadece bir değer döndürür
print(df.loc[df['isim'].isin(['Mehmet', 'Ali']), ['yaş', 'şehir']])  #
```

- df[sütun]: DataFrame’den tek bir sütun veya sütun dizisi seçin; özel durum kolaylıkları: Boolean array (satırları filtreleme), slice (satırları dilimleme) veya Boolean DataFrame (bir kriterle değerleri ayarlama)
- df.loc[satırlar]: Etiketleme ile DataFrame’den tek bir satır veya alt küme seçin
- df.loc[:, sütunlar]: Etiketleme ile tek bir sütun veya sütun alt kümesini seçin
- df.loc[satırlar, sütunlar]: Hem satır(lar) hem de sütun(lar) etiketle seçin
- df.iloc[satırlar]: DataFrame’den tek bir satır veya satır alt kümesini konumla seçin
- df.iloc[:, sütunlar]: Tamsayı pozisyonu ile tek bir sütun veya sütun alt kümesini seçin
- df.iloc[satırlar, sütunlar]: Hem satır(lar) hem de sütun(lar) tamsayı pozisyonu ile seçin
- df.at[satır, sütun]: Bir satır ve sütun etiketiyle tek bir skaler değeri seçin

- df.iat[satır, sütun]: Bir satır ve sütun pozisyonu (tamsayılar) ile tek bir skaler değeri seçin

Sonuç

Bu bölümde, pandas'ın önemli bir araç olmasını sağlayan temel özelliklerini inceledik. Uygulamalı örneklerle kütüphanenin yeteneklerini keşfettik. Seriler ve Veri Çerçeveleri gibi veri nesneleri, int64, float ve object gibi veri tipleri ve veri dönüştürme işlemleri için farklı yöntemler hakkında bilgi sahibi olduk. Daha sonra, veri seçimi ve indeksleme gibi farklı yöntemlerle veri manipülasyonu gerçekleştirdik.

Görsel Veri Analizine Giriş

Bu bölümde, Python'da en yaygın kullanılan ve güçlü veri görselleştirme kitaplıklarından biri olan Matplotlibkullanarak veri görselleştirme dünyasına giriş yapacağız. Matplotlib, Python programlama dili için bir grafik çizme kütüphanesidir ve verilerinizi görsel olarak temsil etmek için kullanılır. Bu kütüphane, doğrusal grafikler, dağılım grafikleri, pasta grafikleri, çubuk grafikleri ve daha birçok grafik türü oluşturmanızı sağlar. Matplotlib, ücretsiz ve açık kaynaklı bir yazılımdır ve Python'da veri analizi, makine öğrenimi, yapay zeka ve daha birçok alanda sıklıkla kullanılır. Bu eğitimde, Matplotlib kütüphanesinin temellerini öğreneceksiniz ve verilerinizi nasıl etkileyici grafiklere dönüştürebileceğinizi öğreneceksiniz.

Matplotlib Kütüphanesi

Matplotlib, 2 ve 3 boyutlu grafiklerin çizilmesi için geliştirilmiş, anasayfası (http://matplotlib.org) adresinde bulunan, özellikle bilimsel ve mühendislik alanlarındaki veri görselleştirmelerinde sıklıkla kullanılan bir kütüphanedir. Veri görselleştirme ihtiyacı verinin olduğu her alanda geçerlidir. Çünkü büyük veri kümelerindeki veriyi nitelendiren özelliklere ve ölçülere genel çerçeveden bakılmak istendiğinde görsel araçlara ihtiyaç duyulur. Örneğin bir veri setinin dağılımı ile ilgili histogram grafikleri hızlı bir izlenim imkanı sunar. Aşağıdaki görselde tablo ile sunulamayacak bilgilerin veri görselleştirme ile göze hitap edecek şekilde sunulması için araçlar gösterilmiştir.

Matplotlib kütüphanesi NumPy üzerine kurulmuş bir kütüphanedir. Bu nedenle Matplotlib kütüphanesinin çağrılmasından önce NumPy kütüphanesinin de yüklü kütüphaneler arasında olması gereklidir. Matplotlib, verileri görselleştirmek için kullanılan güçlü bir kütüphanedir. Bu nedenle, Matplotlib'ın sunduğu avantajları kullanarak, verilerinizi etkileyici grafiklere dönüştürebilir ve daha kolay bir şekilde analiz edebilirsiniz. Kısaca Matplotlib kütüphanesinin önemli avantajlarını şu şekilde sunabiliriz:

- Basit ve Kullanımı Kolay: Matplotlib, kullanımı kolay bir arayüze sahiptir ve temel grafik çizme işlevleri için hızlı ve basit bir şekilde kullanılabilir.
- Geniş Kapsamlı Grafikler: Matplotlib, doğrusal grafikler, dağılım grafikleri, pasta grafikleri, çubuk grafikleri ve daha birçok grafik türü oluşturmanızı sağlar. Bu nedenle, verilerinizi birçok farklı şekilde görselleştirebilirsiniz.
- Özelleştirilebilir Grafikler: Matplotlib ile grafiklerinizi özelleştirmek için birçok seçeneğe sahipsiniz. Renkler, çizgi stilleri, etiketler ve daha birçok şeyi kolayca özelleştirebilirsiniz.

- Çoklu Veri Kümeleri: Matplotlib, birçok veri kümesini aynı grafikte birleştirmenize olanak tanır. Bu, verileriniz arasındaki ilişkileri anlamak için çoklu grafikleri karşılaştırmanızı ve birleştirmenizi sağlar.
- Çoklu Çıktı Biçimleri: Matplotlib, grafikleri birçok farklı çıktı biçiminde kaydetmenizi sağlar. Grafikleri PNG, PDF, SVG ve daha birçok formatta kaydedebilirsiniz.
- Ücretsiz ve Açık Kaynaklı: Matplotlib, ücretsiz ve açık kaynaklı bir yazılımdır ve Python topluluğu tarafından sıkça kullanılan bir kütüphanedir.

Matplotlib, Python dilindeki paket yöneticisi olan pip ile kolayca yüklenebilir. İlk olarak, aşağıdaki kodu kullanarak Matplotlib kütüphanesini Jupyter notebook üzerinden yükleyebilirsiniz:

```
!pip install matplotlib
```

veya

```
!conda install matplotlib
```

Matplotlib Grafiklerine Giriş

Matplotlib ile görselleştirmelere geçmeden önce Matplotlib katmanlarından bahsetmekte fayda vardır. Matplotlib, çizimlerin oluşturulması ve düzenlenmesi için bir dizi katman sağlar. Aşağıda, Matplotlib'deki katmanların kısaca açıklamaları verilmiştir:

1. Figure: Figure, çizimlerin oluşturulduğu en üst düzey katmandır. Figure, genellikle birden fazla alt grafik (subplot) içerir. Birden fazla çizimi bir arada göstermek için kullanılabilir.
2. Axes: Axes, verilerin çizildiği katmandır. Grafikteki x ve y eksenleri gibi her şey, bu katmanda oluşturulur. Axes, tek bir grafik için birden fazla çizgi veya veri kümesi içerebilir.
3. Axis: Axis, x ve y eksenleri gibi her bir eksenin ayrı ayrı bulunduğu katmandır. Eksenlerdeki etiketler, işaretçiler ve çizgiler bu katmanda bulunur.
4. Artist: Artist, grafikte çizilen herhangi bir şeyi (çizgi, işaretçi, şekil, metin vb.) temsil eden bir katmandır. Grafikteki herhangi bir öğeyi özelleştirmek için bu katman kullanılabilir.

Matplotlib'deki bu katmanlar, grafik oluşturma ve özelleştirme sürecinde kullanılan araçlar ve işlevler sağlar. Bu katmanlar, Matplotlib'ın güçlü ve esnek bir grafik kütüphanesi olmasını sağlar.

Matplotlib'de bir grafik oluşturmak için ilk adım, bir "Figure" oluşturmaktır. "Figure", en üst düzey çizim katmanıdır ve grafikteki her şeyin (eksenler, etiketler, veriler vb.) bulunduğu alanı tanımlar. Aşağıda bir "Figure" anatomisi anlatılmaktadır:

1. Figure: Figure, en üst düzey grafik alanıdır. Eksenler, metin, çizgiler ve diğer tüm öğeler bu alan içinde bulunur.
2. Axes: Axes, Figure içindeki verilerin çizildiği alandır. Her “Figure” bir veya daha fazla “Axes” içerebilir.
3. Axis: “Axes” içindeki bir eksen. Her “Axes” nesnesi, iki veya üç eksen içerir: x eksenleri, y eksenleri ve (varsa) z eksenleri.
4. Tick: “Axis” üzerindeki işaretleyicilerdir. Eksenin konumunu ve etiketlerini belirlerler.
5. Label: “Axis” üzerindeki işaretçi etiketidir. Eksenin neyi ölçtüğünü belirtir.
6. Spine: “Axis” çerçevesi boyunca uzanan çizgi. Spine’lar, verilerin eksenlerdeki konumlarını gösteren işaretleyicilerin üzerine yerleştirilebilir.
7. Grid: Verilerin konumunu gösteren çizgiler. Grid, eksenlerin etrafında oluşturulabilir.
8. Title: “Axes” üzerindeki başlık. Grafik hakkında özet bilgi sağlar.
9. Legend: Grafikteki farklı veri kümelerini açıklayan etiketler. “Axes” içindeki bir konuma yerleştirilebilir.

Matplotlib’de, bir “Figure” oluşturduktan sonra, “Axes” nesneleri ekleyebilir ve “Axis”, “Tick”, “Label” vb. özellikleri özelleştirebilirsiniz.

Matplotlib kütüphanesini çağırma

Klasik bir Matplotlib kütüphane çağrılma yöntemi aşağıda verilmiştir.

```
import Matplotlib.pyplot as plt
```

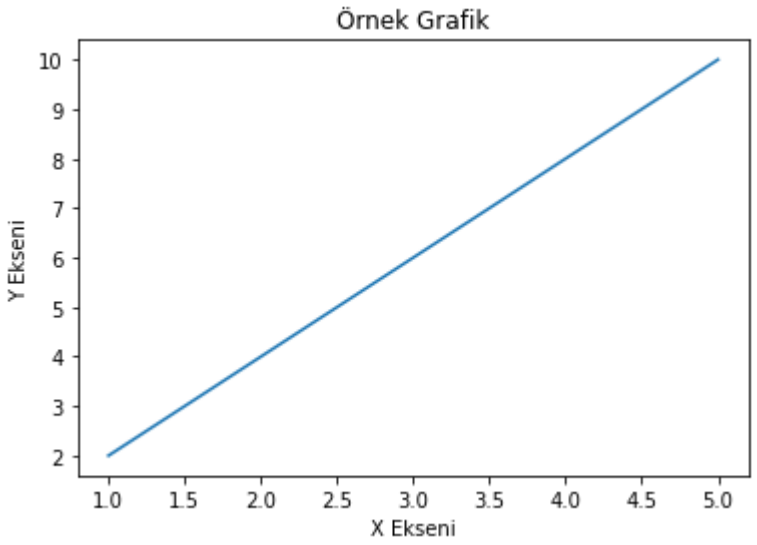
Bu kod `pyplot` isimli modülün çağrılmasını sağlar. Bu modül sayesinde Matplotlib grafikleri, eksenleri ve aksisleri çizdirilebilir. Aşağıdaki görselde pyplot ile çizilen figürlerin objeleri gösterilmiştir.

Aşağıdaki kodlar ile matplotlib kütüphanesi çağrılmış olunur. Aynı zamanda NumPy ve Pandas kütüphanelerinin de başlangıçta çağrılmasında fayda vardır.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# from matplotlib import pyplot as plt
```

Basit bir Matplotlib grafiği kodları aşağıda paylaşılmıştır. plt.plot() ile x değerlerine karşılık gelen y değerleri çizdirilebilir.

```
# x ve y verileri
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Grafik oluşturma
plt.plot(x, y)

# Grafik özellikleri
plt.title("Örnek Grafik")
plt.xlabel("X Ekseni")
plt.ylabel("Y Ekseni")

# Grafik gösterimi
plt.show()
```



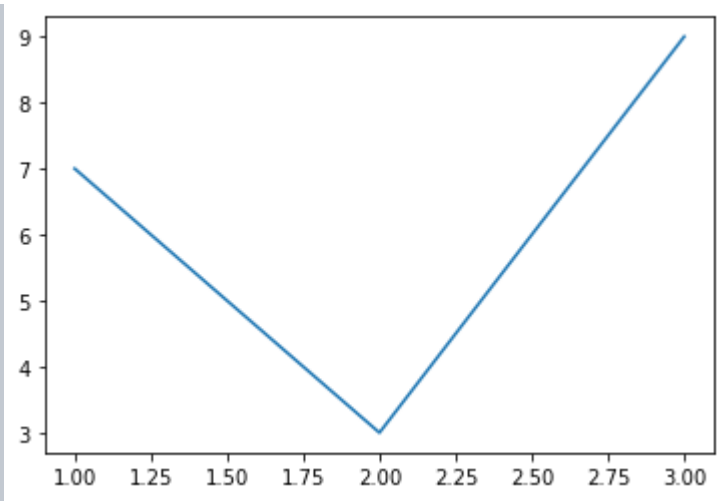
Bu örnekte, öncelikle `matplotlib.pyplot` modülü import edilir. Daha sonra, “x” ve “y” verileri tanımlanır. `plt.plot(x, y)` çağrısı ile, bu verileri içeren bir grafik oluşturulur.

Daha sonra, `plt.title()`, `plt.xlabel()` ve `plt.ylabel()` çağrıları ile grafik başlığı ve eksen etiketleri belirtilir. Son olarak, `plt.show()` çağrısı ile, oluşturulan grafik ekranda gösterilir.

Bu örnek, Matplotlib’in temel kullanımını göstermektedir. Grafik oluşturma sürecinde, “Axes” özelliklerini ve diğer özelleştirme seçeneklerini kullanarak grafikleri daha fazla özelleştirebilirsiniz.

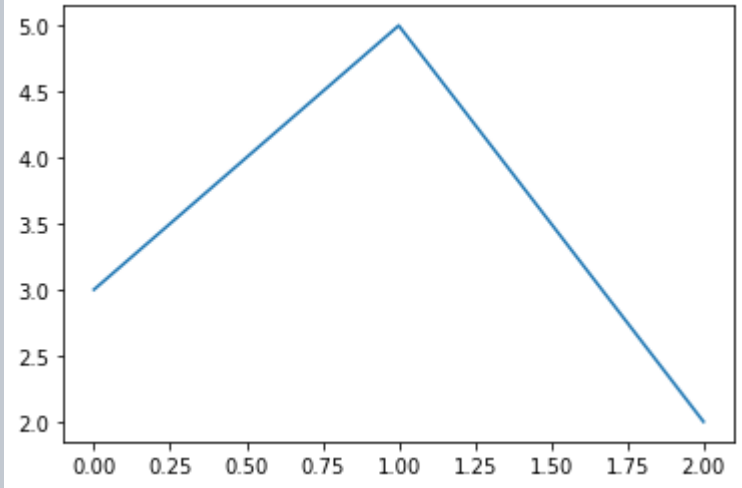
```
plt.plot([1,2,3],[7,3,9]) # (1,7), (2,3), (3,9) noktalarını birleştirir
```

```
[<matplotlib.lines.Line2D at 0x26793ee5a00>]
```



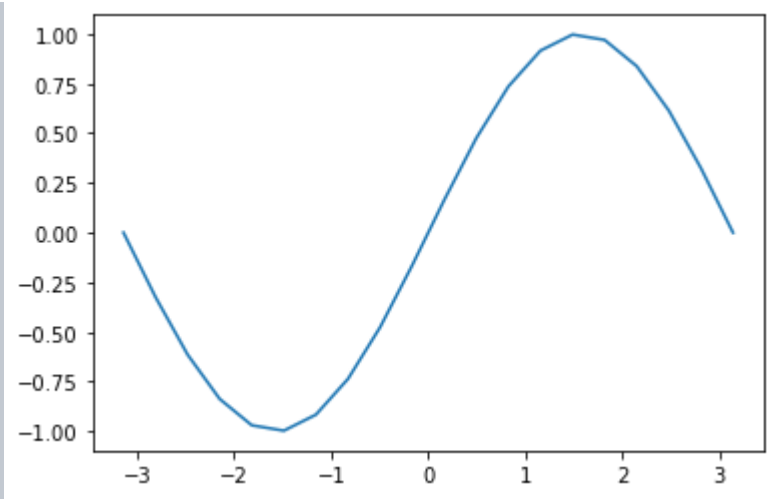

```
plt.plot([3, 5, 2])  # x ekseni verilmez ise varsayılan olarak 0-1-2... gelir
```

```
[<matplotlib.lines.Line2D at 0x26793f5fac0>]
```




```
x = np.linspace(-np.pi, np.pi, 20)  # -pi ile +pi arasında 20 adet değer
plt.plot(x, np.sin(x))
```

```
[<matplotlib.lines.Line2D at 0x26793fcea30>]
```



Grafiklere Özellik Ekleme

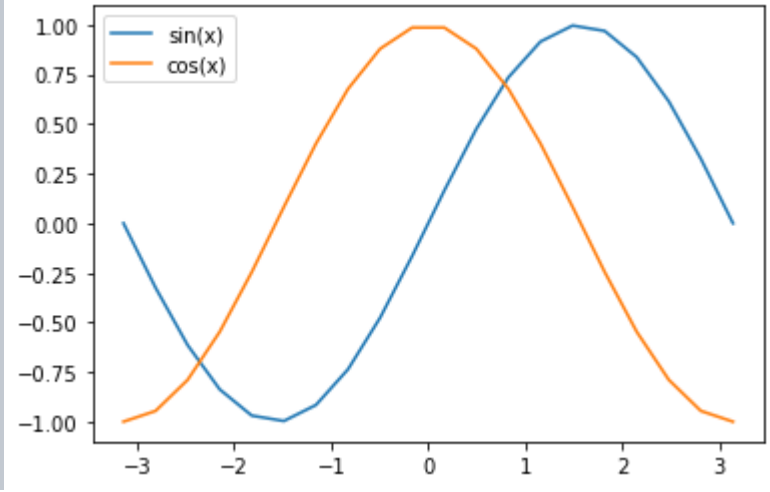
Az önce çizdirilen grafikler x, y eksenleri ve alınan değerlerin birleştirilmesi ile oluşturuldu. Matplotlib kütüphanesinde eğer bir özellik tanımı yapılmaz ise grafikler yalın halde gelir. Ancak grafikler yalın halde anlamsızdır. x, y eksenlerinin neyi gösterdiği, grafiğin başlığının ne olduğu, verilerin neler olduğu grafik üzerinde gösterilmelidir.

Eksen etiketi, seri etiketi ve başlık özelliği ekleme

`label` özelliği verilerin neler olduğunu gösterir. label özelliğinin grafik içerisinde gösterilmesi için `plt.legend()` fonksiyonu eklenmiştir.

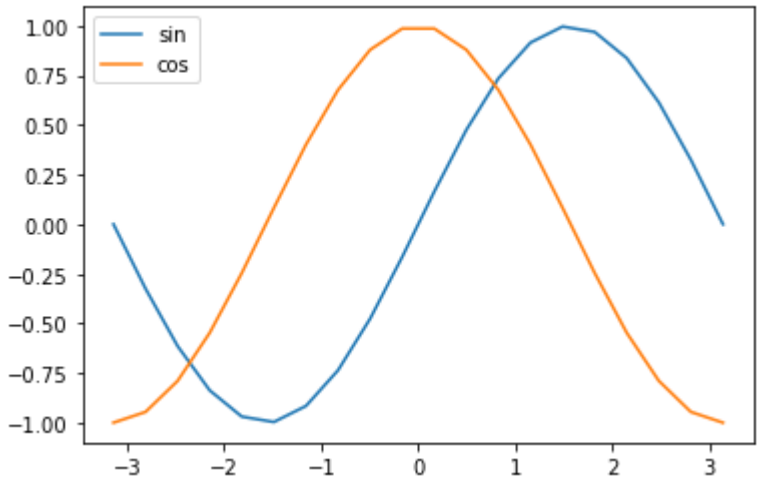
```
plt.plot(x,np.sin(x), label = 'sin(x)')
plt.plot(x,np.cos(x), label = 'cos(x)')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x26794041640>
```



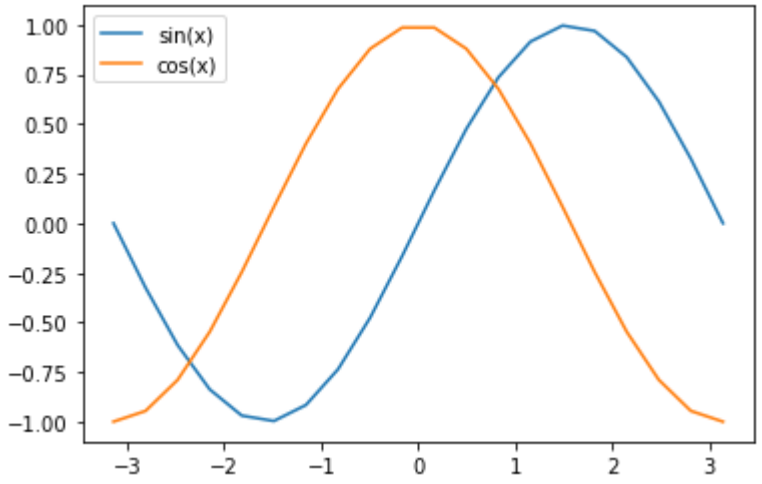
```
# Veya
plt.plot(x,np.sin(x))
plt.plot(x,np.cos(x))
plt.legend(('sin', 'cos'), loc='best')
```

```
<matplotlib.legend.Legend at 0x2679401a280>
```



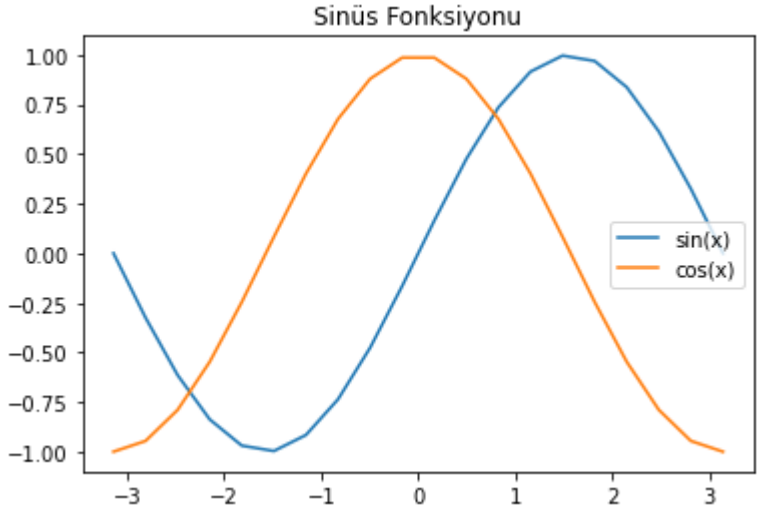
legend() fonksiyonu içerisine tanımlanacak parametreler ile etiketlerin konumu değiştirilebilir.

```
plt.plot(x,np.sin(x), label = 'sin(x)')
plt.plot(x,np.cos(x), label = 'cos(x)')
plt.legend(loc='best'); # upper left upper right
```



Grafiklerin üzerine başlık eklemek, grafiğin içeriği hakkında daha fazla bilgi verir. Başlık, `title()` işlevi kullanılarak eklenebilir. Örneğin:

```
plt.plot(x,np.sin(x), label = 'sin(x)')
plt.plot(x,np.cos(x), label = 'cos(x)')
plt.legend(loc=7); # upper left upper right
plt.title('Sinüs Fonksiyonu')
plt.show()
```

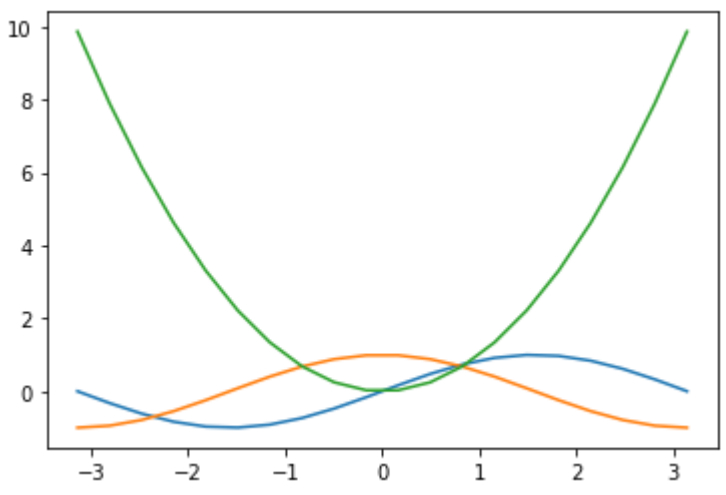


Etiket fontlarını değiştirme

Aşağıdaki örnekte birden fazla çizim aynı grafik üzerinde gerçekleştirilmiştir. Buna göre 3 farklı çizimi gösterdikten sonra etiketlerin fontları üzerinde değişiklikler gerçekleştirdik.

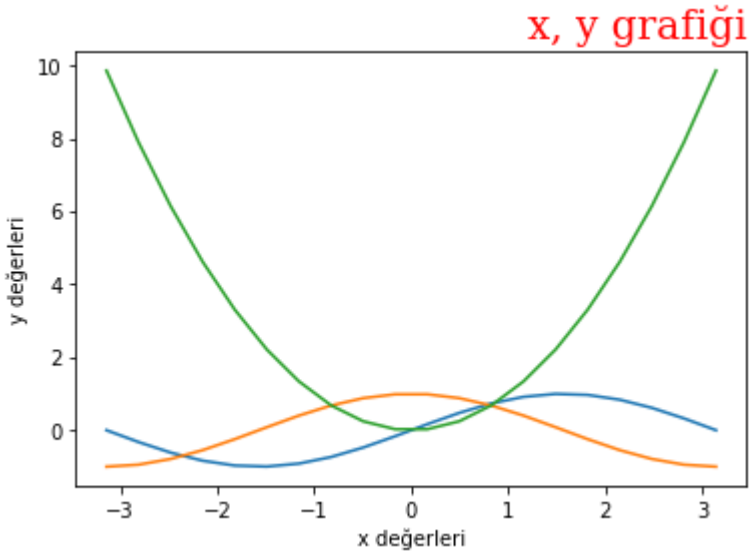
Renk kullanımında `plot()` işlevi, color parametresi ile çizgi rengini belirleyebilir. Ayrıca, hex renk kodlarını veya RGB veya RGBA renk değerlerini de kullanabilirsiniz. Örneğin:

```
# Ya da aynı figür üzerinde şekil çizmek istersek
plt.plot(x, np.sin(x), x, np.cos(x), x, x**2);  # ; fonksiyon yazmak istemezsek
```



```
# Etiketlerin fontlarını değiştirme
plt.plot(x, np.sin(x), x, np.cos(x), x, x**2)
plt.xlabel("x değerleri")
plt.ylabel("y değerleri")
plt.title("x, y grafiği",
          fontdict={
              'family': 'serif',
              'color': 'red',
              'size': 20
          },
          loc='right')
```

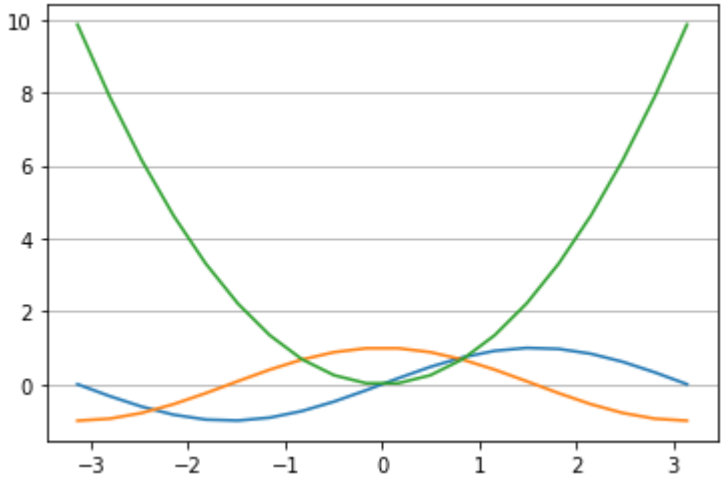
```
Text(1.0, 1.0, 'x, y grafiği')
```



Fontlarla ilgili daha detaylı bilgi şu adresten alınabilir.

Izgaraları gösterme

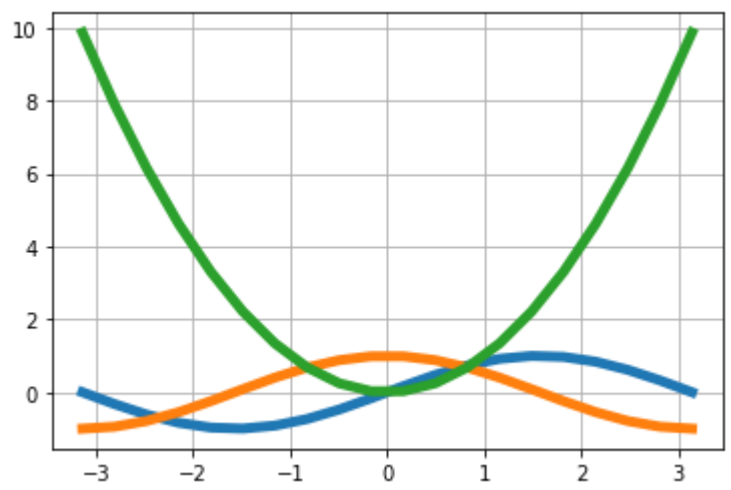
```
plt.plot(x, np.sin(x), x, np.cos(x), x, x**2)
plt.grid(True, axis='y') # axis='x' sadece x ekseninde ızgaralar gösterir
```



Renk ve çizgi stillerini değiştirme

Bu örnekte, “color”, “linestyle” ve “linewidth” parametreleri kullanılarak çizgi rengi, stil ve kalınlığı belirtilir.

```
# çizgilerin kalınlıklarının ayarlanması
plt.plot(x, np.sin(x), x, np.cos(x), x, x**2, linewidth=5);
plt.grid()
```




```
# çizgi stilleri
plt.plot(x, np.sin(x), linestyle='dashed', label='sin')
plt.plot(x, np.cos(x), linestyle='--', label='cos')
plt.plot(x, x**2, label='kare', linestyle='-.')
plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x267951ccbe0>
```

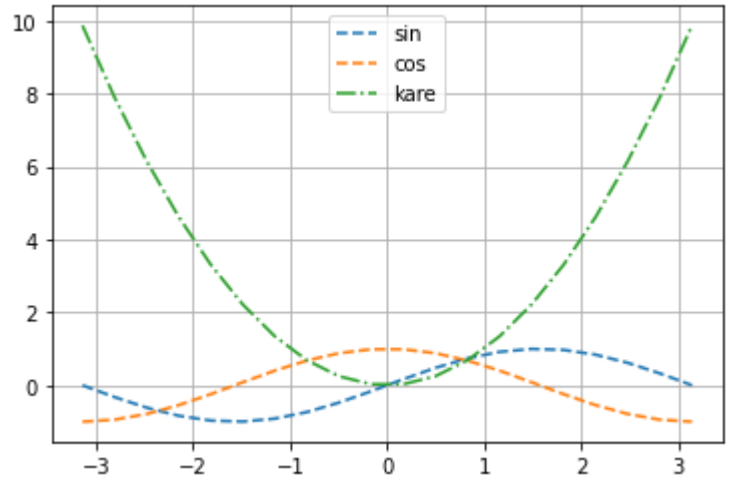


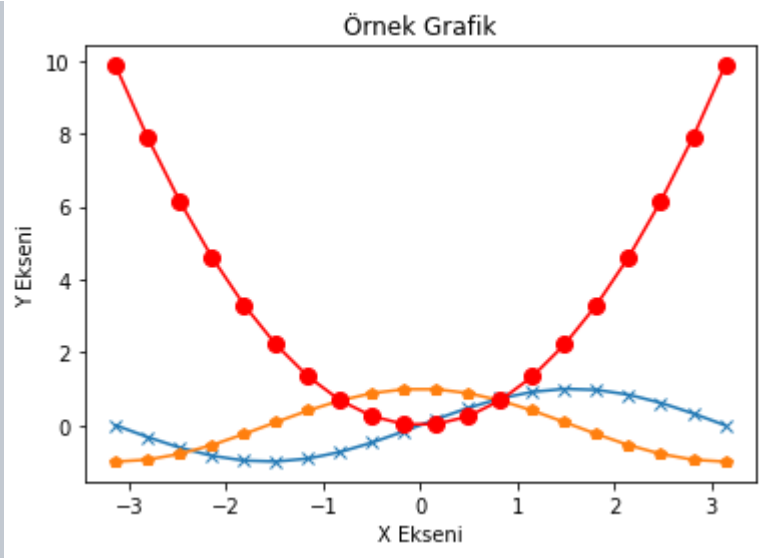
image.png

Diğer diğer parametreler için şu adresten bilgi alınabilir.

Marker özellikleri

Bu örnekte, “marker” ve “markersize” parametreleri kullanılarak çizgi üzerinde işaretleyici (marker) boyutu ve şekli belirtilir. Detaylı bilgi link

```
# Grafik oluşturma
plt.plot(x, np.sin(x), label='sin', marker = 'x')
plt.plot(x, np.cos(x), label='cos', marker= 'p')
plt.plot(x, x**2, label='kare',color='red', marker='o', markersize=8)

# Grafik özellikleri
plt.title("Örnek Grafik")
plt.xlabel("X Ekseni")
plt.ylabel("Y Ekseni")

# Grafik gösterimi
plt.show()
```



Matplotlib, çizgi komutlarının marker parametresi kullanılarak seçilen birden fazla kategoriye sahip markerları destekler:

- Doldurulmamış markerlar
- Doldurulmuş markerlar
- TeX sembollerinden oluşturulan markerlar
- Yollardan oluşturulan markerlar

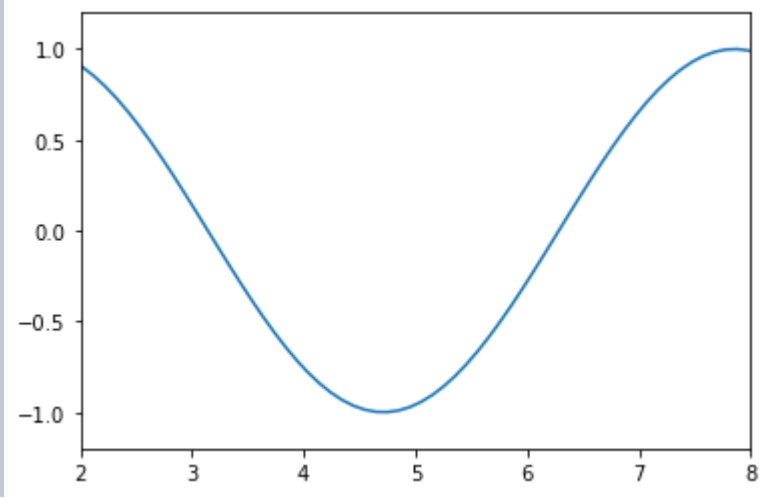
Doldurulmamış markerlar (Unfilled markers) Doldurulmuş markerlar (Filled) Marker fill styles TeX sembollerinden oluşturulan markerlar Yollardan oluşturulan markerlar

Eksenleri sınırlandırma ve işaretleme

`xlim` ve `ylim`, bir grafikte gösterilen x ve y eksenlerinin sınırlarını belirlemek için kullanılan fonksiyonlardır.

`xlim` ve `ylim` fonksiyonlarına verilen değerler, belirtilen eksenin en küçük ve en büyük değerlerini belirler. Örneğin, aşağıdaki kod bloğunda, xlim ve ylim kullanarak bir çizgi grafiğinin x ve y eksenlerinin sınırlarını ayarladık:

```
# x ve y verileri
x = np.linspace(0, 10, 100)
y = np.sin(x)

# çizgi grafiği çizdirme
plt.plot(x, y)

# x ve y eksenlerinin sınırlarını belirleme
plt.xlim([2, 8])
plt.ylim([-1.2, 1.2])

# grafiği gösterme
plt.show()
```

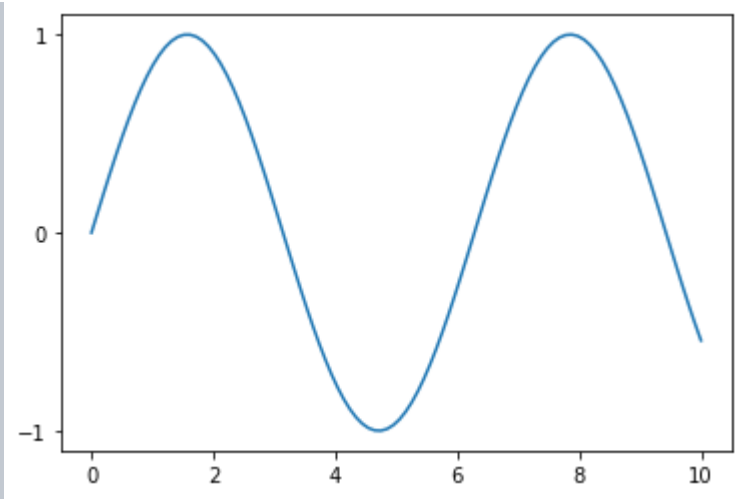


Bu örnekte, xlim([2, 8]) fonksiyonuyla x ekseni sınırları 2 ve 8 arasına, ylim([-1.2, 1.2]) fonksiyonuyla y ekseni sınırları -1.2 ve 1.2 arasına ayarlandı.

`xticks` ve `yticks`, bir grafikteki x ve y eksenlerinin işaretlenmesini (tick) kontrol etmek için kullanılan fonksiyonlardır.

`xticks` ve `yticks` fonksiyonları, aşağıdaki gibi kullanılabilir:

```
plt.plot(x, y)

# x ve y ekseni işaretlerini ayarlama
plt.xticks(np.arange(0, 11, step=2))
plt.yticks([-1, 0, 1])

# grafiği gösterme
plt.show()
```



Bu örnekte, xticks fonksiyonu ile x ekseni işaretlerinin aralıklarını belirliyoruz. np.arange(0, 11, step=2) ifadesi, x ekseni işaretlerinin 0'dan başlayarak 2'şer arttırılarak 10'a kadar olan sayılar olduğunu belirtir. yticks fonksiyonu ise, y ekseni işaretlerini belirliyoruz. [-1, 0, 1] ifadesi ile y ekseni işaretlerinin -1, 0 ve 1 olduğunu belirtiyoruz.

Renk Haritası (Colormap) Özellikleri

Matplotlib kütüphanesi, renk haritaları veya colormap olarak adlandırılan, verilerin renklendirilmesi için kullanılan özel renk paletlerini destekler. Colormap'lar, veri yoğunluğu veya sıcaklık, deniz seviyesi yüksekliği veya diğer özelliklerin belirtilmesi gibi verilerin özelliklerini vurgulamak için kullanılabilir.

Matplotlib'teki çeşitli colormap'lar, farklı renk paletleri ve renk geçişleri sunar. Bazı örnek colormap'lar şunlardır:

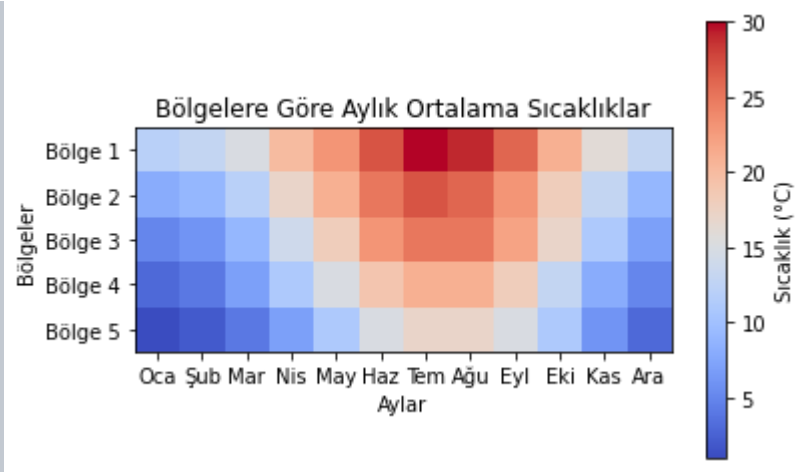
- viridis
- plasma
- inferno
- magma
- coolwarm

Colormap'lar genellikle `plt.imshow()` veya `plt.pcolor()` gibi işlevlerle kullanılır. Colormap özellikleri, cmap parametresi veya renk paleti adı olarak belirtilerek seçilebilir.

Bunun yanı sıra, colormap'ların skalasını özelleştirmek için farklı renk dönüşüm (color map transformation) işlevleri de kullanılabilir. Örneğin, `plt.Normalize()` işlevi, renk skalasının belirli bir aralığa ölçeklendirilmesine olanak tanır.

Aşağıdaki örnekte tablo verilerini renklendirmek için colormap kullanımına örnek olarak, bir heatmap oluşturalım. Veriler, farklı bölgelerin aylık ortalama sıcaklıklarını içeren bir numpy dizisinden alınacak.

```
# Veri dizisi oluşturma
data = np.array([[12, 13, 15, 20, 23, 27, 30, 29, 26, 21, 16, 13],
                 [8, 9, 12, 17, 21, 25, 27, 26, 23, 18, 13, 9],
                 [5, 6, 9, 14, 18, 23, 25, 25, 22, 17, 11, 7],
                 [3, 4, 7, 11, 15, 19, 21, 21, 18, 13, 8, 5],
                 [1, 2, 4, 7, 11, 15, 17, 17, 15, 11, 6, 3]])

# Renk haritası (colormap) özelleştirme
cmap = plt.cm.get_cmap('coolwarm')
norm = plt.Normalize(vmin=data.min(), vmax=data.max())

# Resim çizimi
plt.imshow(data, cmap=cmap, norm=norm)

# Renk skalası oluşturma
cbar = plt.colorbar()
cbar.ax.set_ylabel('Sıcaklık (°C)')

# Eksen etiketleri ekleme
plt.xticks(range(12), ['Oca', 'Şub', 'Mar', 'Nis', 'May', 'Haz', 'Tem', 'Ağu', 'Eyl',
'Eki', 'Kas', 'Ara'])
plt.yticks(range(5), ['Bölge 1', 'Bölge 2', 'Bölge 3', 'Bölge 4', 'Bölge 5'])
plt.xlabel('Aylar')
plt.ylabel('Bölgeler')

# Başlığı ekleme
plt.title('Bölgelere Göre Aylık Ortalama Sıcaklıklar')

# Resmi gösterme
plt.show()
```



Bu kod, `imshow()` işlevi kullanarak veri dizisini bir ısı haritası olarak gösterir. Veriler, coolwarm renk haritasına göre renklendirilir. `Normalize()` işlevi, verilerin minimum ve maksimum değerlerine göre normalleştirir. `colorbar()` işlevi, renk skalasını resmin yanına ekler. Sonuç olarak, verilerin özelliklerine ve sıcaklık değişimlerine göre vurgulanarak görsel olarak daha anlaşılır bir şekilde sunulur.

Çoklu Figürler Çoklu Matplotlib Grafikleri

Bazı durumlarda birden fazla grafik bir figür içerisinde verilmek istenebilir. Böylece tek seferde okuyucuya birden fazla durumla ilgili özet bilgi sunulmuş olunur. Daha önceden belirtilen Figure ve Axes kavramları kullanılarak Matplotlib içerisinde birden fazla grafik

subplot() fonksiyonu ile yerine getirilebilir.

Subplot() fonksiyonu ile istenilen sayıda satır ve sütunda grafikler figürlere eklenebilir.

Subplot fonksiyonundan önce matplotlib kütüphanesindeki iki farklı yaklaşımdan bahsetmekte fayda vardır. Bunlar;

MATLAB tipi pyplot ile grafik çizdirme

Pyplot ile çizdirilen grafik bir hücre için değiştirilebilir bir grafiktir. Başka hücrede yeni bir pyplot objesi eklenirse yeni grafikler çizdirilebilir. MATLAB programındakine benzer şekilde grafik çizdirme yöntemidir. Daha önce gösterilen grafikler pyplot ile çizdirilen grafiklerdi.

Nesne yönelimli şekilde grafik çizdirme.

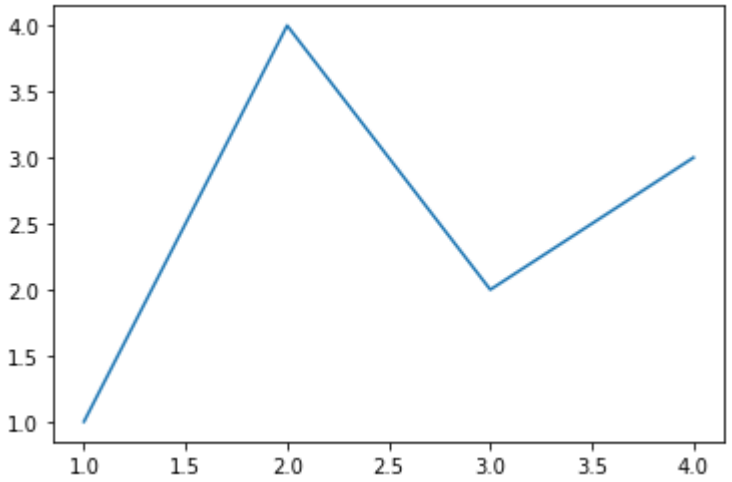
Matplotlib kütüphanesinin önemli özelliklerinden birisi de nesne yönelimli programlamaya izin vermesidir. Çizim alanı anlamına gelen `Figure` objesi Matplotlib kütüphanesinin tepesinde yer alır. Diğer elemanlar ise `Artist` diye bilinir ve aşağıdaki görselde gösterilen mavi yerler Artist olarak geçmektedir. Artistler Axes(Eksenler)den oluşur. Eksenler ise Axis öğelerinden oluşur.

Eksenler(Axes) grafikleri tutan objelerdir. Bir grafik çizdirebilmek için en az bir eksenin figür içerisine eklenmesi gerekmektedir. Axis'ler ise x ve y aksis i olabilir ya da üçüncü boyut varsa z axis i de devreye girer. Bu nedenle x veya y deki limitleri veya çizgileri ayarlamak için axis özelliğini kullanmak gerekecektir. Matplotlib ile daha gelişmiş ve pratik grafik çizimleri için bu hiyerarşiyi anlamak faydalı olacaktır.

Ayrıca nesne yönelimli yapı kullanılarak çok daha hızlı ve temiz grafikler çizilebilir. Hızlı grafik çizimleri için MATLAB stili grafik çizdirmek önerilse de daha detaylı işler için nesne yönelimli yapı önerilmektedir.

```
# basit bir matplotlib grafiği
fig, ax = plt.subplots()  # Tek eksenden oluşan bir figür oluşturmak için
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Eksen içerisine bir grafik çizdirmek için
```

```
[<matplotlib.lines.Line2D at 0x267954067c0>]
```

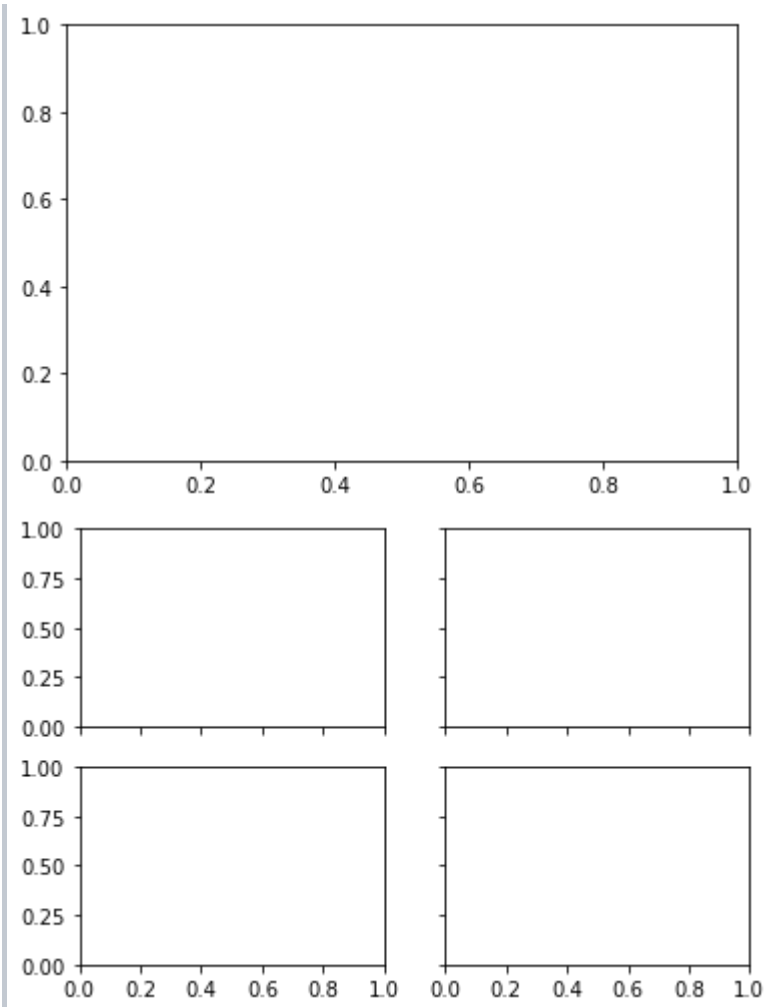



```
print(type(fig))
```

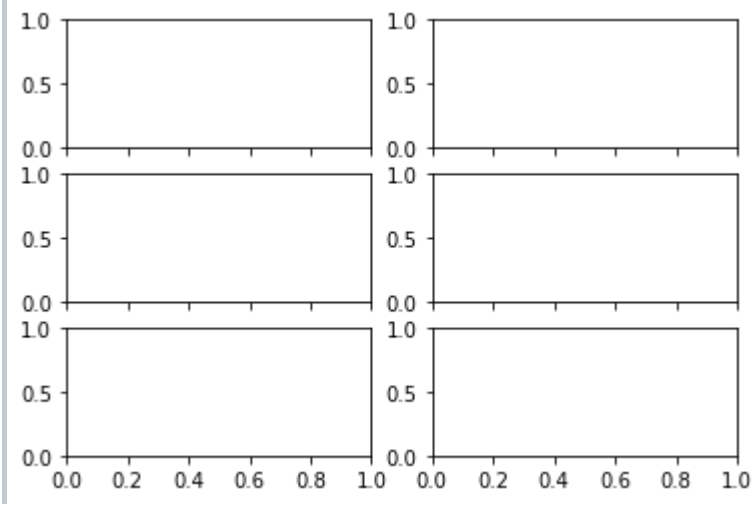
```
<class 'matplotlib.figure.Figure'>
```

```
fig = plt.figure()  # Eksenin olmadığı bir figür
fig, ax = plt.subplots()  # Tek eksenden oluşan bir figür
fig, axs = plt.subplots(2, 2, sharex= True, sharey=True)  # 2x2 li bir grafik
```

```
<Figure size 432x288 with 0 Axes>
```

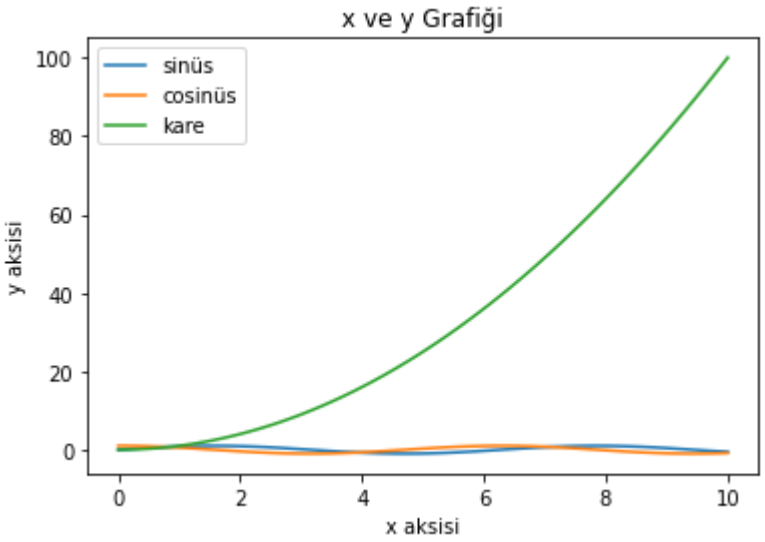


```
fig, axs = plt.subplots(nrows= 3, ncols=2, sharex=True)  # 2x1 li bir grafik ve ortak x
axis i
```



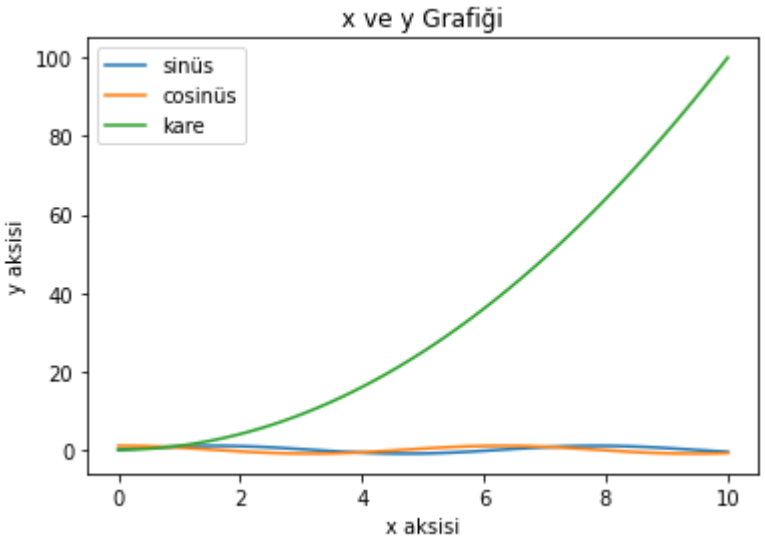
```
# Nesne yönelimli grafikler oluşturmak için aşağıdaki yapı kullanılabilir.
fig, ax = plt.subplots()  # Bir eksenden oluşan bir figür oluşturduk.
ax.plot(x, np.sin(x), label='sinüs')  # Eksenin içerisine grafik çizdirmek için
ax.plot(x, np.cos(x), label='cosinüs')  # Aynı eksene başka grafik çizdirmek için
ax.plot(x, x**2, label='kare')  # istenilen kadar grafik aynı eksene eklenebilir.
ax.set_xlabel('x aksisi')  # Eksene x etiketi koymak için
ax.set_ylabel('y aksisi')  # set_ylabel.
ax.set_title("x ve y Grafiği")  # Grafiğe başlık eklemek için.
ax.legend()  # Etiketleri göstermek için.
```

```
<matplotlib.legend.Legend at 0x267957a83d0>
```



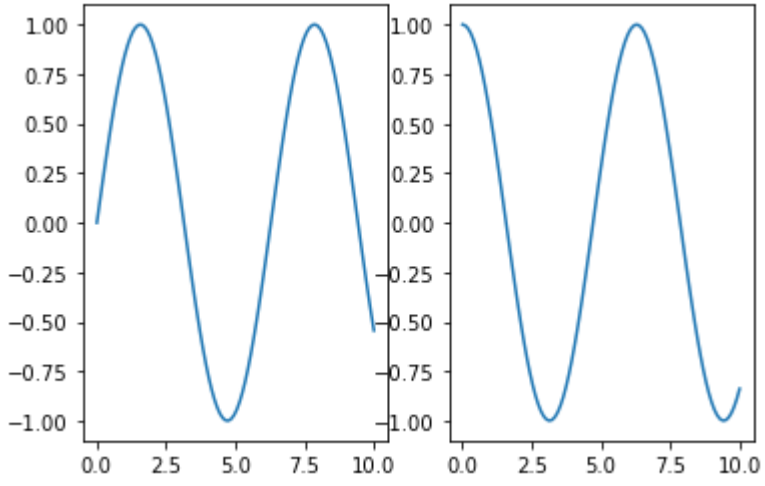

```
# ya da aynı işlemi daha önce gösterdiğimiz gibi de yapabilirdik.

plt.plot(x, np.sin(x), label='sinüs')  # Eksenin içerisine grafik çizdirmek için
plt.plot(x, np.cos(x), label='cosinüs')  # Aynı eksene başka grafik çizdirmek için
plt.plot(x, x**2, label='kare')  # istenilen kadar grafik aynı eksene eklenebilir.
plt.xlabel('x aksisi')  # set_xlabel değişti
plt.ylabel('y aksisi')  # set_ylabel değişti.
plt.title("x ve y Grafiği")  # set_title değişti.
plt.legend()  # Etiketleri göstermek için.
```

```
<matplotlib.legend.Legend at 0x267951b4e20>
```




```
fig, ax = plt.subplots(1, 2) # bir satır iki sütundan oluşan iki grafik ekledik.
ax[0].plot(x, np.sin(x)) # ilk eksene grafik çizdirmek için
ax[1].plot(x, np.cos(x)) # ikinci eksene grafik çizdirdik.
```

```
[<matplotlib.lines.Line2D at 0x26793709820>]
```

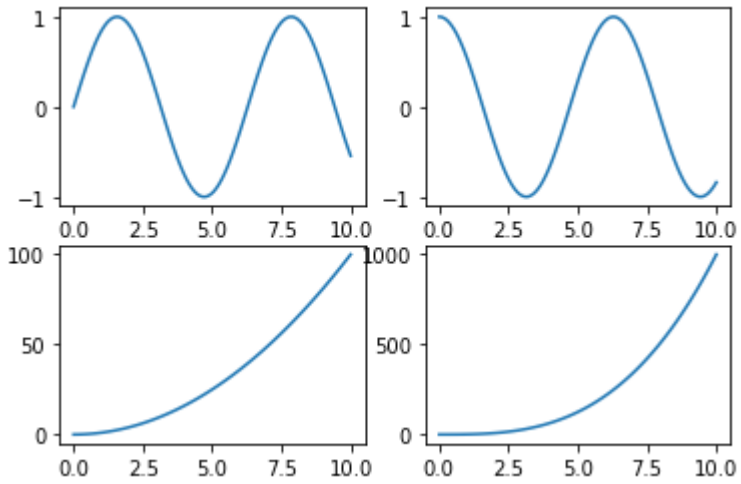



```
type(ax)
```

```
numpy.ndarray
```

```
fig, ax = plt.subplots(2, 2) # iki satır iki sütundan oluşan iki grafik ekledik.
ax[0,0].plot(x, np.sin(x), label='sin') # birinci satır birinci sütundaki grafik
ax[0][1].plot(x, np.cos(x)) # ikinci eksene grafik çizdirdik.
ax[1][0].plot(x, x**2)
ax[1][1].plot(x, x**3)
```

```
[<matplotlib.lines.Line2D at 0x267957df790>]
```




```
type(ax) # ax nesnesi bir NumPy dizisidir. İçerisinde Matplotlib nesneleri bulunmaktadır
```

```
numpy.ndarray
```

```
type(ax[0][0])
```

```
matplotlib.axes._subplots.AxesSubplot
```

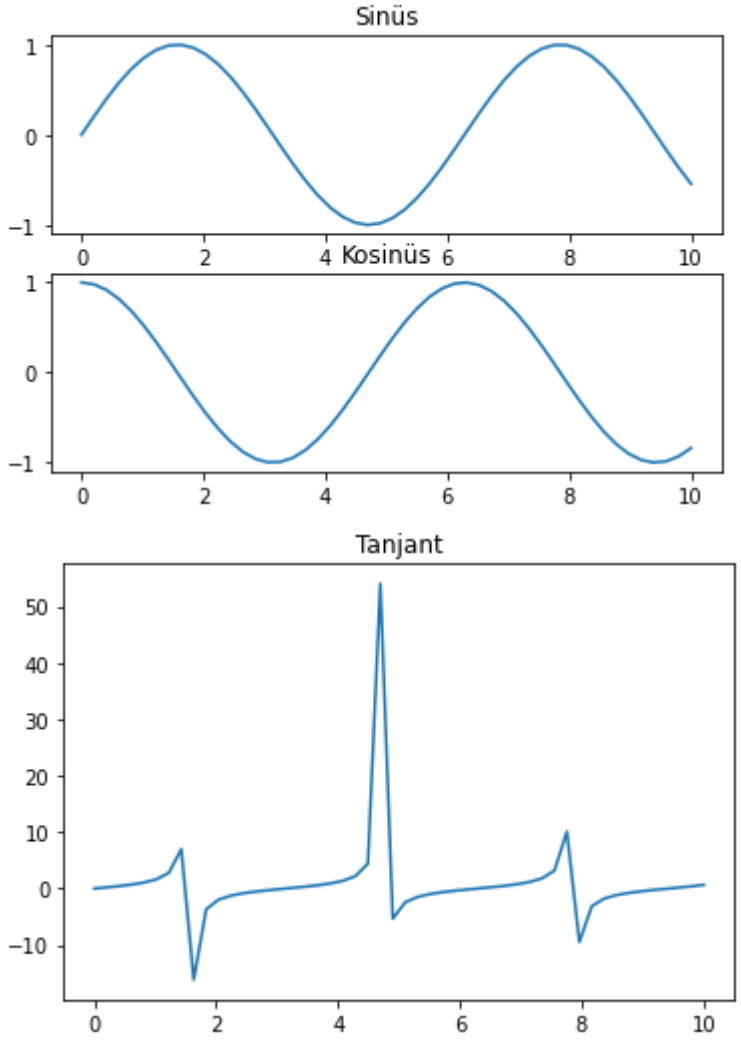
Aşağıdaki örnek, birden fazla figür oluşturma işlemini göstermektedir.

Bu örnek, iki farklı figure oluşturur ve her figure içinde bir veya daha fazla subplot yerleştirir. `add_subplot()` fonksiyonu, figure içindeki subplotların yerleştirme düzenini belirlemek için kullanılır.

Bu örnekte, ilk figure'da 2 satır ve 1 sütun olarak ayarlanmıştır ve ilk subplot 1. satırda, ikinci subplot 2. satırda yerleştirilmiştir. İkinci figure'da 1 satır ve 1 sütun olarak ayarlanmıştır ve tek bir subplot içermektedir.

`plt.show()` fonksiyonu, tüm figürleri görüntülemek için kullanılır.

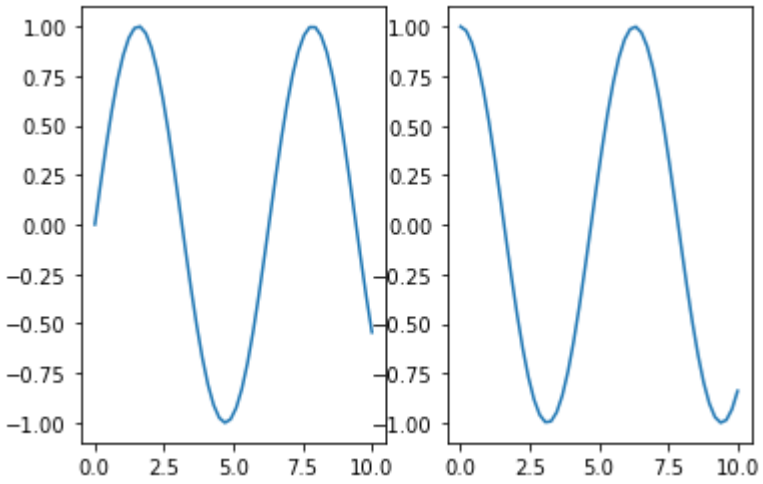
```
# 1. Figure oluşturma
fig1 = plt.figure()

# 2. İlk subplot oluşturma
ax1 = fig1.add_subplot(2, 1, 1) # 2 satır, 1 sütun, 1. subplot
x = np.linspace(0, 10)
y = np.sin(x)
ax1.plot(x, y)
ax1.set_title("Sinüs")

# 3. İkinci subplot oluşturma
ax2 = fig1.add_subplot(2, 1, 2) # 2 satır, 1 sütun, 2. subplot
y = np.cos(x)
ax2.plot(x, y)
ax2.set_title("Kosinüs")

# 4. İkinci figure oluşturma
fig2 = plt.figure()

# 5. Üçüncü subplot oluşturma
ax3 = fig2.add_subplot(1, 1, 1) # 1 satır, 1 sütun, 1. subplot
y = np.tan(x)
ax3.plot(x, y)
ax3.set_title("Tanjant")

# 6. Tüm figürleri gösterme
plt.show()
```



Başka bir örnekte MATLAB stili pyplot grafik çizdirmede birden fazla eksende grafik çizdirilmek istenirse aşağıdaki yapı kullanılabilir.

```
plt.subplot(121) # Bir satır iki sütunlu bir yapıda birinci grafiği çizdir.
plt.plot(x, np.sin(x))
plt.subplot(122) # İkinci grafik
plt.plot(x, np.cos(x))
```

```
[<matplotlib.lines.Line2D at 0x26795965640>]
```

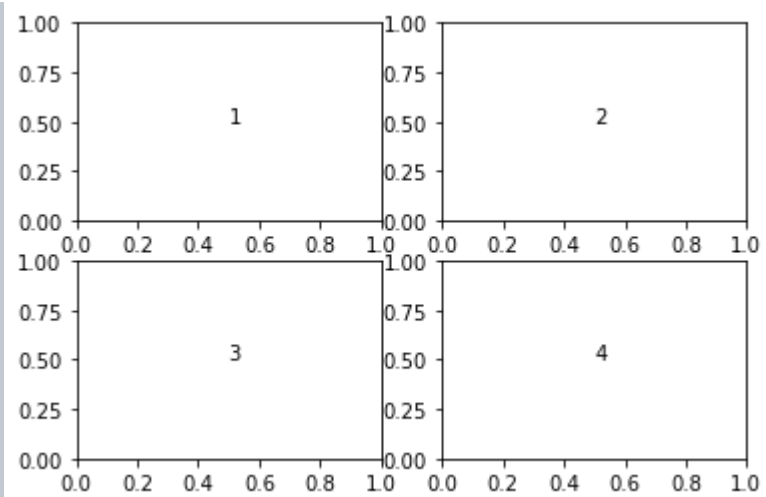


Izgaralar ile Subplot çizdirme

Yaygın kullanım için benzer ölçülerde eksenler ile çalışmak her ne kadar yeterli olsa da Matplotlib istenilen boyutlarda eksenler oluşturmak için imkan sunmaktadır. `plt.subplot()` fonksiyonu satır, sütun ve indeks değerlerinden oluşan 3 tam sayı değeri almaktaydı. Tam sayı değerleri eksenin kapsayacağı ızgara dilimi ile ilgili bilgiyi vermemektedir.

```
for i in range(1,5):
    print(i)
    plt.subplot(2,2,i) # 2 satır 2 sütun eksenlerin i. ekseni
    plt.text(0.5, 0.5, str(i))
```

```
1
2
3
4
```



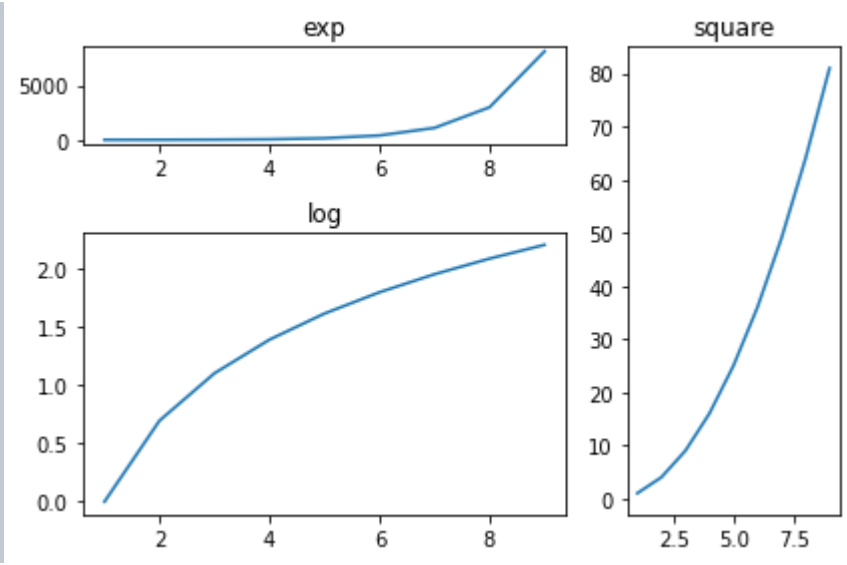
Birden fazla subplot çizdirmek için `plt.subplots()` fonksiyonu kullanılır. Bu durumda ızgara içerisindeki tüm konumlar kullanılabilir hale gelir. Yukarıdaki örneği ortak x ve y için yapmak istersek aşağıdaki yapıyı kullanabiliriz.

plt.subplot2grid() fonksiyonu

Izgaraların içerisine istenilen boyutlarda grafikler eklemek için subplot2grid() fonksiyonu kullanılabilir. Bu fonksiyon sırasıyla shape, location, rowspan, colspan parametrelerini alır.

- shape: satır ve sütun değerleri alır. Birinci girdi satır sayısını ikinci girdi sütun sayısını gösterir.
- loc: 2 tam sayı alır. Izgara içerisinde nereye yerleştirileceğini belirler.
- rowspan: Sağ tarafa doğru ne kadar genişleyeceğini belirtir.
- colspan: Aşağı doğru ne kadar genişleyeceğini belirler.

```
a1 = plt.subplot2grid((3,3),(0,0),colspan = 2) # 3x3 lük bir ızgarada 1. satır birinci
sütunda iki sütun git.
a2 = plt.subplot2grid((3,3),(0,2), rowspan = 3)
a3 = plt.subplot2grid((3,3),(1,0),rowspan = 2, colspan = 2)
x = np.arange(1,10)
a2.plot(x, x*x)
a2.set_title('square')
a1.plot(x, np.exp(x))
a1.set_title('exp')
a3.plot(x, np.log(x))
a3.set_title('log')
plt.tight_layout()
plt.show()
```



Grafiklere Farklı Elemanlar Ekleme

Grafikler içerisinde işaretleyici, metin, şekil gibi elemanlar eklenmek istenirse aşağıdaki yapılar kullanılabilir.

Metin Ekleme

Daha önceden göserilen xlabel(), ylabel(), title() gibi fonksiyonlar figür içerisine metin eklemekte kullanılmıştı. Metin eklemek için `text()` fonksiyonu veya `annotate()` fonksiyonu kullanılabilir.

`text()` fonksiyonu, belirtilen koordinatlara bir metin nesnesi ekler.

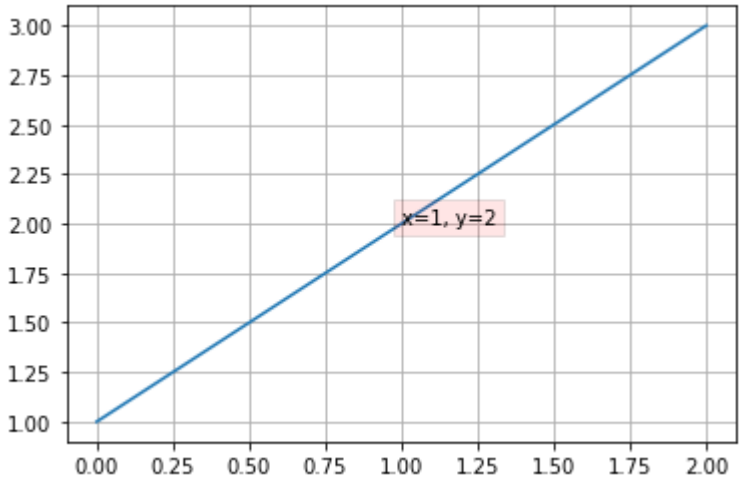
`text(x, y, s, **kwargs)`

Burada, x ve y koordinatları, metnin nerede başlayacağını belirler ve s argümanı, eklenecek olan metni içerir. İsteğe bağlı argümanlar arasında yazı tipi boyutu, renk, yazı tipi ve hizalama gibi metin özellikleri yer alabilir.

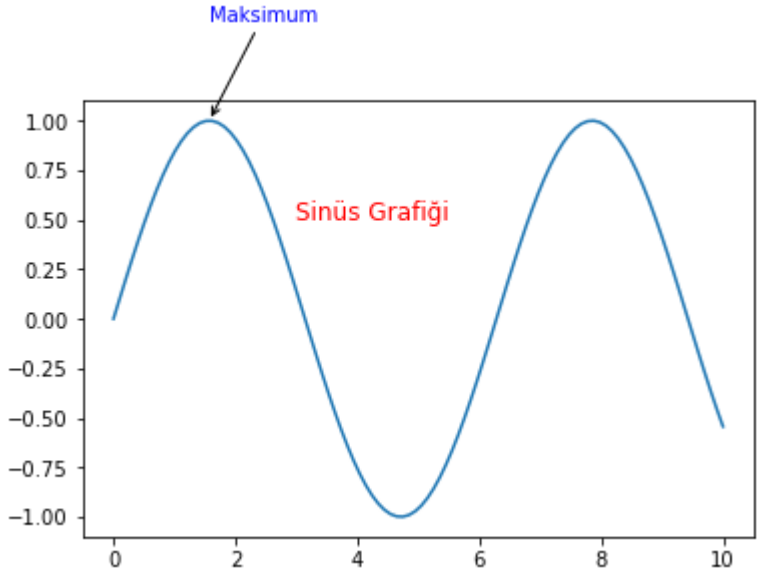
`annotate()` fonksiyonu ise text() fonksiyonuna benzerdir ancak belirtilen noktaya işaretçi (ok) eklemek için daha uygun bir seçenektir.

`annotate(s, xy, xytext, arrowprops=None, **kwargs)` Burada, s argümanı, eklenen metni içerir, xy argümanı, işaretçinin gösterileceği noktayı belirler ve xytext argümanı, metnin gösterileceği noktayı belirler. İsteğe bağlı arrowprops argümanı, işaretçi özelliklerini belirlemek için kullanılabilir.

```
plt.plot([1,2,3])
plt.text(1,2,"x=1, y=2", bbox={'facecolor': 'red', 'alpha': 0.1})
plt.grid() # Konumları daha rahat görebilmek için
```




```
# Örnek veri
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Bir figür oluşturma ve eksenleri ayarlama
fig, ax = plt.subplots()
ax.plot(x, y)

# Metin eklemek için text() fonksiyonunu kullanma
ax.text(3, 0.5, "Sinüs Grafiği", fontsize=12, color="red")

# İşaretçi ile metin eklemek için annotate() fonksiyonunu kullanma
ax.annotate("Maksimum", xy=(np.pi/2, 1), xytext=(np.pi/2, 1.5),
            fontsize=10, color="blue",
            arrowprops=dict(facecolor="blue", arrowstyle="->"))

# Grafik gösterme
plt.show()
```



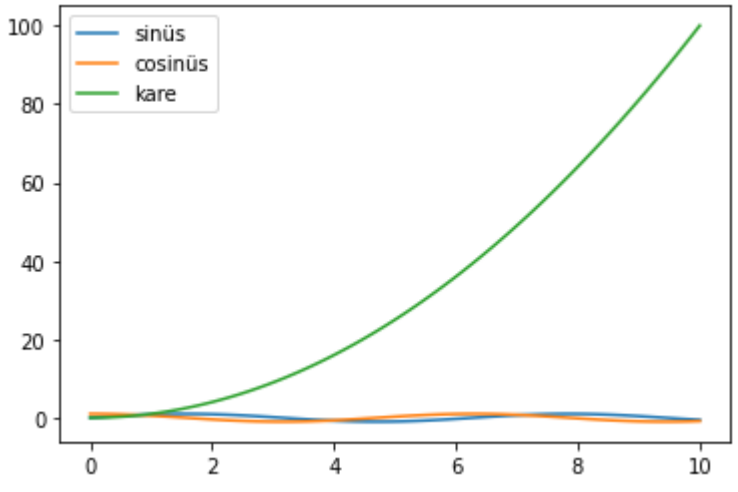
Bu örnek, bir sinüs grafiğine metin eklemeyi göstermektedir. `text()` fonksiyonu, Sinüs Grafiği metnini belirtilen koordinatlarda eklerken, annotate() fonksiyonu, maksimum noktaya işaretçi ile birlikte bir metin ekler.

Legend() fonksiyonu

Daha önce kullanılan bu fonksiyon ile daha önceden etiket verilmemiş olan grafiklere sırası ile etiket verilebilir. Konumu ile ilgili loc parametresine (best, upper right, upper left, right, center, lower center) gibi değerler verilebilir.

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.plot(x, x**2)
plt.legend(['sinüs', 'cosinüs', 'kare'], loc = 'upper left')
```

```
<matplotlib.legend.Legend at 0x2679545c220>
```



Bazı Sihirli Fonksiyonlar

Jupyter notebooklarda kullanılabilecek bir takım sihirli fonksiyonlar(magic functions) ile kolay bir kullanım sağlanabilir. Matplotlib kütüphanesinde kullanılabilecek sihirli fonksiyonlardan bazıları şu şekildedir.

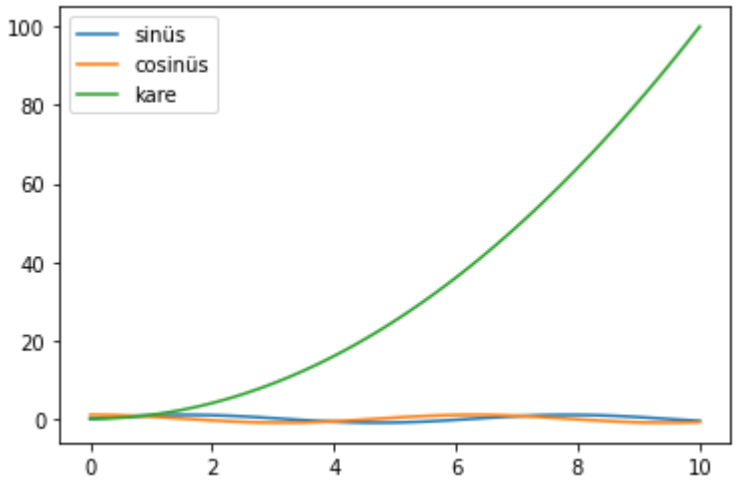
-
-
-

```
%matplotlib notebook
```

```
%matplotlib inline
```

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.plot(x, x**2)
plt.legend(['sinüs', 'cosinüs', 'kare'], loc = 'upper left')
```

```
<matplotlib.legend.Legend at 0x267959f4910>
```




```
%matplotlib inline
```

```
%pwd
```

```
'C:\\Users\\Erden\\Desktop\\veri-analizi-python\\veri-analizi-python'
```

```
# Diğer sihirli fonksiyonları listelemek için
%lsmagic
```

```
Available line magics:
%alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cd
%clear  %cls  %colors  %conda  %config  %connect_info  %copy  %ddir  %debug  %dhist
%dirs  %doctest_mode  %echo  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts
%ldir  %less  %load  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop
%ls  %lsmagic  %macro  %magic  %matplotlib  %mkdir  %more  %notebook  %page  %pastebin
%pdb  %pdef  %pdoc  %pfile  %pinfo  %pinfo2  %pip  %popd  %pprint  %precision  %prun
%psearch  %psource  %pushd  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall
%rehashx  %reload_ext  %ren  %rep  %rerun  %reset  %reset_selective  %rmdir  %run  %save
%sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %unalias  %unload_ext  %who
%who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%cmd  %%debug  %%file  %%html  %%javascript  %%js
%%latex  %%markdown  %%perl  %%prun  %%pypy  %%python  %%python2  %%python3  %%ruby
%%script  %%sh  %%svg  %%sx  %%system  %%time  %%timeit  %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

```
# Örneğin whos sihirli fonksiyonu kullanılan değişkenleri listeler
%whos
```

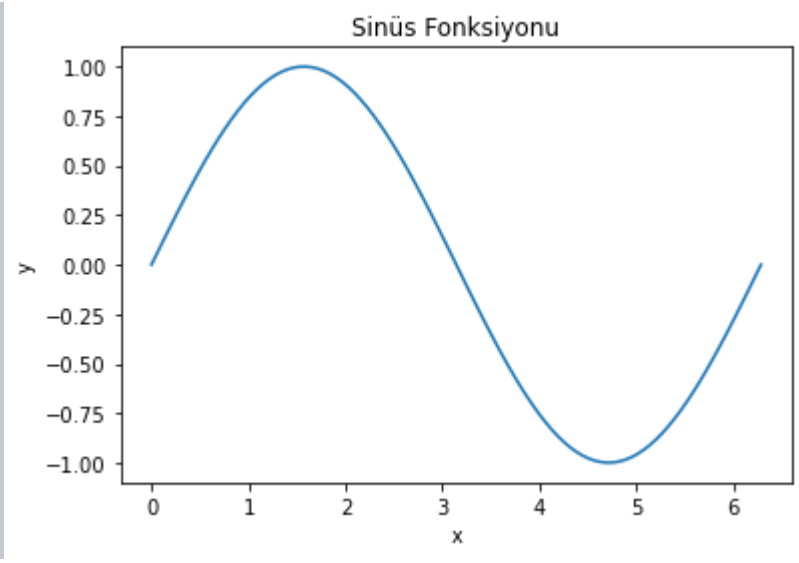
```
Variable   Type                       Data/Info
-------------------------------------------
a1         AxesSubplot                AxesSubplot(0.100231,0.738194;0.559606x0.16875)
a2         AxesSubplot                AxesSubplot(0.730556,0.0965278;0.244444x0.810417)
a3         AxesSubplot                AxesSubplot(0.100231,0.0965278;0.559606x0.489583)
ax         AxesSubplot                AxesSubplot(0.125,0.125;0.775x0.755)
ax1        AxesSubplot                AxesSubplot(0.125,0.536818;0.775x0.343182)
ax2        AxesSubplot                AxesSubplot(0.125,0.125;0.775x0.343182)
ax3        AxesSubplot                AxesSubplot(0.125,0.125;0.775x0.755)
axs        ndarray                    3x2: 6 elems, type `object`, 48 bytes
cbar       Colorbar                   <matplotlib.colorbar.Colo<...>ct at
0x000002679518DD60>
cmap       LinearSegmentedColormap    <matplotlib.colors.Linear<...>ct at
0x00000267930B84C0>
data       ndarray                    5x12: 60 elems, type `int32`, 240 bytes
fig        Figure                     Figure(432x288)
fig1       Figure                     Figure(432x288)
fig2       Figure                     Figure(432x288)
i          int                        4
norm       Normalize                  <matplotlib.colors.Normal<...>ct at
0x000002679511A880>
np         module                     <module 'numpy' from 'D:\
<...>ges\\numpy\\__init__.py'>
pd         module                     <module 'pandas' from 'D:
<...>es\\pandas\\__init__.py'>
plt        module                     <module
'matplotlib.pyplo<...>\\matplotlib\\pyplot.py'>
x          ndarray                    100: 100 elems, type `float64`, 800 bytes
y          ndarray                    100: 100 elems, type `float64`, 800 bytes
```

Grafiklerin Kaydedilmesi

Çizimi bir pdf dosyasına ve bir jpeg resim dosyasına aktarmak oldukça kolaydır. Matplotlib'in farklı formatlar için geliştirilmiş fonksiyonları vardır.

`savefig()` fonksiyonu kullanılarak grafik farklı dosya formatlarında kaydedilebilir. Bu fonksiyonun ilk parametresi dosyanın adı ve yoludur, ikinci parametresi ise dosya formatıdır.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Sinüs Fonksiyonu')

plt.savefig('sinus_grafik.png', format='png')
```



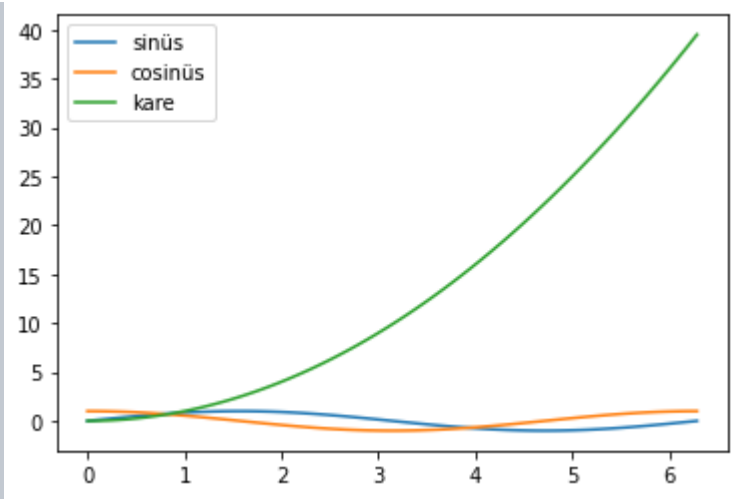
`savefig()` fonksiyonu çağrıldıktan sonra, oluşturulan grafiği kaydetmek için belirtilen dosya adı ve yoluna sahip bir dosya oluşturulur. İkinci parametre olan format parametresi, kaydedilen dosyanın hangi formatta kaydedileceğini belirtir. Bu parametre isteğe bağlıdır ve varsayılan olarak PNG formatında kaydeder.

Ayrıca, dpi parametresi ile kaydedilen dosyanın çözünürlüğü de ayarlanabilir.

- dpi=None: Görüntünün çözünürlüğü, örneğin 300 dpi(inç başına düşen nokta sayısı)
- transparent=False: True değeri verildiğinde grafiğin arka planı transparan hale getirilir.
- bbox_inches=None, grafik çevresindeki sınırlar ile ilgilidir. bbox_inches='tight' genellikle ideal sonuçlar verir.

PNG formatı internet kullanımları ve çizim grafikleri için daha uygundur. JPG formatı ise daha sıkıştırılmış bir resim formatıdır. Bu nedenle daha küçük boyutlarda görüntülerin kaydedilmesini sağlar. Eğer daha küçük boyutlu resimler ile çalışılmak isteniyorsa JPG formatı tercih edilebilir. SVG formatı ise vektörel görüntülerde kullanılan ve diğer formatlara göre boyutu yüksek bir seçimdir.

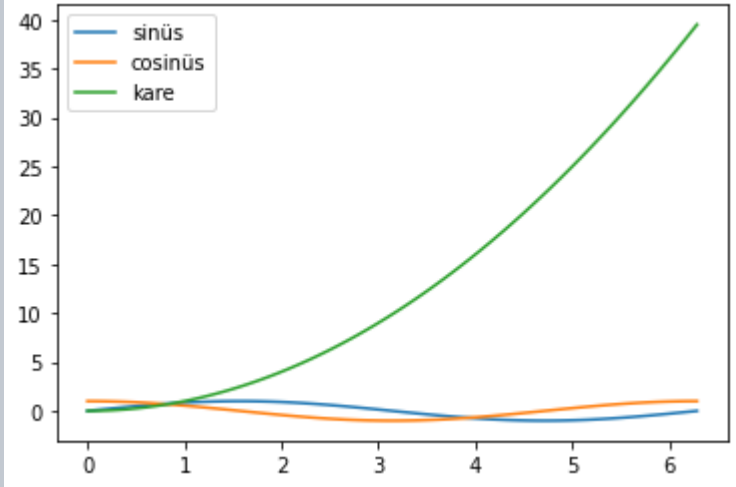
```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.plot(x, x**2)
plt.legend(['sinüs', 'cosinüs', 'kare'], loc='upper left')
plt.savefig('sin_cos_grafigi.png', dpi=300, bbox_inches='tight')
```



Bu kod bloğunda, `dpi` parametresi 300 olarak ayarlandı ve `bbox_inches` parametresi kullanılarak grafiğin etrafındaki beyaz boşlukların otomatik olarak kırpması sağlandı.

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.plot(x, x**2)
plt.legend(['sinüs', 'cosinüs', 'kare'], loc = 'upper left')
plt.savefig('sin_cos_grafigi_transparent.png', dpi=300, transparent=True, figsize=(4, 4))
```

```
C:\Users\Erden\AppData\Local\Temp/ipykernel_15712/1189373220.py:5:
MatplotlibDeprecationWarning: savefig() got unexpected keyword argument "figsize" which
is no longer supported as of 3.3 and will become an error two minor releases later
  plt.savefig('sin_cos_grafigi_transparent.png', dpi=300, transparent=True, figsize=(4,
4))
```



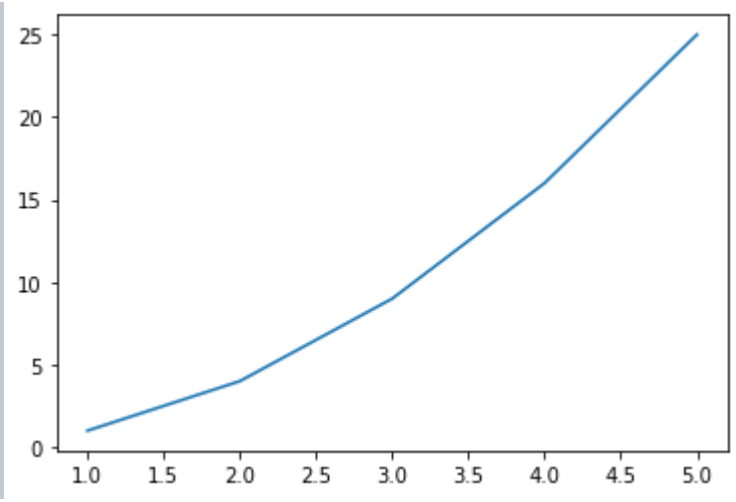
`dpi` parametresi çözünürlüğü ayarlar, `bbox_inches` parametresi grafiğin kenar boşluklarını kırpar ve `figsize` parametresi grafiğin boyutunu ayarlar.

.svg ve .pdf formatları

PDF (Portable Document Format), Adobe Systems tarafından geliştirilmiş bir belge formatıdır. PDF formatındaki belgeler, farklı işletim sistemleri ve cihazlar arasında kolayca paylaşılabilir ve görüntülenebilir. Matplotlib, PDF formatında grafikleri kaydetmek için `savefig()` fonksiyonunu kullanır.

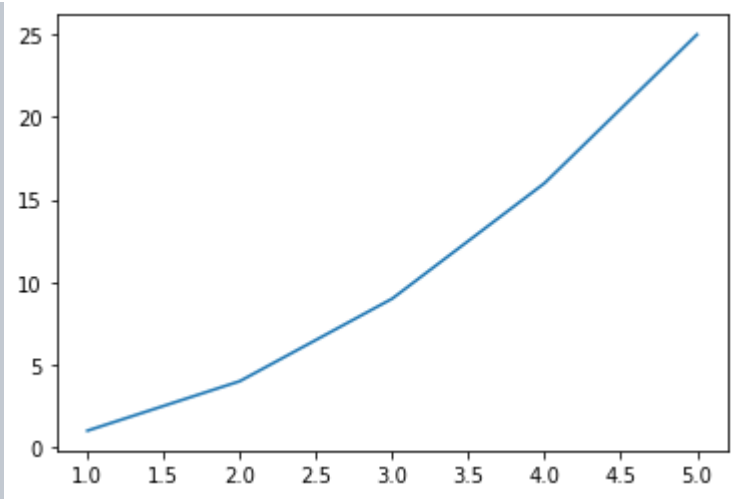
Örneğin, aşağıdaki kod parçası, bir çizgi grafiğini grafik.pdf dosyasına kaydeder:

```
import matplotlib.pyplot as plt

# Verileri oluşturma
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Çizgi grafiği oluşturma
plt.plot(x, y)

# Grafiği PDF formatında kaydetme
plt.savefig('grafik.pdf')
```



SVG (Scalable Vector Graphics), XML tabanlı bir vektör grafik formatıdır. SVG formatı, vektör tabanlı grafikleri kaydetmek için kullanılır ve görüntülerin boyutunu, renklerini ve diğer özelliklerini korur.

```
import matplotlib.pyplot as plt

# Verileri oluşturma
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Çizgi grafiği oluşturma
plt.plot(x, y)

# Grafiği SVG formatında kaydetme
plt.savefig('grafik.svg')
```



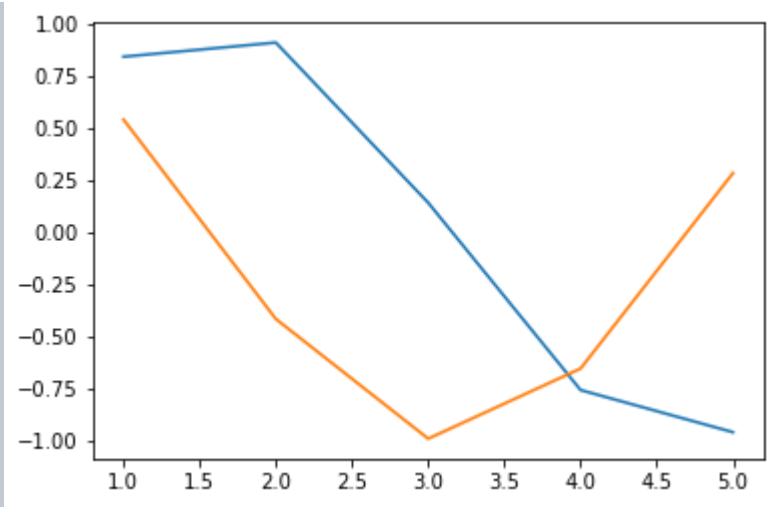
Bu örneklerde `savefig()` fonksiyonu, çizgi grafiğini belirtilen dosya adı ve formatı ile kaydeder. format parametresi kullanılarak kaydedilecek dosya formatı belirtilir. Örneğin, `format='pdf'` şeklinde belirtilerek grafik PDF formatında kaydedilebilir.

Renk kodları ile hızlı bir şekilde grafikteki renkler değiştirilebilir. Renk kodları aşağıda paylaşılmıştır.

- r(red)
- g(green)
- b(blue)
- c(cyan)
- k(black)
- w(white)

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.plot(x, x**2)
plt.legend(['sinüs', 'cosinüs', 'kare'], loc = 'upper left')
plt.savefig('sin_cos_grafigi_transparent.png', dpi=300, transparent=True, facecolor='b')
# ön renk mavi
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15712/627628611.py in <module>
      1 plt.plot(x, np.sin(x))
      2 plt.plot(x, np.cos(x))
----> 3 plt.plot(x, x**2)
      4 plt.legend(['sinüs', 'cosinüs', 'kare'], loc = 'upper left')
      5 plt.savefig('sin_cos_grafigi_transparent.png', dpi=300, transparent=True,
facecolor='b') # ön renk mavi

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```



Grafik Çeşitleri ve Stiller

Matplotlib kütüphanesi, birçok farklı grafik türü için çeşitli çizim fonksiyonları içerir. Bazı popüler grafik türleri şunlardır:

1. Çizgi grafikleri (Line plots): Değişkenler arasındaki ilişkiyi göstermek için kullanılır.
2. Dağılım grafikleri (Scatter plots): İki değişken arasındaki ilişkiyi göstermek için kullanılır.
3. Histogramlar: Değişkenlerin dağılımını göstermek için kullanılır.
4. Kutu grafikleri (Box plots): Değişkenlerin istatistiksel özelliklerini göstermek için kullanılır.
5. Alan grafikleri (Area plots): Değişkenlerin zamana bağlı olarak değişimini göstermek için kullanılır.
6. Pasta grafikleri (Pie charts): Kategorik verilerin oranlarını göstermek için kullanılır.
7. Çubuk grafikleri (Bar charts): Kategorik verilerin karşılaştırılması için kullanılır.

Matplotlib'de birkaç grafik çeşidi özel tanımlı fonksiyonlar vardır. Bu fonksiyonlar, çeşitli grafik türlerini veya özellikleri oluşturmak için birkaç satır kodla çağrılabilen önceden tanımlanmış işlevlerdir. Bu fonksiyonlar, hızlı bir şekilde grafikler oluşturmak için yararlıdır.

Bu fonksiyonlardan bazıları şunlardır:

- `subplots()`: Bu fonksiyon, birden fazla grafik içeren bir figür oluşturmak için kullanılır. Belirtilen sayıda alt grafik oluşturur ve her alt grafik, figürün bir parçasıdır. Örneğin, fig, axs = plt.subplots(2, 3) kodu, 2 satır ve 3 sütundan oluşan bir alt grafik düzeni oluşturur.
- `scatter()`: Bu fonksiyon, nokta bulutu grafikleri oluşturmak için kullanılır. Her bir nokta, verilerdeki iki boyutlu bir noktayı temsil eder. Örneğin, plt.scatter(x, y) kodu, x ve y verilerine dayalı bir nokta bulutu grafik oluşturur.

- `bar()`: Bu fonksiyon, sütun grafikleri oluşturmak için kullanılır. Her bir sütun, verilerdeki bir değeri temsil eder. Örneğin, plt.bar(x, y) kodu, x değerlerine göre yüksekliği belirtilen sütunlar oluşturur.
- `hist()`: Bu fonksiyon, histogramlar oluşturmak için kullanılır. Histogram, verilerin dağılımını göstermek için kullanılan bir grafik türüdür. Örneğin, plt.hist(x, bins=10) kodu, x verilerine dayalı 10 bölümlü bir histogram oluşturur.
- `pie()`: Bu fonksiyon, pasta grafikleri oluşturmak için kullanılır. Her bir dilim, verilerdeki bir değeri temsil eder. Örneğin, plt.pie(x, labels=labels) kodu, x değerlerine göre etiketlenmiş bir pasta grafiği oluşturur.

Çizgi grafikleri

Çizgi grafikleri veri setlerindeki değişimleri zaman içinde izlemek ve trendleri belirlemek için kullanılır. İşletme verilerinde, finansal verilerde ve ekonomik verilerde sıklıkla kullanılırlar. Ayrıca, çok sayıda veri noktası arasındaki ilişkiyi göstermek için de yararlıdırlar.

Çizgi grafiklerinin avantajları şunlardır:

- Verilerin zaman içindeki değişimini net bir şekilde gösterirler.
- Birden fazla veri seti arasındaki ilişkiyi kolayca anlamamızı sağlar.
- Verilerin trendlerini ve değişimlerini hızlı bir şekilde anlayabiliriz.
- Basit ve anlaşılır bir grafik türüdür.
- Bu nedenlerle, işletmelerde, finansal kurumlarda ve ekonomi alanında sıklıkla kullanılan bir grafik türüdür.

Tablo verilerinin belirli bir zaman aralığındaki değişimini görselleştirmek için çizgi grafikleri sıkça kullanılır. Örnek olarak, aylık gelirleri gösteren bir veri seti üzerinden bir çizgi grafiği çizelim.

```
aylar = ['Ocak', 'Şubat', 'Mart', 'Nisan', 'Mayıs', 'Haziran']
gelirler = [20000, 25000, 30000, 28000, 32000, 35000]
plt.plot(aylar, gelirler, marker='o')
plt.title('Aylık Gelirler')
plt.xlabel('Aylar')
plt.ylabel('Gelirler')
plt.ylim(0, 40000)
```

Bu kodları çalıştırdığımızda, aylık gelirlerimizi gösteren bir çizgi grafiği elde ederiz. Grafikte her ay için bir nokta ve bu noktaları birleştiren bir çizgi görüntülenir. Ayrıca grafiğimizde başlık, etiketler ve y-ekseni sınırları da vardır.

Dağılım grafikleri

Dağılım grafikleri, verilerin dağılımını görselleştirmek için kullanılır. Bu grafik türü, verilerin merkezi eğilimlerini, dağılımını ve aykırı değerlerini göstermek için kullanışlıdır. Dağılım grafikleri, veri setlerindeki değişkenlikleri ve dağılımları anlamak için özellikle yararlıdır.

Histogram grafikleri

Histogramlar, veri dağılımını görselleştirmek için kullanılan bir grafik türüdür. Özellikle büyük veri kümelerinde kullanılmaktadır. Histogramlar, verilerin belirli bir aralıktaki sıklığını ve yoğunluğunu gösterir.

Histogramlar, birçok alanda kullanılabilir. Örneğin, finansal verileri incelemek için, hisse senedi fiyatlarının belirli bir aralıktaki dağılımını analiz etmek için kullanılabilirler. Ayrıca, tıp alanında, hastaların belirli bir özelliklerinin (örneğin, yaş veya kan basıncı) dağılımını incelemek için de kullanılabilirler.

Histogramlar, veri dağılımını anlamak için oldukça yararlıdır. Verilerin nasıl dağıldığı, hangi aralıklarda yoğunlaştığı veya dağıldığı, hangi değerlerin daha yaygın olduğu gibi bilgileri görselleştirerek anlamak kolaylaşır. Ayrıca, histogramlar, veri setindeki aykırı değerleri de tespit etmek için kullanılabilir.

Histogramların bir diğer avantajı, veri kümesinin şekil ve boyutuna göre özelleştirilebilmesidir. Böylece, verilerin daha iyi anlaşılmasını sağlayacak bir histogram oluşturulabilir.

```
kadin_yaslar = np.random.randint(low=1,high=75,size=100)
plt.hist(kadin_yaslar,label='kadın yaşları')
plt.legend()
```

Bu örnekte histogram, `plt.hist()` fonksiyonu ile çizildi. Histogramda kullanılan parametreler arasında, `density=True` seçeneği histogramın yoğunluk çizimini gösterirken, facecolor parametresi histogramın dolgu rengini belirler.

```
plt.hist(kadin_yaslar,label='kadın yaşları', alpha=0.5)
plt.hist(erkek_yaslar,label='erkek yaşları', alpha=0.5)
plt.legend()
```

```
plt.hist([erkek_yaslar, kadin_yaslar], label=['erkek', 'kadin'])
plt.legend()
```

Aynı şekilde histogram grafikleri olasılık yoğunluk fonksiyonlarının çizdirilmesinde de kullanılabilir. Örneğin normal dağılıma uyan bir veri dizisinin grafiği aşağıdaki çizdirilebilir.

```
np.random.seed(12345)
x_normal = np.random.normal(size=100) # normal dağılıma uyan 100 veri
plt.hist(x_normal, density=True, bins=30)  # `density=False` sayıları alacaktı. Oranları
aldık
plt.ylabel('Olasılıklar')
plt.xlabel('Veriler');
```

```
np.random.seed(12345)
x_normal = np.random.normal(size=1000000) # normal dağılıma uyan 10000 veri
plt.hist(x_normal, density=True, bins=30)  # `density=False` sayıları alacaktı. Oranları
aldık
plt.ylabel('Olasılıklar')
plt.xlabel('Veriler');
```

Kutu grafikleri

Kutu grafiği, verilerin dağılımını ve değişkenliğini görselleştirmek için kullanılan bir grafik türüdür. En yaygın kullanımı, farklı kategoriler veya gruplar arasındaki sayısal verilerin karşılaştırılmasıdır. Ayrıca, verilerin aykırı (outlier) değerlerini belirlemek ve ortalamadan sapmaları görselleştirmek için de kullanılabilir.

Kutu grafikleri, aşağıdaki avantajlara sahiptir:

- Görselleştirme açısından kolaydır ve verilerin dağılımını hızlı bir şekilde anlamamızı sağlar.
- Aykırı değerleri (outlier) belirlemede ve veriler arasındaki değişkenliği göstermede etkilidir.
- Farklı gruplar veya kategoriler arasındaki sayısal verilerin karşılaştırılması için idealdir.

Kullanım örneği olarak, bir şirketin farklı bölümlerinin aylık satış verilerini karşılaştırmak istediğimizi düşünelim. Kutu grafiği, farklı bölümlerin satış verilerinin ortalamasını, medyanını, minimum ve maksimum değerlerini, ayrık değerlerini ve verilerin dağılımını göstererek kolayca karşılaştırmamıza olanak sağlayacaktır.

```
# Verileri oluşturma
bolumler = ['Finans', 'Satış', 'Pazarlama', 'Muhasebe']
veriler = [[25000, 35000, 40000, 45000, 50000],
           [30000, 35000, 40000, 45000, 55000],
           [20000, 25000, 35000, 40000, 45000],
           [15000, 20000, 25000, 30000, 35000]]

# Kutu grafiği oluşturma
plt.boxplot(veriler)
plt.xticks(range(len(bolumler)), bolumler)
plt.xlabel('Bölümler')
plt.ylabel('Aylık Satış')
plt.title('Bölümler Arasındaki Aylık Satış Farklılıkları')
plt.show()
```

Bu kod, farklı bölümlerin aylık satış verilerinin kutu grafiğini oluşturacak ve her bir bölüm için ayrı bir kutu oluşturarak verilerin karşılaştırılmasına olanak sağlayacaktır.

Alan grafikleri

Alan grafiği, verilerin belirli bir aralıktaki değişimlerini göstermek için kullanılan bir grafik türüdür. Genellikle zamana bağlı değişimlerin gösterilmesinde tercih edilir ve verilerin toplamını veya yüzdesini de gösterir.

Alan grafiklerinin avantajları şunlardır:

- Zamanla değişen verilerin görselleştirilmesi için idealdir.
- Verilerin toplamını veya yüzdesini göstererek, toplamın ne kadarını hangi kategoride harcadığımızı anlamamızı sağlar.
- Birden fazla veri seti arasındaki ilişkiyi göstermek için kullanılabilir.
- Verilerin dağılımını gösterirken, grafikteki renkler ve gölgelendirme yardımıyla görsel olarak daha çekici hale getirilebilir.

Örnek kullanımı aşağıdaki gibi olabilir:

```
# Verileri oluşturma
yil = [2016, 2017, 2018, 2019, 2020]
gelir = [10000, 15000, 20000, 25000, 30000]
gider = [5000, 8000, 10000, 15000, 20000]

# Alan grafiği oluşturma
plt.stackplot(yil, gelir, gider, labels=['Gelir', 'Gider'])
plt.legend(loc='upper left')
plt.title('Gelir ve Giderler')
plt.xlabel('Yıl')
plt.ylabel('Tutar')
plt.show()
```

Pasta grafikleri

Pasta grafikleri, bir veri kümesindeki oranları görselleştirmek için kullanılan grafik türleridir. Pasta grafikleri, her bir veri kategorisi için yüzde veya mutlak sayısal değerler kullanarak, her bir kategori arasındaki oranları kolayca anlamamızı sağlar.

Pasta grafikleri ayrıca şunlara da yardımcı olabilir:

- Bir veri kümesindeki dağılımı anlamak
- Toplamın ne kadarını her kategorinin temsil ettiğini anlamak
- Farklı veri kategorileri arasındaki oranları karşılaştırmak

Örneğin, bir şirketin müşterilerinin demografik dağılımını göstermek istediğinizi varsayalım. Pasta grafikleri bu verileri kolayca görselleştirmenize yardımcı olabilir. Ayrıca, bir ürün veya hizmetin farklı pazar segmentleri arasındaki payını göstermek için de

kullanılabilir.

İşte bir pasta grafik örneği:

```
# Verileri oluşturma
kategoriler = ['Erkek', 'Kadın', 'Diğer']
oranlar = [60, 35, 5]

# Pasta grafik oluşturma
plt.pie(oranlar, labels=kategoriler, autopct='%1.1f%%')

# Grafik özellikleri
plt.title('Müşteri Demografisi')
plt.axis('equal')

# Grafiği gösterme
plt.show()
```

Bu örnek, bir şirketin müşteri tabanının demografik dağılımını göstermek için bir pasta grafik kullanır. “Erkek”, “Kadın” ve “Diğer” kategorileri için oranları gösterir ve her bir dilimde yüzde değerlerini görüntüler.

Çubuk grafikleri

Çubuk grafikleri yaygın bir kullanıma sahiptir. Histogram grafiklerinden farklı yanı x aksisindeki değerlerin kategorik değil nümerik değerler almasıdır. bar() fonksiyonu ile çizim yapılabilir. Diğer grafiklerdeki benzer parametreler burada da geçerlidir. Örneğin align parametresi ile verinin çubuğun ortasından geçmesi istendiği durumda kullanılabilir. Ya da orientation parametresi ile çubuklar dikey eksende mi olsun yatay eksende mi bununla ilgili ayarlama için kullanılabilir. Genellikle ölçüm değerinin nasıl değiştiği görülmek için kullanılır. x değerlerine karşılık gelen y değerlerine kadar bir çubuk çizilir.

```
x1_koordinatlari = np.random.randint(10, size=20)
y1_koordinatlari = np.random.randint(5,size=20)

x2_koordinatlari = np.random.randint(10, size=20)
y2_koordinatlari = np.random.randint(5,size=20)

plt.bar(x1_koordinatlari,y1_koordinatlari,label='grup 1')
```

Bunun yanı sıra çubuk grafikleri kategorik veriler için de kullanılabilir. Bunun için;

```
x_bar = np.random.randint(5,size=5)
plt.barh(x_bar, np.arange(5))
plt.xticks(np.arange(5),['bir','iki','üç','dört','beş']) # kategoriler ayrıca
verilebilir.
```

Diğer grafik çeşitleri

- pie pasta grafikleri

- scatter dağılım grafikleri
- boxplot kutu grafikleri

Kelime bulutları

Kelime bulutu, bir metin belgesinde geçen kelimelerin frekans dağılımını görselleştirmek için kullanılır. Genellikle, kelime bulutu oluşturma işlemi şu adımları içerir:

1. Metnin yüklenmesi ve işlenmesi: Kelime bulutu oluşturmak için öncelikle metin belgesinin yüklenmesi ve işlenmesi gerekir. Bu işlem, metnin önceden belirlenmiş bir dil işleme adımından geçirilmesi ile gerçekleştirilir. Bu adım, kelime ayıklama, özel karakterlerin kaldırılması ve ayrıştırma işlemlerini içerebilir.
2. Kelime frekanslarının hesaplanması: Metnin işlenmesinden sonra, belgedeki her kelimenin frekansı hesaplanır. Bu, kelimenin belgedeki toplam sayısıdır.
3. Kelime bulutu oluşturma: Hesaplanan kelime frekansları, kelime bulutu oluşturmak için kullanılır. Kelimeler, büyükten küçüğe doğru sıralanır ve en sık kullanılan kelimeler daha büyük yazı tipiyle gösterilir.
4. Kelime bulutları, özellikle sosyal medya analizi, web sayfası içeriği analizi ve pazarlama kampanyalarında kullanılan bir veri görselleştirme aracıdır.

3D grafikler

Matplotlib kütüphanesi, 3 boyutlu verileri görselleştirmek için de kullanılabilir. Bu işlem için `mplot3d` alt paketi kullanılır. Bu alt paket sayesinde, 3 boyutlu grafikler oluşturmak için gerekli araçlar sağlanır.

Öncelikle, `mplot3d` paketini kullanarak 3 boyutlu grafikler için gerekli araçları içe aktaralım:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = [1,2,3,4,5]
y = [2,3,4,5,6]
z = [0,1,2,3,4]
ax.scatter(x, y, z)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```

Bir başka örnek verelim. Burada `add_subplot()` fonksiyonu kullanılarak, 1 satır ve 1 sütundan oluşan bir alt çizim alanı oluşturulur ve projection parametresine “3d” değeri verilerek, 3 boyutlu bir alt çizim alanı oluşturulur.

3 boyutlu bir grafik çizdirmek için, `plot()` fonksiyonunun yerine `plot_wireframe()` fonksiyonu kullanılabilir. Bu fonksiyon, bir kablosuz çerçeve (wireframe) çizgisi şeklinde verileri görselleştirir.

Örneğin, aşağıdaki kod bloğunda, `plot_wireframe()` fonksiyonu kullanılarak bir kürenin yüzeyi çizdirilir:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d")

u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)

x = 10 * np.outer(np.cos(u), np.sin(v))
y = 10 * np.outer(np.sin(u), np.sin(v))
z = 10 * np.outer(np.ones(np.size(u)), np.cos(v))

ax.plot_wireframe(x, y, z, color="b")

plt.show()
```

Bu örnekte, `np.outer()` fonksiyonu kullanarak, kürenin yüzeyine ait x, y ve z koordinatları hesaplanır ve `plot_wireframe()` fonksiyonu kullanılarak bu koordinatlar kullanılarak bir kablosuz çerçeve çizdirilir. Son olarak, `show()` fonksiyonu kullanılarak grafik ekrana çizdirilir.

Bu şekilde, mplot3d paketi ile 3 boyutlu grafikler oluşturabilirsiniz.

Stil dosyaları

Matplotlib kütüphanesinde bir dokümanda düzenli bir şekilde renk ve görsel seçenekleri kullanmak için stiller kullanılmaktadır. Kütüphane içerisindeki stil dosyaları `plt.style.available` komutu ile gösterilebilir. Tercih edilen stil `plt.style.use()` komutu ile seçilebilir. Bir dokümanda çizilen grafiklerin renkleri, yazı fontları, font büyüklükleri, ızgara şekilleri gibi tasarım özelliklerinin benzer şekilde olması beklenir.

Diğer stiller ile ilgili bilgi şu adresten alınabilir.

```
plt.style.available
```

```
plt.style.use('seaborn')
```

```
%matplotlib inline
```

```
np.random.seed(12345)
x_normal = np.random.normal(size=10000) # normal dağılıma uyan 10000 veri
plt.hist(x_normal, density=True, bins=30)  # `density=False` sayıları alacaktı. Oranları
aldık
plt.ylabel('Olasılıklar')
plt.xlabel('Veriler')
```


```
np.random.seed(12345)
x_normal = np.random.normal(size=10000) # normal dağılıma uyan 10000 veri
plt.hist(x_normal, density=True, bins=30)  # `density=False` sayıları alacaktı. Oranları
aldık
plt.ylabel('Olasılıklar')
plt.xlabel('Veriler')
```


```
# varsayılanlar belirlemek
plt.rcParams['lines.linewidth']=2.0 # tüm çizgiler bu çizgi genişliğine sahip olur
```

Veri görselleştirmede ipuçları

Matplotlib'i veri görselleştirme için kullanmak, verilerinizdeki bilgileri ve desenleri etkili bir şekilde iletmek için güçlü bir araç olabilir. Ancak, herhangi bir araç gibi, en iyi sonucu elde etmek için doğru kullanmak önemlidir. Matplotlib kullanırken aklınızda bulundurmanız gereken bazı en iyi uygulamalar şunlardır:

1. Basit tutun: Grafiklerinizi fazla sayıda öğe veya gereksiz ayrıntı ile karmaşık hale getirmekten kaçının. Mesajınızı iletmeye yardımcı olan temel bilgilerle sınırlı kalın.
2. Doğru grafik türünü seçin: Farklı grafik türleri, farklı veri türleri ve mesajlar için daha uygun olabilir. Örneğin, bir çizgi grafiği zaman içindeki eğilimleri göstermek için daha iyidir, bir bar grafiği ise kategorileri karşılaştırmak için daha iyidir.
3. Renkleri etkili kullanın: Renk, önemli bilgileri vurgulamak için güçlü bir araç olabilir, ancak ölçülü kullanılmalıdır. Kolayca ayırt edilebilen renkler seçin ve çok fazla farklı renk kullanmaktan kaçının.
4. Eksenlerinizi etiketleyin: Eksenlerinizdeki birimleri ve ölçekleri net bir şekilde etiketleyin, böylece izleyicileriniz verileri anlayabilirler.
5. Başlık ve açıklama ekleyin: Başlık ve açıklama, grafiğinizin ana mesajını açıklamanıza ve bağlam sağlamanıza yardımcı olabilir.
6. Bu en iyi uygulamaları aklınızda bulundurarak, veri görselleştirmelerinizin net, etkili ve anlaşılır olduğundan emin olabilirsiniz.

Ayrıca, veri görselleştirmede Matplotlib'i kullanırken performansı da göz önünde bulundurmak önemlidir. Büyük miktarda veri ile uğraşırken, çizim süresi ve bellek kullanımını dikkate almak önemlidir.

Performansı iyileştirmenin bir yolu, `matplotlib.pyplot.xlim()` ve `matplotlib.pyplot.ylim()` yöntemlerini kullanarak sırasıyla x ve y eksenlerinin sınırlarını belirlemektir. Bu, çizilecek veri miktarını azaltarak çizim performansını iyileştirebilir.

Başka bir yol, birden çok figür yerine tek bir figürde birden çok grafik oluşturmak için `matplotlib.pyplot.subplots()` işlevini kullanmaktır. Bu, bellek kullanımını azaltarak çizim performansını artırabilir.

Bu en iyi uygulamaları ve ipuçlarını takip ederek, Matplotlib kullanırken veri görselleştirmelerinizin hem etkili hem de verimli olmasını sağlayabilirsiniz.

Doğru görselleştirme şeklini belirleme

Veriler için doğru görselleştirme şekli, verilerin türüne, boyutuna, özelliklerine ve anlatmak istediğiniz hikayeye bağlı olarak değişebilir. Bazı veriler, örneğin sayısal veriler, çizgi grafiği veya histogram gibi nicel grafiklerle daha iyi anlaşılabilirken, diğerleri, örneğin kategorik veriler, pasta grafikleri veya kutu grafikleri gibi nitel grafiklerle daha iyi anlaşılabilir.

Doğru görselleştirme şeklini belirlemek için aşağıdaki faktörleri dikkate alabilirsiniz:

- Verilerin tipi: Sayısal veriler, kategorik veriler, zaman serileri vb. farklı görselleştirme teknikleri gerektirir.
- Verilerin boyutu: Verileriniz ne kadar büyükse, daha karmaşık görselleştirme teknikleri gerektirebilirsiniz.
- Verilerin dağılımı: Verilerinizin dağılımı, verilerinizi nasıl görselleştireceğinizi etkileyebilir. Örneğin, normal dağılıma sahip veriler için histogram kullanmak doğru olabilirken, çarpık dağılıma sahip veriler için kutu grafikleri daha uygun olabilir.
- Mesajınız: Görselleştirmenin amacı nedir? Verilerinizde hangi mesajı vermek istiyorsunuz? Verilerinizin hikayesini en iyi anlatan görselleştirmeyi seçmelisiniz.

Doğru görselleştirme şeklini seçmek, verilerinizin en iyi şekilde anlaşılmasını sağlar ve hikayenizi en iyi şekilde anlatmanıza yardımcı olacaktır.

Seaborn Kütüphanesi

Seaborn, Python'da veri görselleştirme için kullanılan bir kütüphanedir. Matplotlib kütüphanesi temelde kullanılan grafiklerin yanı sıra, Seaborn kütüphanesi daha özel ve daha karmaşık grafiklerin oluşturulmasına yardımcı olur.

Seaborn ve Matplotlib

Seaborn ve Matplotlib, veri görselleştirme için Python dilinde kullanılan iki popüler kütüphanedir. Matplotlib, Python topluluğunun uzun süredir kullandığı bir kütüphanedir ve temel grafik türlerini çizmek için kullanılır. Seaborn ise Matplotlib'e dayanarak, daha karmaşık görselleştirme tekniklerine odaklanan daha yeni bir kütüphanedir.

- Seaborn, Matplotlib'in üzerine inşa edildiği için Matplotlib'i de kullanır. Ancak Seaborn, daha yüksek seviye araçlar sunar ve Matplotlib'den daha az kod yazmayı gerektirir. Örneğin, Seaborn ile daha karmaşık görselleştirmeleri yapmak daha kolaydır ve daha az ayarlama gerektirir. Seaborn, renkli grafikleri kolayca oluşturmak için önceden tanımlanmış renk paletleri de içerir.
- Matplotlib daha esnek bir yapıya sahiptir ve daha özelleştirilmiş grafikler oluşturmak için kullanılabilir. Matplotlib ile çok daha fazla özelleştirme yapılabilir ve her yönü tam olarak kontrol edilebilir.
- Seaborn, Matplotlib'in daha yüksek seviye araçlarına odaklanırken, Matplotlib daha özelleştirilmiş grafikler için daha fazla seçenek sunar. Seaborn ve Matplotlib birlikte kullanılabilir ve veri görselleştirme için güçlü bir kombinasyon sağlar.
- Seaborn, özellikle istatistiksel veri görselleştirmesi için tasarlanmıştır ve Matplotlib'den daha yüksek seviyeli bir arayüz sağlar. Buna ek olarak, Seaborn, renk paletleri, özel grafikler ve istatistiksel gösterimler için önceden tanımlanmış tema ve stiller gibi birçok özellik sunar.
- Seaborn ayrıca, regresyon analizi, faktör analizi ve kümeleme analizi gibi veri analizi tekniklerini de içeren birçok özel grafik fonksiyonuna sahiptir.

Özellikle, Seaborn kütüphanesi aşağıdaki görselleştirme türlerini içerir:

- Çizgi Grafikleri
- Dağılım Grafikleri
- Çift Değişkenli Dağılım Grafikleri
- Çizgi ve Nokta Çizgileri
- Bar Grafikleri
- Isı Haritaları
- Çift Eksenli Grafikler

Seaborn aynı zamanda, veri setlerindeki ilişkileri analiz etmek ve bu verileri görselleştirmek için kullanılan regresyon analizi, faktör analizi ve kovaryans analizi gibi istatistiksel yöntemler de sunmaktadır.

```
import seaborn as sns
```

```
tips = sns.load_dataset("tips")
tips.head()
```

replot()

Seaborn kütüphanesi içinde yer alan `relplot` fonksiyonu, verilerin ilişkisini incelemek ve görselleştirmek için kullanılan bir fonksiyondur. Bu fonksiyon, scatterplot ve lineplot fonksiyonlarının yerine geçebilen bir çoklu grafik çizim aracıdır.

`relplot` fonksiyonunun en önemli özelliği, veri kümesindeki iki veya daha fazla değişken arasındaki ilişkileri aynı anda göstermek için kullanılabilecek çoklu grafikler çizdirme olanağı sağlamasıdır. Bu grafiklerin türleri, kind parametresiyle belirlenebilir ve seçenekler arasında `scatter`, `line`, `strip`, `swarm`, `box`, `violin`, `bar`, `count` gibi seçenekler yer alır.

Ayrıca, col ve row parametreleri yardımıyla veri kümesinin farklı kategorilerine göre ayrılmış alt gruplarını da aynı anda görselleştirmek mümkündür. col ve row parametreleri, değişkenlerin sıralanması veya ayrılması için kullanılabilecek kategorik değişkenleri belirtmek için kullanılır.

Örneğin, aşağıdaki kod bloğu, relplot fonksiyonunu kullanarak tips veri kümesindeki `total_bill` ve `tip` değişkenleri arasındaki ilişkiyi gösteren bir scatter grafiği çizdirir:

```
sns.relplot(x='total_bill', y = 'tip', data = tips)
```

```
sns.relplot(x='total_bill', y = 'tip', data = tips, hue = 'day') # günlere göre ayrı
tipler
```

```
sns.relplot(x='total_bill', y = 'tip', data = tips, hue = 'day', style='time') # zamana
göre de ayrılmış
```

```
sns.relplot(x='total_bill', y = 'tip', data = tips, hue = 'day', style='time', col =
'sex') # cinsiyet de eklendi
```

catplot()

Seaborn kütüphanesi içinde yer alan `catplot()`, kategorik verilerin görselleştirilmesi için kullanılan bir fonksiyondur. catplot(), farklı tiplerdeki kategorik verileri farklı grafik türleri ile görselleştirmek için tasarlanmıştır.

catplot(), aşağıdaki grafik tiplerini destekler:

- stripplot: nokta grafiği
- swarmplot: sinek grafiği
- boxplot: kutu grafiği
- violinplot: keman grafiği
- boxenplot: kutu ve bıyık grafiği
- pointplot: nokta grafiği
- barplot: çubuk grafiği
- countplot: sayım grafiği

Örneğin, tips veri kümesi ile catplot() fonksiyonunu kullanarak, her gün için farklı öğünlerdeki ortalama hesap miktarlarını çizdirebiliriz:

```
g = sns.catplot(x='day', y='total_bill', hue='sex', kind='bar', data=tips)
plt.show()
```

Bu örnekte `catplot()` fonksiyonuna x, y ve hue parametreleri verilerek, x ekseninde günler, y ekseninde hesap miktarı, hue ile de cinsiyet bilgisi görselleştirilmiştir. Ayrıca kind parametresine bar değeri verilerek, çubuk grafiği çizdirilmiştir.

Tablo veya veri kümesinin özet istatistiklerini göstermek için kullanılan boxplot, Seaborn kütüphanesi içinde yer alan `boxplot()` fonksiyonu ile çizdirilebilir. Boxplot, verilerin çeyrekliklerini (çeyrekler arası yayılma), medyanı ve aykırı değerleri grafiksel olarak gösterir.

Örneğin, tips veri kümesindeki bahşiş miktarları ile boxplot çizdirelim:

```
sns.boxplot(x='day', y='tip', data=tips)
plt.show()
```

Çiftli Grafikler (Pair Plots)

Çiftli grafikler, veri kümesindeki tüm sayısal değişkenlerin birbirleriyle olan ilişkilerini gösteren bir tür grafiktir.

Seaborn kütüphanesi içinde bulunan `sns.pairplot()` fonksiyonu, veri kümesindeki tüm sayısal değişkenlerin birbirleriyle olan ilişkilerini görselleştirmek için kullanılabilir. Veri kümesindeki her bir sayısal değişkenin diğer sayısal değişkenlerle olan ilişkisini, histogramlar ve yoğunluk grafiği olarak gösterir. Bu sayede, veri kümesindeki değişkenler arasındaki ilişkiler daha kolay anlaşılabilir.

`sns.pairplot()` fonksiyonu kullanılırken, veri kümesindeki tüm sayısal değişkenlerin bulunması gerekmektedir. Fonksiyon otomatik olarak tüm sayısal değişkenleri bulur ve pair plotu oluşturur. Ayrıca, veri kümesindeki sayısal olmayan değişkenlerin de görüntülenmesi için, hue parametresi kullanılabilir.

Örneğin, bir veri kümesindeki tüm sayısal değişkenlerin birbirleriyle olan ilişkilerini görselleştirmek için `sns.pairplot()` fonksiyonu kullanılabilir:

```
iris = sns.load_dataset("iris")
iris.head()
```

```
sns.pairplot(iris, hue='species');
```

Yönlü Histogramlar (Faceted Histograms)

Yönlü Histogramlar, verileri farklı kategorilere veya gruplara ayırdıktan sonra her bir kategorideki dağılımı gösteren bir histogram türüdür. Bu tür histogramlar, farklı kategoriler arasındaki veri dağılımını karşılaştırmak için kullanılabilir. Bu grafikler, Seaborn kütüphanesi içinde yer alan `FacetGrid()` fonksiyonu ile oluşturulabilir.

```
tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']

grid = sns.FacetGrid(tips, row="sex", col="time", margin_titles=True)
grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```

`g.map()` fonksiyonu, her bir hücredeki total_bill ve tip değişkenlerinin dağılımını gösteren `histplot()` fonksiyonunu çağırır. g.add_legend() fonksiyonu ile renklerin neyi temsil ettiği gösterilir. Elde edilen grafik şöyle görünür:

Violin Grafikleri (Violin Plots)

Violin grafiği, veri setinin dağılımını görselleştirmek için kullanılan bir grafik türüdür. Genellikle kutu grafiği ile benzer bir amaca hizmet eder ancak kutu grafiğinin sunduğundan daha fazla bilgi sağlar. Violin grafiği, veri dağılımının yoğunluğunu, ortalamasını, medyanını ve çeyrekler arası aralığı görselleştirmek için kullanılır.

Violin grafiği, veri setindeki her bir değerin yoğunluğunu gösteren bir çekirdek (kernel) yoğunluk grafiği ve her bir değer için bir kutu grafiği içerir. Çekirdek yoğunluk grafiği, veri dağılımının şeklini gösterirken, kutu grafiği veri setinin merkezlerinin ve çeyrekler arası aralıklarının gösterilmesinde kullanılır.

Seaborn kütüphanesi, violin grafiğini oluşturmak için kullanılabilir. Veri setindeki her bir değişken çifti için bir çift violin grafiği oluşturulabilir. Bu, değişkenler arasındaki ilişkiyi ve aynı anda her bir değişkenin dağılımını görselleştirmek için kullanışlı bir yöntemdir.

Örneğin, “tips” veri setinde, “total_bill” ve “tip” değişkenleri arasındaki ilişkiyi göstermek için bir çift violin grafiği oluşturulabilir:

```
sns.violinplot(x='day', y='total_bill', hue='sex', kind='bar', data=tips)
```

Çörek grafikleri (Donut Plots)

Çörek grafikleri, pasta grafiklerine benzeyen, ancak merkezlerinde boşluk olan bir görselleştirme türüdür. Bu görselleştirme, yüzdelik oranları temsil etmek için kullanılır ve verilerin toplamının 100%’den daha az olduğu durumlarda özellikle yararlıdır.

Pasta grafiklerinde olduğu gibi, verilerin oransal dağılımını görselleştirmek için kullanılır. Ancak, çörek grafikleri, merkezlerindeki boşluk nedeniyle daha fazla bilgi sunar. Merkezdeki boşluk, grafikteki verilerin toplamının 100%'den az olduğu durumlarda kullanışlıdır. Bu boşluğu, grafikte gösterilmeyen bir diğer kategori için kullanabilirsiniz.

Matplotlib, Seaborn ve Plotly gibi birçok Python veri görselleştirme kütüphanesi donut grafikleri çizme işlevselliği sağlar.

Örnek bir çörek grafik oluşturmak için, öncelikle verilerinizin yüzdelik oranlarını hesaplamalısınız. Daha sonra, bu yüzdelik oranları kullanarak bir çörek grafik oluşturabilirsiniz.

Örnek kod:

```
import matplotlib.pyplot as plt

# Verileri oluşturma
veriler = [25, 35, 20, 10, 10]

# Renkler ve etiketleri belirleme
renkler = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'orange']
etiketler = ['A', 'B', 'C', 'D', 'E']

# Donut grafik oluşturma
fig1, ax1 = plt.subplots()
ax1.pie(veriler, colors=renkler, labels=etiketler, autopct='%1.1f%%', startangle=90,
pctdistance=0.85)
cember = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(cember)

# Grafik ayarları
ax1.axis('equal')
plt.tight_layout()

# Grafik gösterme
plt.show()
```

Bu kod, bir çörek grafik oluşturur ve ayrıca etiketler ve renkler ekler. Ayrıca, etiketlerin yüzdelik oranlarını da gösterir.

Kabarcık grafikleri (Bubble Charts)

Kabarcık grafikleri (bubble charts), verileri hem boyut hem de renk kullanarak görselleştiren bir grafik türüdür. Her bir nokta bir veriyi temsil eder ve konumları x ve y koordinatları ile belirtilirken, boyutları ve renkleri de farklı verileri temsil eder.

Kabarcık grafikleri, verilerin kategorik veya sayısal olarak gruplanabileceği herhangi bir veri seti için kullanılabilir. Özellikle, büyük veri kümelerinde, birkaç değişken arasındaki ilişkileri görselleştirmek için kullanışlıdır.

Kabarcık grafikleri, yüzey alanının ve rengin birlikte kullanımı sayesinde, birçok veri setinde daha fazla bilgi sunabilir. Ancak, çok sayıda veri noktası ile grafikler karmaşık hale gelebilir ve okunması zor olabilir.

Örnek olarak, bir ürünün satışını etkileyen faktörleri incelemek için bir kabarcık grafik kullanılabilir. X ve Y eksenleri, ürünün fiyatı ve pazarlama harcamaları gibi sayısal değişkenler olabilir. Kabarcık boyutu, ürünün satışlarını temsil ederken, kabarcık rengi değişik bölgelerdeki mağaza konumlarını temsil edebilir. Böyle bir grafik, veri analizcilerin farklı pazarlama stratejilerini test etmek için ürün fiyatı ve pazarlama harcamaları arasındaki ilişkiyi incelemelerine olanak tanır.

```
# Verileri oluşturma
sehirler = ['İstanbul', 'Ankara', 'İzmir', 'Bursa']
populasyon = [15029231, 5445026, 4279677, 2936806]
gelir = [15200, 12600, 13900, 11800]

# Kabarcık grafik çizimi
sns.scatterplot(x=populasyon, y=gelir, size=populasyon, sizes=(100, 1000), hue=sehirler)
plt.xlabel('Nüfus')
plt.ylabel('Ortalama Gelir')
plt.title('Şehirlerin Nüfus ve Ortalama Gelir Dağılımları')
plt.show()
```

Bu kodda, `sns.scatterplot()` fonksiyonu kullanılarak kabarcık grafik çizimi gerçekleştirilir. x ve y parametreleri verilerin x ve y eksenindeki dağılımını belirtirken, `size` parametresi kabarcık boyutlarını belirler. `sizes` parametresi minimum ve maksimum kabarcık boyutlarını belirlerken, `hue` parametresi renk skalasını belirlemek için kullanılır.

Sonuç

Bu bölümde, Matplotlib kütüphanesinin çeşitli özelliklerini ve fonksiyonlarını öğrendik. Matplotlib'i, bunun nasıl kullanılacağını ve veri görselleştirme için ihtiyaç duyduğumuz diğer çeşitli yararlı işlevleri ve yöntemleri anlıyor ve iyi uygulamalı uygulamalara sahibiz.

Veri analizi kavramlarının, araçlarının ve veri analizi için ihtiyaç duyduğumuz gerekli Python kitaplıklarının temellerini ele aldık.

Scikit-Learn ve Makine Öğrenmesi

Bu bölümde genellikle makine öğrenmesi kütüphanesi olarak bilinen Scikit-Learn kütüphanesinden bahsedilecektir. Makine öğrenmesi uygulamaları için Pandas, NumPy ve Matplotlib kütüphaneleri birlikte kullanılarak bir model geliştirilebilir. Ancak bu durumda makine öğrenmesi uygulaması için fonksiyonları tekrar tekrar yazmak gereklidir. Scikit-Learn kütüphanesi, makine öğrenmesi süreçlerini içerdiği fonksiyonlar ve matematiksel hesaplamaları sayesinde pratik birtakım yöntemler sunmaktadır. Bu kütüphane veri bilimi süreçlerini kolay bir şekilde yerine getirebilmek için tasarlanmıştır. Bu bölümde öncelikle makine öğrenmesi hakkında teorik bilgi verilecektir. Ardından Scikit-Learn kütüphanesi ile makine öğrenmesi konularına bakılacaktır.

Makine Öğrenmesine Giriş

Makine öğrenmesi ya da yapay öğrenme geçmiş verilerden faydalanarak gelecekteki verilerin öngörülmesi yani geçmiş verilerden öğrenmesi anlamına gelmektedir. Günümüzde verilerin toplanması ve depolanmasından ziyade veriden bilgi üretme çalışmaları daha değerli bir konuma sahiptir. Yani verinin dönüşümü, yapısal olmayan verilerin yapısal formata getirilmesi, verinin kullanılabilir hale getirilmesi, gizli bilgilerin ortaya çıkarılması çalışmaları ön plana çıkmaktadır. Makine öğrenmesi yapay zekanın uygulama alanlarından birisidir. Makine öğrenmesinin daha spesifik bir alanı ise derin öğrenmedir. Derin öğrenmede görüntü işleme, ses tanıma, akıllı robotlar gibi alanlarda çalışmalar gerçekleştirilir. Makine öğrenmesi uygulamalarında ise genellikle veri setlerinden bilgi çıkarılmaya çalışılır.

Makine öğrenmesi uygulaması için e-postaların spam veya spam olmama olarak iki ayrılması örnek olarak verilebilir. Daha önceden gönderilen ve kullanıcılar tarafından spam olarak işaretlenen veya işaretlenmeyen e-postalardan çıkarım yapılarak yeni e-postalar için bir sınıflandırma yapılabilir. Bu tip çalışmalar sınıflandırma veya gözetimli öğrenme çalışmaları olarak bilinmektedir. Sınıflandırma çalışmalarında geçmişteki verilerin etiketleri ya da hedef değişkeninin değerlerinin bilindiği varsayılır. Hedef değişkenin değerleri bilinmiyor ve tahmin edilmeye çalışılıyorsa bu tip çalışmalar da gözetimsiz öğrenme çalışmaları olarak bilinmektedir.

Makine Öğrenmesinde Temel Kavramlar

Daha önce veri madenciliğinin kalbinde veri vardır denilmişti. Makine öğrenmesinin en önemli kavramı ise öğrenme tanımıdır. Yapay öğrenmede insanlar ya da hayvanların öğrenme mekanizmaları taklit edilerek makineler üzerinde uygulanmak istenir. Makine öğrenmesindeki esas amaç genelleştirme kabiliyeti olan modeller ortaya koymaktır. Örneğin 100 hastaya ait hastalık verilerinin hastaneye yeni gelecek kişilerde de aynı özelliklerin olması durumunda hasta olup olmadığına bakılabilir. Bu işleme genelleştirme işlemi denilir.

Hasta veri seti örneğinde olduğu gibi eğer veri seti çıktı değişkenine sahipse bu öğrenme tipine gözetimli öğrenme denilir. Gözetimli öğrenmede sınıflandırma çalışmaları gerçekleştirilir. Veri setindeki gözlemlerin her biri girdi değişkenlerine(x) ve çıktı değişkenine(y) sahiptir. Dolayısıyla, sınıflandırma çalışmalarında x noktalarıyla y noktası açıklanmaya çalışılır.

Makine öğrenmesi modellerinde ayrıca çıktı değişkenin sayılabilir ya da sayılamaz olması ile de ilgilenilir. Eğer sayılabilir(kantitatif) çıktı değişkeni varsa bu durumda regresyon çalışmaları gerçekleştirilir. Eğer çıktı değişkeni sayılamaz(kalitatif) veri tipinde ise bu durumda sınıflandırma çalışmaları gerçekleştirilir. Regresyon çalışmalarında girdi değişkeni kategorik veya nümerik veri tipinde olabilir. Ancak çıktı değişkeninin nümerik veri tipinde olması gereklidir. Çıktı değişkeninin nümerik olması halinde eğer sınıflandırma çalışması yapılmak istenirse veriler kategorik hale dönüştürülerek işlem yapılabilir. Örneğin çıktı değişkeni 1-60 arasındaki yaşlardan oluşuyorsa 0-20 çocuk, 20-40 genç, 40 ve sonrası orta yaşlı denilerek bir kategorizasyon yapılabilir. Böylece problem sınıflandırma problemi olarak ele alınabilir. Bu bölümde makine öğrenmesi yaklaşımları gösterilecek ve regresyon analizi ile ilgili detaylı inceleme bir sonraki bölümde verilecektir.

Makine öğrenmesi, bir bilgisayar sisteminin veri örnekleri kullanarak kendisini eğitmesi ve belirli bir görevi yerine getirmesi için programlanmasıdır. Bu süreçte kullanılan öğrenme çeşitleri şunlardır:

Gözetimli öğrenme

Gözetimli Öğrenme (Supervised Learning): Bu öğrenme türünde, önceden belirlenmiş etiketler (labels) ile veriler kullanılarak, bir model oluşturulur. Bu model, daha sonra yeni verileri sınıflandırmak, tahmin etmek veya keşfetmek için kullanılır. Örnek olarak, bir e-postanın spam veya spam olmadığı gibi bir sınıflandırma problemi gözetimli öğrenme örneklerinden biridir.

Sınıflandırma çalışmaları için kullanılacak veri setini oluşturmak veya elde etmek kolay olmaz. Çünkü bazı durumlarda çıktı değişkeninin bilinmesi maliyetli olabilir. Örneğin bir hastalığa ait veri setinde çıktı değişkeni kişinin hasta/hasta değil durumları olsun. Bu durumda hastalık teşhisi gerçekleştikten sonra veri seti oluşabilecektir. Yani veri setindeki her kişi için hastalık teşhisinin yapılmış olması gereklidir. Bu da hastalık teşhisi için zaman ve para maliyetini getirir. Bu nedenle veri setinde çıktı değişkeninin olmaması durumunda da kullanılabilecek modellere ihtiyaç vardır. Bu modeller gözetimsiz öğrenme modelleri olarak bilinir. Gözetimsiz öğrenme modellerinde kullanılan veri setlerine çıktı değişkeni olmadığı için etiketsiz veri setleri de denilebilir. Gözetimsiz öğrenmenin en önemli örneği kümeleme çalışmalarıdır. Kümeleme çalışmalarında, daha önce veriler arasındaki uzaklık ve yakınlık ölçüleri ile veri noktaları arasındaki yakınlıklar ortaya koyularak benzer veriler aynı gruplara atanır. Gözetimsiz öğrenme çalışmaları bir keşif çalışmasıdır. Bu keşif çalışmasında veriler arasındaki ilişkiler, yakınlıklar veya uzaklıklar önemli yer tutar.

Gözetimsiz öğrenme

Diğer bir öğrenme çeşidi ise yarı gözetimsiz öğrenmedir. Gözetimsiz Öğrenmede (Unsupervised Learning), verilerdeki örüntüleri ve yapıları ortaya çıkarmak için kullanılır. Bu yöntemde verilerin etiketleri veya sınıflandırmaları yoktur. Örnek olarak, bir müşteri veri setinde benzerlik grupları oluşturmak denetimsiz öğrenme örneklerinden biridir.

Gözetimsiz öğrenme algoritmaları, veri kümesindeki örnekler arasındaki benzerlikleri ve farklılıkları belirleyerek veri kümesini önceden belirlenmiş gruplara ayırabilir. Buna kümeleme (clustering) denir. Örneğin, bir müşteri veri kümesi üzerinde yapılan bir kümeleme işlemi, benzer satın alma davranışları sergileyen müşterileri aynı gruba yerleştirebilir.

Gözetimsiz öğrenme ayrıca boyut indirgeme işlemlerinde de kullanılır. Bu işlem, veri kümesindeki öznitelik sayısını azaltarak verilerin işlenmesini kolaylaştırır ve daha az gürültülü sonuçlar elde edilir. Örneğin, bir resim veri kümesindeki yüksek boyutlu öznitelikleri, temel özelliklerine indirgenerek daha az boyutlu bir temsil ile ifade edilebilir.

Gözetimsiz öğrenme, özellikle büyük veri kümelerinde yapısal olmayan desenleri tespit etmek için kullanılır. Ancak, gözetimsiz öğrenme yöntemleri, gözetimli öğrenme yöntemleri kadar kesin sonuçlar veremeyebilir ve elde edilen sonuçların yorumlanması zor olabilir.

Yarı Gözetimli Öğrenme (Semi-Supervised Learning)

Bu öğrenme türünde, verilerin bir kısmı etiketlenmiş, diğer kısmı ise etiketlenmemiştir. Gözetimli ve denetimsiz öğrenme yaklaşımlarını birleştirir ve bir öğrenme modeli oluşturur. Bu model, etiketlenmemiş verileri sınıflandırmak, tahmin etmek veya keşfetmek için kullanılabilir.

Etiketli veriler, veri kümesindeki örneklerin doğru çıktı bilgilerinin olduğu verilerdir. Bu verilerin kullanılması, gözetimli öğrenme algoritmalarının eğitilmesine olanak tanır. Ancak, etiketli verilerin toplanması genellikle zaman alıcı ve maliyetlidir. Etiketlenmemiş veriler, doğru çıktı bilgilerinin olmadığı verilerdir. Yani, veri kümesindeki örneklerin etiketleri bilinmemektedir. Yarı gözetimli öğrenme, bu verilerin de kullanılmasını sağlar ve veri kümesinin daha iyi işlenmesine olanak tanır.

Yarı gözetimli öğrenme algoritmaları, hem etiketli hem de etiketlenmemiş verileri birleştirerek bir model inşa eder. Etiketlenmemiş verilerin kullanımı, modelin daha iyi özellikler (features) öğrenmesini sağlar ve böylece daha iyi bir performans elde edilir. Bu öğrenme yöntemi, özellikle veri kümesindeki etiketli verilerin sayısının az olduğu durumlarda etkili olabilir.

Yarı gözetimli öğrenme, çeşitli uygulamalarda kullanılır. Örneğin, bir web sitesindeki kullanıcıların davranışlarını takip etmek ve kullanıcıların hangi sayfaları ziyaret ettiğini, hangi ürünleri incelediğini vb. öğrenmek için kullanılabilir. Bu sayede, kullanıcılara daha iyi

hizmet sunulabilir ve öneriler yapılabilir. Ayrıca, tıbbi görüntülerde yapısal olmayan desenleri tanımlamak ve kanser tanısında yardımcı olmak gibi diğer uygulamalarda da kullanılabilir.

Pekiştirmeli (Takviyeli) öğrenme

Pekiştirmeli Öğrenme (Reinforcement Learning) türünde, bir karar verme modeli, verilen bir görev için ödüllendirilir veya cezalandırılır. Bu ödüller veya cezalar, modelin hangi kararları alması gerektiğini öğrenmesine yardımcı olur. Örnek olarak, bir oyun oynayan bir yapay zeka örneği düşünebilirsiniz. Burada, doğru hareketler yaparak oyunu kazanmak için ödüllendirilir, yanlış hareketler ise cezalandırılır.

Bu öğrenme türlerinin her biri farklı bir amaca hizmet etmektedir. Gözetimli öğrenme, sınıflandırma veya regresyon problemlerinde kullanılırken, denetimsiz öğrenme, veri analizi veya kümeleme gibi problemlerde kullanılır. Pekiştirmeli öğrenme, genellikle oyunlar, robotik veya otomatik sürüş gibi algoritmik karar problemleri için kullanılır.

Bu öğrenme yönteminde, model bir ortamda bir dizi eylem gerçekleştirir ve bu eylemler sonucunda bir ödül alır veya ceza alır. Modelin amacı, uzun vadede en yüksek ödülü alacak eylemleri öğrenmektir. Bu süreç, modelin başarılı eylemleri öğrenmesini ve hatalı eylemlerden kaçınmasını sağlar. Örneğin, bir robotun hareketlerini kontrol etmek için pekiştirmeli öğrenme kullanılabilir. Robot, bir ortamda çeşitli eylemler gerçekleştirir (örneğin, ilerler veya döner) ve bu eylemler sonucunda ödül veya ceza alır (örneğin, hedefe yaklaşır veya duvara çarpar). Bu süreç, robotun zamanla en etkili hareketleri öğrenmesini sağlar. Pekiştirmeli öğrenme, öğrencinin herhangi bir önceden tanımlanmış sonuç etiketine bağlı kalmaksızın, doğrudan çevreye yanıt vermesine izin verir. Bu nedenle, pekiştirmeli öğrenme, özellikle robotik, oyunlar, otomatik araçlar ve hatta tıbbi uygulamalar gibi uygulamalar için ideal bir öğrenme yöntemidir.

Eğitim ve Test Setleri

Gözetimli öğrenmede kullanılan veri seti öğrenmenin gerçekleştirildiği eğitim seti ve öğrenmenin test edildiği test seti olarak ikiye ayrılmaktadır. Birinci set eğitim seti, burada algoritmanın öğrenme gerçekleştireceği veriler bulunur. Sınıflandırmada özellikler ve hedef değişkenlerin olduğu bir settir. Test veri seti ise yine aynı şekilde gözlem değerlerinden oluşur. Ancak bu set eğitime girmez. Onun yerine eğitilen algoritmanın ne kadar iyi çalıştığı ile ilgili kullanılır. Eğitim ve test setleri birbirinden bağımsız olmalıdır. Yani ortak veri içermemelidir. Çünkü ortak veri olması durumunda eğitim setinin ezberlenmesi durumu söz konusu olabilir. Makine öğrenmesinde karşılaşılan önemli problemlerden birisi modelin ezberlemesidir. İngilizcede ezberleme olayına “over-fitting” denilir. Türkçede aşırı öğrenme olarak da isimlendirilebilir. Model eğitim sürecinde ezberler ise performansı yeni gelecek veriler için daha kötü olacaktır. Ezberlemeden kaçınmak için neler yapılacağı ile ilgili detaylı bilgi daha sonra verilecektir.

Bu 2 set dışında ayrıca üçüncü bir setten de bahsedilebilir. O da validasyon veya doğrulama setidir. Bu setin kullanım amacı modelin parametrelerini düzenlemektir. Modelin parametrelerine İngilizcede hyperparameters denilir. Algoritmanın performansı test seti üzerinden değerlendirilir, performans değerlendirmesi doğrulama seti üzerinden yapılmamalıdır. Genel olarak eğitim, test ve doğrulama setlerinin ayrımı için bir oran vermek doğru değildir. Her model için farklılık arz edebilir. Bir oran söylemek şart ise, eğitim seti için %50-60, test seti için %20-30 ve doğrulama seti için %20-30 aralığında veriler kullanılabilir denilebilir.

Çapraz Doğrulama

Algoritmanın performansının test edilebilmesi için kullanılabilecek bir yöntemdir. Bu yöntemde veri seti belirli sayıda parçaya bölünür. Bölünen parçalardan bir parça test veri seti olarak kullanılır. Diğer parçalar eğitim veri seti olarak kullanılır. Bu sayede tüm parçalar kullanılacak şekilde eğitimin performansı test edilmiş olunur. Parça sayısına İngilizcede “fold” denilir. Yani “10-folds cross validation” 10 parçalı çapraz doğrulama demektir. Aşağıdaki görselde çapraz doğrulamaya ilişkin bir şekil paylaşılmıştır. Görüldüğü gibi eşit parçalara bölünen veri setinde tüm parçalar test amacıyla kullanılacak şekilde bir doğrulama gerçekleştirilmektedir.

Performans Ölçütleri

Eğitim setinde gerçekleşen öğrenmenin test setinde öğrenme derecesi ile performans ölçütleri hesaplanmaktadır. Ele alınan makine öğrenmesi uygulamasına göre performans ölçütleri farklılık göstermektedir. Örneğin kategorik hedef değişkeninin olduğu sınıflandırma problemlerinde genel olarak kullanılan performans ölçütleri için şu örnekler verilebilir. Karışıklık matrisi sınıfların doğru tahmin edilip edilmediği hakkında bilgi veren bir matristir. Aşağıdaki görselde gösterildiği gibi TP(True Pozitive) doğru tahmin edilen pozitif verilerin sayısını, FP(False Positive) yanlış tahmin edilen negatif sayısını, FN(False Negative) yanlış tahmin edilen negatif değerlerin, TN(True Negative) ise doğru tahmin edilen negatif değerlerin sayısını göstermektedir.

Karışıklık matrisi ile hesaplanacak performans ölçütlerinden ilki doğruluk skorudur. İngilizcede Accuracy Score olarak geçer. Toplam doğru tahminlerin sayısının toplam sayıya bölünerek hesaplanır. En fazla kullanılan performans ölçüsü olmasına rağmen tek başına kullanılması yeterli olmayabilir. Eğer çıktı değerlerinin sayısı birbirinden çok farklı ise doğruluk skoru iyi sonuçlar vermez. Bu durumda kesinlik, duyarlılık ve F1 skoru ölçütlerine bakılabilir. Kesinlik(precision) değerinde pozitif tahmin edilenlerin tüm pozitifler içerisindeki oranı hesaplanır. Kesinlik skoru ile FP yani yanlış tahmin edilen pozitif değerlere dikkat çekilir. Diğer ölçüt ise duyarlılık(recall) performans ölçüsüdür. Bu ölçütte

ise yanlış tahmin edilen negatif değerlere dikkat çekilir. İki ölçütün bir arada kullanıldığı F1 skoru ise diğer bir performans ölçüsüdür. Performans ölçütlerinin formülasyonu aşağıda verilmiştir.

\[doğruluk = TP+TN / TP+FP+TN+FN\]
\[kesinlik=TP/(TP+FP)\]
\[duyarlılık=TP/(TP+FN)\]
\[F1=2\times(kesinlik\times duyarlılık)/(kesinlik+duyarlılık)\]

Regresyon Analizinde Performans Ölçütleri

Regresyon analizinde hedef değişkenler sürekli veriler olduğu için karışıklık matrisi kullanılamaz. Regresyon analizinde gerçekleşen ile tahmin edilen arasındaki farka hata terimi denilir. Hata terimi aşağıdaki görseldeki gibi gösterilebilir.

Regresyonda kullanılan performans ölçütlerine aşağıda yer verilmiştir. Ortalama Mutlak Hata(Mean Absolute Error): Tahminlerin gerçek değerlerden mutlak farkının ortalamasıdır. Ortalama Mutlak Hata

MAE=$\(\frac{1}{N}\sum_{i=1}^{N}\left|y_i-\hat{y}\right|\)$

Ortalama Karesel Hata (Mean Squared Error)

MSE=$\(\frac{1}{N}\sum_{i=1}^{N}\left(y_i-\hat{y}\right)^2\)$

Ortalama Karesel Hataların Karekökü (Root Mean Squared Error)

RMSE=$\(\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_i-\hat{y}\right)^2}\)$

R^2 Skoru (Determinasyon Katsayısı)

\[1-\frac{\sum_{i=1}^{N}\left(y_i-\hat{y}\right)^2}{\sum_{i=1}^{N}\left(y_i-\bar{y}\right)^2}\]

Makine Öğrenmesi Projeleri için Uygulanabilecek Adımlar

Makine öğrenmesindeki temel kavramlardan ve örnek çalışmalardan bahsettikten sonra bir makine öğrenmesi projesi geliştirilirken uygulanabilecek adımlara yer verilebilir.

Makine öğrenmesinin ilk aşamasında problemin ortaya koyulması vardır. Bu aşama bilimsel bir çalışma yapılırken genellikle ilk adım olarak bilinir. Bu aşamada problemin çözümündeki amaçlardan, problemin çerçevesinden, sınırlılıklarından ve kapasitesinden bahsedilebilir. Örneğin spam e-postaların sınıflandırılması probleminde amaç gelen kutusuna gelecek e-postaların filtrelenmesi ve spam e-postaların spam klasörüne gitmesini sağlamaktır.

Projenin sınırlılıklarına örnek olarak metin içeren e-postalar ya da 50 kelimenin üzerinde metin içeren e-postalar gibi kısıtlamalar eklenebilir. Bu bilgilerin ortaya koyulması ile problemin tanımlanması aşaması geçilmiş olunur.

İkinci aşamada verilerin elde edilmesi aşaması vardır. Daha önceki bölümlerde verilerin elde edilebileceği ücretli ve ücretsiz kaynaklardan bahsedilmişti. Kimi durumda veri setinin internet kaynaklarından bulunması mümkün olmayabilir. Proje için özel veri seti üretmek, veri toplamak, verileri istenilen formatta kaydetmek bu aşamada yerine getirilmesi gereken işlemlerden olabilir. Diğer aşamada veri setinin istatistiksel yöntemler ile analiz edilmesi aşaması vardır. Proje için belirlenen veya oluşturulan ilk veri seti analize başlamadan önceki haliyle ham veri seti(raw dataset) olarak isimlendirilir. Ham veri setinin analize hazır hale getirilmesi için daha önce bahsedilen veri önişleme süreçlerinin çalıştırılması gerekmektedir. Ardından veri seti içerisinde anlamlı ilişkilerin ortaya çıkarılması için korelasyon gibi ölçütlerin ortaya koyulması gerekebilir. Eğer bir regresyon çalışması yapılacaksa bağımsız değişkenler arasındaki korelasyonun düşük bağımsız değişken ile bağımlı değişken arasındaki korelasyonun ise yüksek olması istenir. Bu çerçevede bir analiz gerçekleştirilerek bazı bağımsız değişkenlerin analizden çıkarılması sağlanabilir. Ayrıca özellikler setindeki hedef değişkenle alakasız olduğu düşünülen değişkenler de sistemden çıkarılmalıdır. Spam e-posta örneğinde e-postaları gönderen kişilerin cinsiyetine bakmak gereksizdir. Cinsiyet özelliğinin veri setinden ve özellikler kümesinden çıkarılması başta düşünülmesi gereken bir süreç olmalıdır. Bunun gibi varsa başka alakasız özellikler bunların tespit edilerek analizden çıkarılması öğrenmenin kalitesine olumlu yönde etki edecektir. Bu aşamada özellik çıkarımı, özellik mühendisliği gibi yöntemlerden faydalanılmalıdır. Özellikler setine karar verildikten sonra ise veri setinin eğitim ve test seti ya da gerekli görülürse doğrulama veri seti olarak ayrılması işlemine geçilebilir.

Sonraki aşamada makine öğrenmesi algoritmasının ortaya koyulması var. Modelin seçilmesi ile modelin diğer modeller ile kıyaslanması, modelin performans değerlendirmesi gibi süreçler bu aşamada yerine getirilir.

Modeldeki parametrelerin ayarlanması, optimizasyonu süreçleri, modelin seçilmesinden sonra gerçekleştirilecek aşamadır. Bu aşamada parametre optimizasyonu veya hiperparametrelerin belirlenmesi işlemleri için literatürde geliştirilmiş yöntemlerden faydalanılabilir. Bu yöntemlere örnek olarak “Grid Search” verilebilir. Bu yöntem dışında çeşitli optimizasyon algoritmaları da kullanılabilir. Optimizasyon algoritmaları problemin karmaşıklığına göre sezgisel veya deterministik yöntemler olabilir.

Son aşamada ise projenin raporlanması ve sonuçlarının uygun şekilde sunulması işlemleri vardır. Bu aşamada gerekli istatistiksel testler, çapraz doğrulama sonuçları, sonuçların geçerlilik testleri gibi analizlerin verilmesi gerekir.

Yukarıda verilen aşamalar ayrıca şekilde özetlenmiştir. Makine öğrenmesi projeleri için bu aşamalar yerine getirilebilir. Tabii bu aşamalar şart olarak söylenemez. Bu aşamalar projeden projeye değişiklik gösterebilir.

Doğru makine öğrenmesi algoritmasının seçilmesi

Makine öğrenmesi algoritmalarını seçerken dikkat edilmesi gereken bazı faktörler vardır:

1. Veri türü: Verilerin türü, hangi makine öğrenmesi algoritması seçileceği konusunda önemli bir rol oynar. Sınıflandırma, regresyon veya kümeleme gibi farklı problemler için farklı algoritmalar mevcuttur.
2. Veri boyutu: Veri boyutu, hangi algoritmanın seçileceğini belirler. Büyük veri kümeleri için farklı algoritmalar kullanılabilirken, küçük veri kümeleri için farklı algoritmalar daha etkilidir.
3. Hız ve performans: Algoritmaların hızı ve performansı, belirli bir veri kümesinde nasıl çalışacağına dair bir fikir verir. Büyük veri kümeleri için daha hızlı algoritmalar tercih edilir.
4. Eğitim verileri: Hangi algoritmanın seçileceği, eğitim verilerinin doğru bir şekilde anlaşılmasıyla da ilgilidir. Eğitim verileri, algoritmanın öğrenme sürecini etkiler ve bu nedenle doğru bir şekilde analiz edilmelidir.
5. Doğruluk ve sonuçların yorumlanması: Makine öğrenmesi sonuçları doğru bir şekilde yorumlanmalıdır. Farklı algoritmaların doğruluğu farklı olabilir ve sonuçların ne anlama geldiği doğru bir şekilde anlaşılmalıdır.
6. Ölçeklenebilirlik: Algoritmanın ölçeklenebilir olması, daha büyük veri kümeleriyle çalışmak isteyenler için önemlidir.
7. Parametreler: Makine öğrenmesi algoritmaları, bir dizi parametre ile yapılandırılabilir. Hangi parametrelerin seçileceği, doğru sonuçların elde edilmesi için önemlidir.

Bu faktörlerin dikkate alınması, doğru makine öğrenmesi algoritmasının seçilmesine yardımcı olabilir. Ancak, bu seçim, deneyimli bir veri bilimcisi veya makine öğrenmesi uzmanı tarafından yapılmalıdır. Aşağıdaki görselde bir yol haritası gösterilmiştir. Ayrıca bu alanda yazılmış olan şu makale de önemli bir izlenim sunacaktır.

Başarısızlık Nedenleri

Makine öğrenmesi çalışmaları birçok denemeyi, deneme ve yanılmayı, tekrarlı çalışmaları, stresli işlemleri kapsayan zor bir iştir. Genel anlamda makine öğrenmesinin zorlukları veri setinden kaynaklı veya seçilen modelden kaynaklı olabilir. Veri setinin doğru seçilmemesi, eksik, hatalı, yanlış veriler içermesi gibi sorunlar makine öğrenmesi çalışmasını da zora sokabilir. Aynı şekilde veri seti doğru şekilde elde edilse de ardından uygulanan modelle ilgili sorunlar ortaya çıkabilir. Makine öğrenmesinde karşılaşılan zorluklar şu şekilde verilebilir.

- Yeterli verinin olmaması

- Verilerin kaliteli olmaması
- Doğru algoritmanın seçilmemesi
- Problemin makine öğrenmesi ile çözülmesinin mantıklı olmaması
- Aşırı öğrenme(Overfitting)
- Az Öğrenme (Underfitting)

Scikit-Learn Kütüphanesi Hakkında

Scikit-Learn kütüphanesine geçmeden önce Scikit-Learn ile kullanılan diğer kütüphaneler hakkında aşağıdaki bilgiler verilebilir. Verilen kütüphaneler, bilimsel çalışmalarda sıklıkla kullanıldığı için SciKits (Science Kits – Bilim Araçları) olarak bilinmektedir. SciKits kütüphaneleri aşağıdaki gibi listelenebilir.

- NumPy: N-Boyutlu diziler ve hesaplama kolaylığı sağlayan fonksiyonları sayesinde önemli avantajlar sunmaktadır.
- Pandas: Seriler ve veri çerçeveleri veri yapıları ile tablo formatındaki verilerin manipülasyonu ve analizi için önemli avantajlar sunmaktadır.
- Matplotlib: Verilerin görselleştirilmesi için kullanılmaktadır.
- SciPy: Bilimsel hesaplamalar için kullanılan Python kütüphanesidir.
- Scikit-Learn kütüphanesi açık kaynak kodlu bir Python kütüphanesidir. İlk olarak 2007 yılında Google Summer of Code etkinliğinde David Cournapeau tarafından geliştirilmiştir. Şu anda Google, the Python Software Foundation ve INRIA tarafından desteklenen 30’a yakın çalışanı ile geliştirilmesi devam etmektedir. Temel olarak NumPy ve SciPy kütüphaneleri üzerine kurulmuştur. Kütüphanenin ana sayfası http://scikit-learn.org/ adresinde bulunmaktadır.

Ana sayfada verildiği gibi Scikit-Learn kütüphanesi şu şekilde tanımlanabilir.

- Tahmine dayalı veri analizi için basit ve verimli araçlar
- Herkes tarafından erişilebilir ve çeşitli bağlamlarda yeniden kullanılabilir.
- NumPy, SciPy ve Matplotlib üzerine inşa edilmiştir.
- Açık kaynak kodlu, ticari olarak kullanılabilir - BSD lisansına sahip. Yine kendi sayfasından Scikit-Learn kütüphanesinin çalıştığı alanlar şu şekilde özetlenebilir.

Uygulama Alanı	Tanım	Uygulamalar	Algoritmalar
Sınıflandırma	Nesnenin hangi kategoriye ait olduğunu belirleme.	İstenmeyen posta algılama, görüntü tanıma.	en yakın komşular, rastgele orman ve daha fazlası…
Regresyon	Bir nesneyle ilişkili sürekli değerli bir özniteliği tahmin etme.	İlaç tepkisi, hisse senedi fiyatları.	en yakın komşular, rastgele orman ve daha fazlası…
Kümeleme	Benzer nesnelerin kümeler halinde otomatik gruplandırması.	Müşteri segmentasyonu, Gruplandırma deneme sonuçları	k-Means, spectral clustering, mean-shift,
Boyut Azaltma	Dikkate alınması gereken rastgele değişkenlerin sayısını azaltma.	Görselleştirme, Artan verimlilik	k-Means, feature selection, non-negative matrix factorization
Model Seçme	Parametreleri ve modelleri karşılaştırma, doğrulama ve seçme.	Parametre ayarlama ile geliştirilmiş doğruluk	grid search, çapraz doğrulama, metrikler
Veri Önişleme	Özellik ayıklama ve normalleştirme.	Metin gibi giriş verilerini makine öğrenimi algoritmalarıyla kullanmak üzere dönüştürme.	önişleme, özellik çıkarımı

Scikit-Learn kütüphanesinin öne çıkmasının en önemli nedenlerinden birisi de iyi belgelendirilmiş olmasıdır. Makine öğrenmesi metotları kimi zaman karmaşık matematiksel ifadeler içerebilir. Scikit-Learn olabildiğince basit bir şekilde makine öğrenmesi metotlarını sunmaktadır. Böylece ileri düzey bir matematiğe ihtiyaç duymadan da makine öğrenmesi uygulaması geliştirmek mümkün olabilmektedir.

Bu özellikleri sayesinde Scikit-Learn birçok önemli işletme tarafından kullanılmaktadır. Örneğin Spotify, Scikit-Learn kütüphanesi ile müşterilerine yeni müzikler önermektedir. Geçmişteki müzik dinlemeleri ve müzik türleri hakkındaki veriler kullanılarak dinleyicilere daha önce dinlemedikleri ve dinlemek isteyebilecekleri müzikler önerilir. Diğer bir örnek de not alma uygulaması olan Evernote uygulamasıdır. Evernote da Scikit-Learn kütüphanesini sıklıkla projelerinde kullanmaktadır.

Scikit-Learn Kütüphanesinin Yüklenmesi

Scikit-Learn kütüphanesini bilgisayara yüklemek için aşağıdaki şekilde gösterilen komutların komut istemcisine veya Anaconda Prompt içerisine yazılması gereklidir. Ancak Anaconda yüklemesi gerçekleştirilince Scikit-Learn kütüphanesi de yüklü gelmektedir. O nedenle yeniden yüklenmesine gerek yoktur. Eğer farklı bir ortamda çalışılacaksa ortam içerisine Scikit-Learn kütüphanesinin eklenmesi için bu yöntem kullanılabilir.

Yükleme doğrulandıktan sonra Jupyter notebook içerisinde aşağıdaki kodlar yardımıyla Scikit-Learn kütüphanesi çağrılabilir.

```
import sklearn as sk
import numpy as np
import pandas as pd
sk.__version__
```

```
'0.24.2'
```

Kitapta kullanılan versiyon 0.24.1 versiyonudur. İlerleyen kısımlardaki bazı kodlar farklı versiyonlarda doğru çalışmayabilir.

Scikit-Learn Veri Setleri

Veri madenciliği yöntemlerinin uygulanabilmesi için veri setlerine ihtiyaç vardır. Veri setleri kaynaklarından birisi de Scikit-Learn veri setleridir. Bu kaynağın içerisinde de bilinen ve sıklıkla kullanılan veri setleri yer almaktadır. Yani Scikit-Learn veri setleri genellikle çok ön işleme aşaması gerektirmeyen, analize hazır veri setleridir. Makine öğrenmesinde kullanılan veri setleri çoğunlukla 2 boyutludur yani tablo formatındadır. Tablodaki her satır bir veriyi temsil eder. Sütunlar ise özellikleri gösterir. Aşağıdaki görselde Boston veri seti için satırlar ve sütunlar gösterilmiştir.

Veri setleri Scikit-Learn kütüphanesi ile gelmektedir. Eğer kütüphane yüklemesi tamamlandıysa veri setleri de bilgisayara yüklenmiş olacaktır. Veri setleri bilgisayarda csv dosyaları şeklinde bulunabilir. Bu işlem için izlenmesi gereken yol “ANACONDA YÜKLEME KLASÖRÜ”\pkgs\scikit-learn-0.24.1-py37hf11a4ad_0\Lib\site-packages\sklearn\datasets\data”.

Veri setlerine bilgisayardan ulaşma
Scikit-Learn veri setleri “toy” veri setleri olarak bilinir. Yani makine öğrenmesine yeni giriş yapacak kişiler için önerilen veri setlerinden bahsedilmektedir. Bu veri setleri hakkında kısaca bilgi verilmesi gerekirse;

- Boston Ev Fiyatları (boston_house_prices.csv): Binanın yaşı, dairenin oda sayısı, binanın bulunduğu bölgedeki suç oranı gibi birtakım özelliklere bakılarak Boston eyaletindeki evlerin fiyatlarının bulunduğu veri setidir. Bu özellikler kullanılarak bir evin fiyatı tahmin edilmeye çalışılır.
- Meme Kanseri (breast_cancer.csv): Bir tıp veri setidir.
- Diyabet (diabetes_data.csv): Diyabet hastalığı veri setidir.
- Zambak Çiçeği (iris.csv): Zambak çiçeğinin türlerinin olduğu veri setidir.
- Şarap Verileri (wine_data.csv): Şarap verilerine ait veri setidir.

Bu veri setleri Jupyter notebooka çağrılırken sklearn.datasets içerisinden çağrılır. Aşağıdaki yöntem ile tüm veri setleri Jupyter notebook içerisine alınmış olunur.

```
from sklearn import datasets
boston = datasets.load_boston()
type(boston)
```

```
sklearn.utils.Bunch
```

Veri setlerinin kendilerine ait “load_VERİSETİ()” şeklinde bir yapısı mevcuttur. Dolayısıyla veri seti çağrılırken “load_boston()” ile çağrıldı. Aynı şekilde eğer meme kanseri veri seti çağrılmak istenseydi “load_breast_cancer()” fonksiyonu ile çağrılması gerekirdi.

```
breast_cancer = datasets.load_breast_cancer()
type(breast_cancer)
```

```
sklearn.utils.Bunch
```

```
# Bunch içerisinde hangi anahtarlar var?
print(boston.keys())

# Anahtarların veri yapıları neler?
type(boston.data),type(boston.target),type(boston.DESCR),type(boston.feature_names)
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```
(numpy.ndarray, numpy.ndarray, str, numpy.ndarray)
```

Görüldüğü gibi veri setlerinin veri yapısı Bunch olarak gösterilmektedir. Bunch Python sözlüklerinin Scikit-Learn kütüphanesindeki karşılığı olarak görülebilir. Daha önce bahsedildiği gibi sözlük içerisinde anahtar:değer ilişkileri bulunmaktadır. Sözlük içerisindeki anahtar(keys) değerleri girilerek karşılığında gelecek değerler alınır. Bu sözlüğü parçalayarak veri setinin içeriğinden bahsedilmek gerekirse aşağıdaki kodlar kullanılabilir.

Boston veri seti için geçerli olan bu anahtarlar diğer veri setleri için de geçerlidir. Anahtarlar genellikle NumPy dizisi veri tipinde tutulmuştur. Anahtarlar hakkında kısaca bilgiler aşağıda verilmiştir.

- data (boston.data): Bir NumPy dizisi şeklinde verilerin tutulduğu anahtardır. NumPy dizilerinde sunulan avantajlardan faydalanılarak incelenebilir.

- target(boston.target): Hedef değişkeninin tutulduğu anahtardır. Yine NumPy dizisi şeklinde tutulur. Data ve target veri yapıları veri setini oluşturacak şekilde ayarlanmıştır. Aşağıdaki görselde Scikit-Learn kütüphanesinde geçtiği şekilde veri setlerinde kullanılan kavramlar tekrar gösterilmiştir.

Scikit-Learn Veri Setlerinin Yapısı

Boston veri setindeki veriler X değişkenine, hedef değişkeni de y değişkenine kaydedilebilir.

```
# X ve y değişkenleri
X = boston.data
y = boston.target
boston_features = pd.DataFrame(X, columns=boston.feature_names)
```

```
boston_features.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```
dfy = pd.DataFrame(y,columns=['price'])
dfX = pd.DataFrame(X, columns=boston.feature_names)
df = dfX.join(dfy)
```

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Veri Önişleme Aşaması

Veri önişleme makine öğrenmesinde oldukça önemli bir aşamadır. Veri önişleme, veri setlerini temizleme, düzenleme ve hazırlama sürecidir. Bu süreç, verilerin doğru bir şekilde analiz edilebilmesi ve makine öğrenmesi modellerinin doğru sonuçlar vermesi için gereklidir.

Veri önişleme aşamaları şunları içerebilir:

1. Veri toplama: Makine öğrenmesi modeli oluşturmak için uygun verilerin toplanması gereklidir. Bu veriler, çeşitli kaynaklardan gelir ve genellikle büyük miktarda veri içerir.
2. Veri ön işleme: Verilerin temizlenmesi ve hazırlanması için birkaç adım gereklidir. Bu adımlar, verilerin düzenlenmesini, eksik verilerin giderilmesini ve verilerin normalleştirilmesini içerebilir. Ayrıca, verilerin özelliklerinin seçimi de bu adımda gerçekleştirilir.
3. Veri bölme: Veri setleri genellikle eğitim, test ve doğrulama setleri olarak bölünür. Eğitim seti, modelin eğitiminde kullanılır, test seti modelin doğruluğunu kontrol etmek için kullanılır ve doğrulama seti, modelin performansını izlemek için kullanılır.
4. Veri özellikleri seçimi: Veri setleri genellikle birçok özelliğe sahiptir. Bu özelliklerin sayısı, modelin karmaşıklığını artırabilir ve aynı zamanda modelin performansını düşürebilir. Bu nedenle, veri özellikleri seçimi yapılırken, gereksiz özelliklerin çıkarılması ve yalnızca önemli özelliklerin kullanılması gereklidir.
5. Veri normalleştirme: Veri setleri, farklı aralıklarda olabilen özellikler içerebilir. Bu nedenle, verilerin normalleştirilmesi gereklidir. Normalizasyon işlemi, verilerin ortalama değerinin sıfır ve standart sapmasının bir olduğu bir ölçekte yeniden ölçeklendirilmesini içerir.

Veri önişleme, makine öğrenmesinde oldukça önemlidir ve doğru bir şekilde yapılması, modelin doğruluğunu artırabilir.

Eğitim ve test setlerinin ayrılması

Makine öğrenmesi modellerinin performansını ölçmek ve doğru sonuçlar elde etmek için, veriler eğitim seti ve test seti olarak ayrılmalıdır. Eğitim seti, modelin öğrenme sürecinde kullanılan veri kümesidir ve test seti, modelin doğruluğunu ölçmek için kullanılan veri kümesidir.

Eğitim seti, modelin öğrenmesinde kullanılan verilerdir. Model, bu verileri kullanarak özelliklerin arasındaki ilişkileri öğrenir ve sonunda bir tahmin yapmak için bu özellikleri kullanabilir. Eğitim setinin yeterince büyük ve temsil edici olması, modelin doğru bir şekilde öğrenmesi için önemlidir.

Test seti, modelin performansını ölçmek için kullanılan verilerdir. Model eğitildikten sonra, test setindeki verileri kullanarak tahminler yaparız ve gerçek sonuçlarla karşılaştırırız. Bu sayede modelin doğruluğunu ölçeriz ve gerektiğinde iyileştirmeler yaparız.

Veri kümesinin eğitim seti ve test seti olarak nasıl ayrılacağına karar vermek için farklı yöntemler kullanılabilir. En yaygın yöntem, verilerin belirli bir oranda rastgele bölünmesidir. Örneğin, verilerin %80'i eğitim seti olarak kullanılırken, %20'si test seti olarak kullanılabilir.

Eğitim ve test setleri ayrıldıktan sonra, eğitim seti üzerinde model eğitilir ve ardından test seti üzerinde doğrulama yapılır. Bu işlem, modelin gerçek dünya verilerinde nasıl performans gösterdiğini anlamak için önemlidir.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.20,
                                                    random_state=12345)
```

Bu kodlar, Scikit-learn kütüphanesinin `model_selection` modülünden `train_test_split` fonksiyonunu kullanarak, veri setini rastgele eğitim ve test alt kümelerine ayırmak için kullanılır.

`X` özellik matrisini, `y` hedef değişkenini ve `test_size` parametresi olarak verilen oranı kullanarak, veri setini eğitim ve test setleri olarak ayırır. `random_state` parametresi ise her seferinde aynı rastgele bölmenin oluşturulmasını sağlar.

Fonksiyon çağrısı sonucunda, `X_train` ve `y_train` eğitim setini, `X_test` ve `y_test` ise test setini temsil eder. Bu şekilde, veri seti iki ayrı alt kümeye ayrılarak, makine öğrenmesi modelinin eğitiminde kullanılan veri ile test edilmesi için kullanılan veri ayrılmış olur.

Eksik veriler

```
df.isnull().sum()
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
price      0
dtype: int64
```

```
# özet bilgileri sunalım.
df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
count	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0	506.0
mean	3.613	11.24	11.13	0.06	0.55	6.284	68.57	3.795	9.549	408.2	18.45	356.67	12.65	22.53
std	8.601	23.32	6.86	0.25	0.11	0.702	28.14	2.105	8.707	168.5	2.164	91.29	7.141	9.197
min	0.006	0.0	0.46	0.0	0.385	3.561	2.9	1.129	1.0	187.0	12.6	0.32	1.73	5.0
25%	0.082	0.0	5.19	0.0	0.449	5.885	45.02	2.1	4.0	279.0	17.4	375.38	6.95	17.02
50%	0.256	0.0	9.69	0.0	0.538	6.208	77.5	3.207	5.0	330.0	19.05	391.44	11.36	21.2
75%	3.677	12.5	18.1	0.0	0.624	6.623	94.07	5.188	24.0	666.0	20.2	396.225	16.955	25.0
max	88.976	100.0	27.74	1.0	0.871	8.78	100.0	12.126	24.0	711.0	22.0	396.9	37.97	50.0

Ölçeklendirme

Makine öğrenmesi modellerinin performansı, verilerin ölçeği ile doğrudan ilişkilidir. Ölçeklendirme, verilerin farklı ölçeklerde veya aralıklarda olduğu durumlarda önemlidir. Örneğin, bir veri kümesinde bir özellik binlerce dolarlık aralıklarda iken, diğer bir özellik 0 ile 1 arasında bir aralığa sahip olabilir. Bu durumda, doğru sonuçlar almak için verilerin ölçeklendirilmesi gerekebilir.

Ölçeklendirme işlemi, verilerin belirli bir aralığa veya dağılıma getirilmesini içerir. Verileri ölçeklendirmek için yaygın olarak kullanılan iki yöntem vardır:

- Normalizasyon: Verilerin belirli bir aralığa sığdırılması için kullanılır. Örneğin, verilerin 0 ile 1 arasında ölçeklendirilmesi için Min-Max normalizasyonu kullanılabilir.
- Standartlaştırma: Verilerin bir standart dağılıma sahip olacak şekilde ölçeklendirilmesini sağlar. Bu, verilerin ortalama değerinden çıkarılması ve standart sapmaya bölünmesi ile yapılır.

MinMaxScaler ile Veri Ölçeklendirme

- `fit()`: ölçeklemede belirli bir verinin ortalamasını ve std sapmasını hesaplamak için kullanılır.
- `transform()`: fit() yöntemi ile hesaplanan ortalama ve std sapma kullanılarak ölçekleme gerçekleştirmek için kullanılır.
- The `fit_transform()` hem fit hem de transform eder.

$$x_{std} = \frac{x - x_{\min (\text{axis}=0)}}{x_{\max (\text{axis}=0)} - x_{\min (\text{axis}=0)}}$$

$$x_{scaled} = x_{std} * (max – min) + min$$

```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler()
```

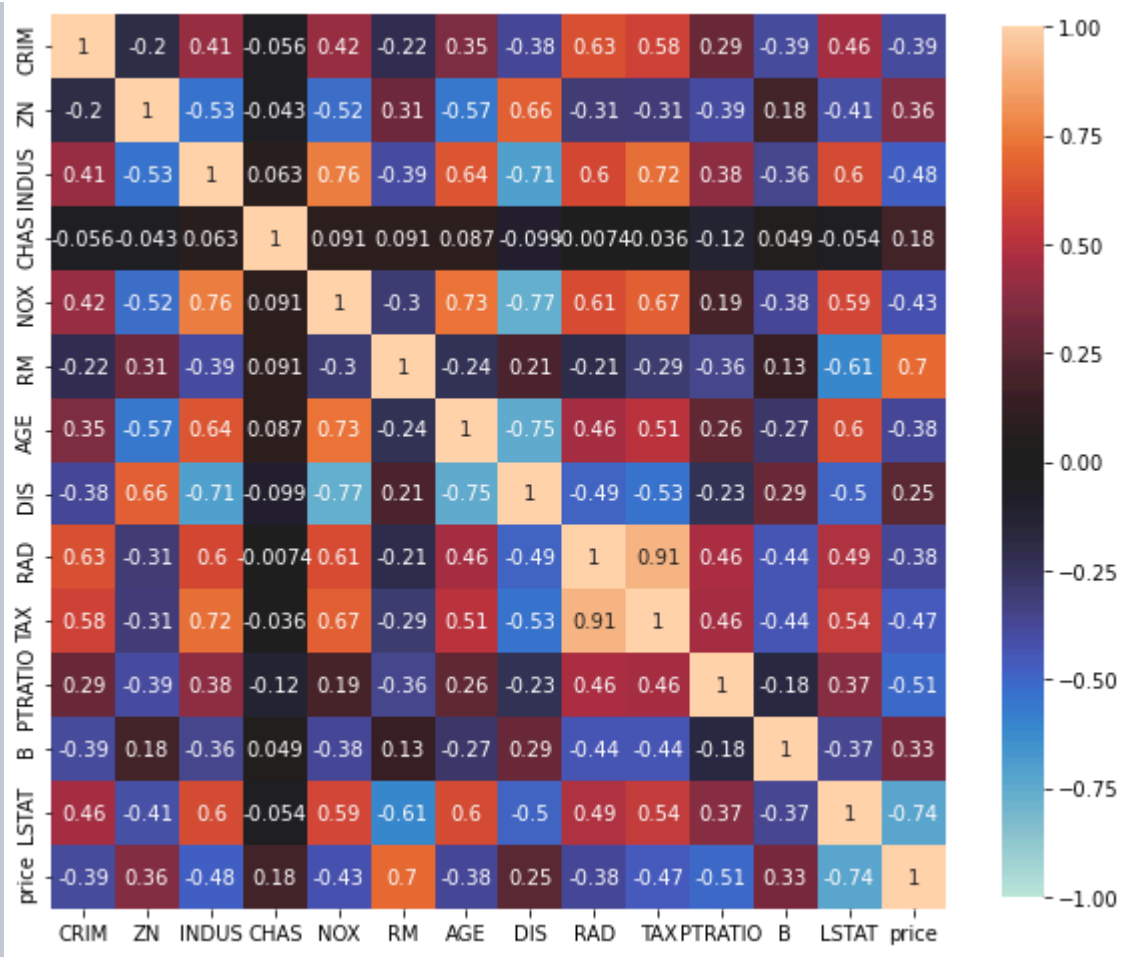
Ölçeklendirme, yalnızca eğitim kümesindeki verileri kullanarak, verileri eğitim ve test kümesi arasında böldükten sonra yapılmalıdır. Çünkü test seti görülmeyen yani yeni verileri göstermektedir. Eğitime test verilerinin katılmaması gerekir. Eğitimden önce ölçeklendirme yapmak bu nedenle doğru olmayacaktır.

```
X_train_scaled = mm_scaler.fit_transform(X_train)

print(mm_scaler.min_)
print(mm_scaler.scale_)
np.set_printoptions(precision=3, suppress=True)

print(X_train_scaled[:5])
```

```
[-7.10352762e-05  0.00000000e+00 -1.68621701e-02  0.00000000e+00
 -7.92181070e-01 -6.82314620e-01 -2.98661174e-02 -1.02719857e-01
 -4.34782609e-02 -3.56870229e-01 -1.34042553e+00 -6.38977636e-03
 -4.77373068e-02]
[1.12397589e-02 1.00000000e-02 3.66568915e-02 1.00000000e+00
 2.05761317e+00 1.91607588e-01 1.02986612e-02 9.09347180e-02
 4.34782609e-02 1.90839695e-03 1.06382979e-01 2.53562554e-03
 2.75938190e-02]
[[0.001 0.    0.236 0.    0.13  0.615 0.    0.418 0.087 0.088 0.564 0.971
  0.086]
 [0.001 0.    0.42  0.    0.387 0.654 0.907 0.094 0.    0.164 0.894 1.
  0.108]
 [0.006 0.    0.21  0.    0.245 0.464 0.671 0.231 0.304 0.229 0.511 0.953
  0.274]
 [0.    0.8   0.048 0.    0.    0.511 0.295 0.724 0.    0.103 0.596 0.86
  0.309]
 [0.001 0.    1.    0.    0.461 0.464 0.83  0.089 0.13  1.    0.798 1.
  0.321]]
```

```
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 10))
sns.heatmap(df.corr(),center=0, vmin=-1, vmax=1, square=True, annot=True,cbar_kws=
{'shrink': 0.8})
```

```
<AxesSubplot:>
```




```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  price    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

```
X.shape, y.shape # Orijinal veri seti
```

```
((506, 13), (506,))
```

```
X_train.shape, y_train.shape # eğitim veriseti
```

```
((404, 13), (404,))
```

```
X_test.shape, y_test.shape # test veri seti
```

```
((102, 13), (102,))
```

Doğrusal Regresyon

Doğrusal regresyon, iki değişken arasındaki ilişkiyi anlamak için kullanılan bir istatistiksel modellemedir. Bu ilişki doğrusal olarak ifade edilir ve bir çıktı değişkeni (bağımlı değişken) ile bir veya daha fazla girdi değişkeni (bağımsız değişkenler) arasındaki ilişkiyi tanımlar. Doğrusal regresyon, özellikle sürekli sayısal verilerde, bir değişkenin diğer değişkenlerle olan ilişkisini anlamak ve tahminler yapmak için sıklıkla kullanılan bir yöntemdir. Doğrusal regresyon modelleri, verilerdeki değişkenliği açıklayan en az kareler yöntemi kullanılarak elde edilir.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
model = LinearRegression()
model.fit(X_train_scaled,y_train)
```

```
LinearRegression()
```

Bu kodlar, doğrusal regresyon modeli oluşturma ve eğitme işlemini gerçekleştirir. İlk olarak, `sklearn.linear_model` kütüphanesinden `LinearRegression` sınıfı çağrılır ve `model` adı verilen bir örnek oluşturulur. Daha sonra, `fit()` fonksiyonu kullanılarak oluşturulan model eğitilir. Bu işlem, `X_train_scaled` ve `y_train` veri setleri üzerinde gerçekleştirilir.

Ayrıca, doğrusal regresyon modelinin performansını değerlendirmek için `sklearn.metrics` kütüphanesinden `r2_score`, `mean_absolute_error` ve `mean_squared_error` fonksiyonları çağrılır. Bu fonksiyonlar, sırasıyla, R-kare skoru, ortalama mutlak hata ve ortalama karesel hata değerlerini hesaplar. Bu değerler, modelin doğruluğunu değerlendirmek için kullanılabilir.

Özetle, bu kodlar doğrusal regresyon modelini oluşturur, eğitir ve performansını değerlendirir.

```
print("katsayılar: ", model.coef_)
print("kesme terimi: ", model.intercept_)
print("katsayı sayısı: ", model.coef_.shape)
```

```
katsayılar:  [ -9.421   4.949   0.774   2.974  -8.379  23.783  -0.556 -14.66    6.019
  -5.746  -9.308   3.172 -16.17 ]
kesme terimi:  24.309530869937667
katsayı sayısı:  (13,)
```

Burada:

`model.coef_` ifadesi, öğrenilmiş regresyon modelinin her bir öznitelik için katsayılarını içeren bir NumPy dizisini döndürür.

`model.intercept_` ifadesi, öğrenilmiş regresyon modelinin kesme terimini döndürür.

`model.coef_.shape` ifadesi, katsayı dizisinin şeklini gösterir, yani modelde kaç öznitelik olduğunu gösterir.

```
X_test_scaled = mm_scaler.transform(X_test)
```

`mm_scaler.transform(X_test)` kodu, önceden oluşturulan `MinMaxScaler` nesnesi olan `mm_scaler`'ı kullanarak test verilerinin ölçeklendirilmesini sağlar. `X_test` veri seti, önceden eğitilen ölçekleyici nesnesi `mm_scaler` ile ölçeklendirilir ve ölçeklendirilmiş test verileri `X_test_scaled` olarak kaydedilir. Bu, eğitim verileriyle aynı ölçeklendirme işleminin kullanıldığından emin olmak için önemlidir ve daha sonra test verileri üzerinde tahmin yapmak için kullanılacaktır.

```
y_predict = model.predict(X_test_scaled)
```

Modelin `predict()` metodu kullanılarak, önceden ölçeklendirilmiş test verileri olan `X_test_scaled` argümanı modelin tahmin etmesi için kullanılır ve sonuç `y_predict` değişkeninde saklanır. Bu sonuçlar daha sonra modelin performansını değerlendirmek için gerçek test hedefleri `y_test` ile karşılaştırılabilir.

```
y_predict
```

```

```

```
array([24.217, 31.641, 20.588, 22.856, 16.684, 14.317, 26.809, 25.617,
       32.369, 16.666, 24.377, 16.778, 28.064, 16.162, 20.256, 20.495,
       20.951, 33.067, 10.946, 17.373, 21.892, 19.969,  8.552, 25.359,
       32.245, 30.801, 14.852, 27.706, 16.94 , 17.08 , 28.097, 19.534,
       14.018, 10.163, 14.051, 25.358, 30.418, 19.687, 31.597, 21.768,
       28.319, 14.314, 29.707, 17.699, 33.596, 28.267, 27.92 , 23.356,
       17.563,  9.874, 20.834, 19.42 , 18.267, 19.79 , 33.428, 19.927,
       19.93 , 36.306, 17.491, 37.359, 22.827, 19.233, 13.543, 17.412,
       14.857, 34.159, 25.847, 36.226, 16.213, 34.577,  8.936, 27.602,
       21.733, 29.249, 19.118, 32.53 , 31.958, 14.508, 27.029, 14.379,
       20.339, 11.307, 25.34 , 24.702, 17.094, 17.761, 19.012, 21.203,
       21.284, 19.444,  5.939, 28.83 , 20.001, 13.202, 21.267, 42.906,
       20.044, 28.248, 24.296, 30.158, 16.669, 28.498])
```

```
y_test
```

```
array([21.7, 24.8, 20.6, 23. , 19.1, 15. , 22.6, 24.7, 27. , 20.8, 27.5,
       19.1, 22. , 10.2, 18.3, 21.7, 21.2, 29. , 16.3, 19.9, 19.8, 15. ,
       23.1, 26.5, 35.4, 31.5, 14.8, 24.8, 23.1, 13.4, 22.8, 18.4, 21.9,
       17.8, 13.1, 24.2, 29.4, 19.5, 37. , 18.9, 24.5, 11.7, 22.5, 18.6,
       28.5, 24.3, 25. , 22.4, 20.8, 14.4, 21.5, 22.2, 14.5, 17.7, 36.1,
       19.3, 19.2, 38.7, 17.4, 33.3, 17. , 19.1, 13.1, 18.1,  9.6, 30.3,
       19.4, 50. , 17.6, 34.9, 11.8, 26.6, 20.6, 30.1, 12.5, 33.2, 27.9,
        7.5, 25.2, 19. , 21.8,  6.3, 23.3, 21.9, 14.9, 18.2, 20.6, 50. ,
       21. , 18.9, 13.8, 28.6, 13.1, 15.6, 19.3, 50. , 21.8, 28.7, 25. ,
       32.5, 17.3, 24.6])
```

Modelin Değerlendirilmesi

Modelin değerlendirmesi, öncelikle belirlenen performans metrikleri kullanılarak yapılır. Bu metrikler, modelin ne kadar doğru sonuçlar verdiğini ölçer. Örneğin, sınıflandırma problemleri için doğruluk (accuracy), hassasiyet (precision), duyarlılık (recall) ve F1-score kullanılabilir. Regresyon problemleri için ise ortalama mutlak hata (mean absolute error), ortalama karesel hata (mean squared error) ve belki de belirlenen bir eşiğe göre doğruluğu ölçen bir metrik kullanılabilir.

Değerlendirme işlemi, eğitim verileri üzerinden eğitilen modelin test verileri üzerinde nasıl performans gösterdiğini ölçerek yapılır. Bunun için, öncelikle test verileri üzerinde modelin tahminleri hesaplanır ve gerçek sonuçlarla karşılaştırılır. Bu karşılaştırma sonucunda elde edilen performans metrikleri, modelin performansını değerlendirmeye yardımcı olur.

Ayrıca, bazı durumlarda çapraz doğrulama (cross-validation) gibi teknikler de kullanılabilir. Bu teknikler, farklı veri parçaları üzerinde modelin performansını değerlendirerek, sonuçların daha güvenilir olmasını sağlar. Bu örnekte R^2 kullanılacaktır. `score()` fonksiyonu determinasyon katsayısını verir.

```
model.score(X_train_scaled, y_train)
```

```
0.7586732434392979
```

```
# Determinasyon katsayısını verir.
model.score(X_train_scaled,y_train)
```

```
0.7586732434392979
```

```
r2_score(y_test,y_predict)
```

```
0.6160762605146681
```

```
# measure the performance of the model
mse = mean_squared_error(y_test, y_predict)
rmse = np.sqrt(mse)
print(rmse)
```

```
4.962046573144257
```

```
mse
```

```
24.621906194052666
```

Diğer metrikler için bu sayfadan daha fazla bilgi alınabilir.

Çapraz doğrulama

Değerlendirme aşamasında bir diğer yaklaşım da çapraz doğrulamadır. Daha önce bahsedildiği gibi veri seti birden fazla eğitim ve test setine ayrılarak bir öğrenme gerçekleştirilir. Farklı sayıda gerçekleştirilen öğrenmelerin ortalaması alınarak daha geçerli sonuçlar üretilmiş olur. Çapraz doğrulama, veri kümesini eğitim ve test kümelerine ayırarak, modelin performansını ölçer ve aynı zamanda modelin genelleme yeteneğini de test eder. Scikit-learn kütüphanesi, çapraz doğrulama yöntemlerini kullanarak makine öğrenimi modellerinin doğruluğunu değerlendirmek için bir dizi araç sunar. Bu araçlar arasında K-Fold çapraz doğrulama, ShuffleSplit, StratifiedKFold ve GroupKFold yer alır.

- K-Fold çapraz doğrulama, veri kümesini k eşit parçaya ayırarak çalışır ve her bir parçayı sırayla test kümesi olarak kullanır. Kalan k-1 parça ise eğitim kümesi olarak kullanılır. Bu işlem, k kez tekrarlanır ve sonuçlar ortalaması alınarak modelin

performansı ölçülür. Bu yöntem, bir veri kümesinin tamamının kullanılmasını sağladığı için daha güvenilir sonuçlar verir.

-
- ShuffleSplit yöntemi, veri kümesini belirli sayıda rastgele parçaya ayırır ve bu parçaların her birini test kümesi olarak kullanır. Bu yöntem, K-Fold yöntemine göre daha hızlıdır ve daha büyük veri kümeleri için daha uygun olabilir.
-
- StratifiedKFold yöntemi, sınıflandırma problemleri için kullanılır ve her bir katmanda sınıfların oranlarını korumaya çalışır. Bu yöntem, özellikle dengesiz sınıf dağılımlarına sahip veri kümelerinde daha iyi sonuçlar verir.
-
- GroupKFold yöntemi, veri kümesinde belirli gruplar veya kümeler varsa, bu grupları test ve eğitim kümeleri olarak korumaya çalışır. Örneğin, bir kullanıcının birden fazla örneği varsa, bu örnekler aynı test veya eğitim kümesinde yer almayacak şekilde ayarlanır.

Scikit-learn kütüphanesi, bu yöntemlerin yanı sıra başka çapraz doğrulama yöntemleri de sunar. Bu yöntemler, makine öğrenimi modellerinin performansını doğru bir şekilde değerlendirmek için önemli bir araçtır ve modelin genelleme yeteneğini test etmek için kullanılırlar.

Scikit-learn kütüphanesi kullanarak K-Fold çapraz doğrulama yöntemini uygulayabiliriz. Örneğin, boston veri kümesindeki çıktıları tahmin etmek için bir karar ağacı sınıflandırıcısı eğitmek istediğimizi varsayalım. Aşağıdaki kod örneği, bu işlemi gerçekleştirecektir:

```
from sklearn.model_selection import cross_val_score

# K-Fold çapraz doğrulama yöntemini kullanarak modelin doğruluğunu ölç
scores = cross_val_score(model, X, y, cv=5)

# Sonuçları yazdır
print("Doğruluk: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Doğruluk: 0.35 (+/- 0.75)
```

Burada `cross_val_score` fonksiyonu, veri kümesini K-Fold çapraz doğrulama yöntemiyle bölerek modelin doğruluğunu ölçer. Fonksiyon, modelin her bir test seti için doğruluğunu hesaplar ve sonuçları bir dizi olarak döndürür. Daha sonra, bu sonuçların ortalama doğruluğunu ve standart sapmasını hesaplayarak, modelin performansının ne kadar tutarlı olduğunu ölçeriz.

Tahminlerin karşılaştırılması

Makine öğrenmesinde model performansının değerlendirilmesi için farklı metrikler kullanılır. Bu metrikler, modelin tahminlerinin gerçek değerlerle ne kadar iyi eşleştiğini ve ne kadar doğru olduğunu ölçer.

Bir sınıflandırma modeli için kullanılan bazı performans metrikleri şunlardır:

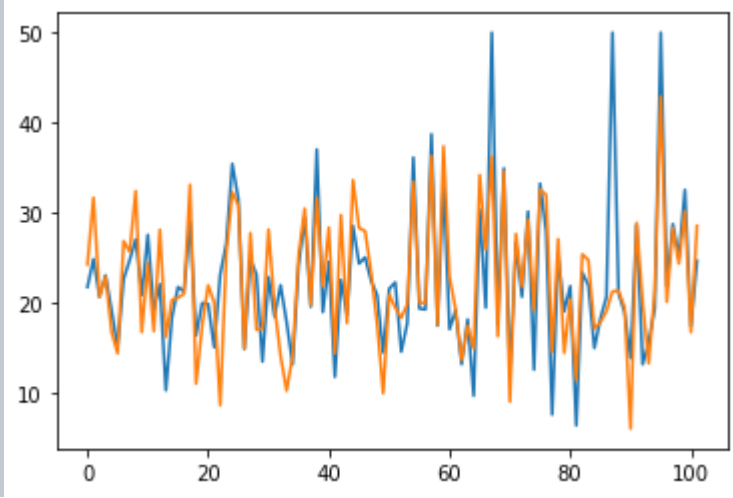
- Confusion Matrix (Karmaşıklık Matrisi): Modelin doğruluğunu değerlendirmek için gerçek ve tahmini sınıfları görsel olarak gösterir.
- Accuracy (Doğruluk): Tahmin edilen sınıfların doğru sınıfların yüzdesi olarak hesaplanır.
- Precision (Hassasiyet): Pozitif olarak tahmin edilen örneklerin gerçekten pozitif olanların yüzdesi olarak hesaplanır.
- Recall (Duyarlılık): Gerçekten pozitif olan örneklerin, pozitif olarak tahmin edilenlerin yüzdesi olarak hesaplanır.
- F1 Score: Precision ve recall metriklerinin harmonik ortalamasıdır. Bir modelin hem hassasiyet hem de duyarlılık açısından iyi performans göstermesi gerektiğinde kullanılır.

Regresyon modelleri için ise kullanılan bazı performans metrikleri şunlardır:

- Mean Squared Error (MSE): Gerçek ve tahmin edilen değerler arasındaki farkların karelerinin ortalamasını hesaplar.
- Root Mean Squared Error (RMSE): MSE'nin karekökünü alarak elde edilir. Gerçek ve tahmin edilen değerler arasındaki farkın ortalamasıdır.
- R-Squared (R^2): Tahmin edilen değerlerin gerçek değerlerin varyansını açıklama yüzdesini hesaplar. 1'e yakın bir R^2 değeri, modelin iyi bir uyum sağladığını gösterir.

Bu metrikler, modelin başarısını objektif bir şekilde değerlendirmemizi sağlar ve geliştirme sürecinde ne yapmamız gerektiği hakkında ipuçları sağlar. Diğer yöntemde ise görsel karşılaştırma, tahminlerin gerçek değerlerle karşılaştırılması için kullanılan bir yöntemdir. Bu yöntemde genellikle çizgi grafikleri veya dağılım grafikleri kullanılır.

```
plt.plot(y_test, label='gerçek değerler')
plt.plot(y_predict, label='tahmin değerleri');
```



Bu kodlar matplotlib kütüphanesi ile birlikte kullanılarak gerçek ve tahmin edilen değerlerin görselleştirilmesini sağlar.

`plt.plot(y_test, label='gerçek değerler')` kodu gerçek değerlerin grafiğini çizerken `plt.plot(y_predict, label='tahmin değerleri')` kodu ise tahmin edilen değerlerin grafiğini çizer. Her iki grafiği karşılaştırmak için aynı grafik üzerinde çizilir ve her bir grafiğin etiketi belirtilir.

Bu şekilde, gerçek ve tahmin edilen değerlerin benzerlikleri veya farklılıkları hakkında görsel bir karşılaştırma yapılabilir.

Sonuç

Bu bölümde, makine öğrenmesinde veri önişleme ve model değerlendirme konuları ele alınmıştır. Veri önişleme aşamaları, verilerin temizlenmesi, ölçeklendirilmesi, aykırı veri analizi, eğitim ve test setlerinin ayrılması ve daha fazlasını içerir. Model değerlendirme, model performansının değerlendirilmesi, tahminlerin gerçek değerlerle karşılaştırılması, görsel karşılaştırma gibi konuları kapsar. Bu bölümde kullanılan örneklerde, sklearn kütüphanesi kullanılarak veri önişleme işlemleri ve model değerlendirme teknikleri uygulanmıştır. Diğer makine öğrenmesi algoritmaları ile ilgili geliştirilecek uygulamalara ilerleyen bölümlerde yer verilecektir.

Kaynaklar

Buitinck, L., Louppe, G., Blondel, M., et al. “API design for machine learning software: experiences from the scikit-learn project”, ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013).