Python'da bir dosya nesnesi oluşturmak için **open** fonksiyonunu kullanırız. Bu fonksiyon, iki ana argüman alır: birincisi dosya yolu (file path), ikincisi ise **mod** (mode). Mod, dosyayı nasıl açmak istediğimizi belirtir. Örneğin, 'r' (read) ile dosyayı okumak için açarız. Dosya nesnesini oluşturduktan sonra, bu nesne üzerinden dosya hakkında bilgi alabiliriz. Örneğin, dosya adını almak için **name** özelliğini kullanabiliriz.

Bir dosyayı açarken, dosya nesnesini kapatmayı unutmamak önemlidir. Bunu yapmak için **close** metodunu kullanırız. Ancak, bu işlemi daha kolay hale getirmek için **with** ifadesini kullanmak iyi bir uygulamadır. **With** ifadesi, dosyayı açtığımızda, içindeki kod bloğu çalıştıktan sonra dosyayı otomatik olarak kapatır. Bu sayede, dosya işlemlerimizi daha güvenli ve düzenli bir şekilde gerçekleştirebiliriz.

Python'un open fonksiyonu ile bir dosya nesnesi (file object) oluşturup, bu dosyaya veri yazabiliyoruz. Dosya oluşturmak için open fonksiyonunu kullanıyoruz. İlk argüman dosya yoludur (file path), bu dosya adı (file name) ve dosya dizininden (directory) oluşur. Yazma modu (mode) olarak 'w' (write) ayarlıyoruz. Son olarak, with ifadesini kullanarak dosyayı açıyoruz; bu, kod bloğu çalıştıktan sonra dosyanın otomatik olarak kapanmasını sağlar.

Örneğin, Example2.txt adında bir dosya oluşturup, içine "This is line A" ve "This is line B" yazabiliriz. Eğer bir liste (list) içindeki her bir öğeyi dosyaya yazmak istersek, yine with komutunu ve open fonksiyonunu kullanarak bir döngü (for loop) ile her öğeyi dosyaya yazabiliriz. Ayrıca, mevcut bir dosyaya ekleme yapmak için 'a' (append) modunu kullanabiliriz. Bu, yeni bir dosya oluşturmak yerine mevcut dosyaya veri ekler.

Pandas, veri analizi (data analysis) için popüler bir kütüphanedir. Pandas'ı kullanmak için önce kütüphaneyi **import** etmemiz gerekiyor. Bunu yapmak için şu komutu yazmalıyız:

import pandas as pd

Bu komut, pandas kütüphanesini **pd** kısaltmasıyla kullanmamıza olanak tanır. Artık pandas'ın sunduğu birçok önceden yazılmış sınıf (class) ve fonksiyona (function) erişimimiz var.

Örneğin, bir **CSV** dosyasını yüklemek için pandas'ın **read_csv** fonksiyonunu kullanabiliriz. CSV, verileri saklamak için yaygın bir dosya türüdür. Aşağıdaki gibi bir kod yazabiliriz:

df = pd.read_csv('dosya_yolu.csv')

Burada, **df** değişkeni (variable) bir **dataframe**'i (veri çerçevesi) temsil eder. Dataframe, satırlar (rows) ve sütunlar (columns) içeren bir veri yapısıdır. Artık verilerimizi bu dataframe üzerinde işleyebiliriz.

Pandas kütüphanesini kullanarak bir veri çerçevesi (data frame) ile nasıl çalışacağımızı öğrendik. Özellikle, bir veri çerçevesindeki bir sütundaki (column) benzersiz (unique) elemanları bulmak için unique metodunu kullanmayı öğrendik. Örneğin, bir müzik veri setinde albümlerin hangi yıllarda yayımlandığını bulmak istiyorsak, Released sütununu kullanarak bu bilgiyi elde edebiliriz.

Bir başka önemli konu ise, belirli bir koşula uyan verileri seçmek. Örneğin, 1979'dan sonra yayımlanan şarkıları içeren yeni bir veri çerçevesi oluşturmak istiyorsak, Released sütununu kullanarak bu koşulu belirleyebiliriz. Bu işlemi tek bir satırda gerçekleştirebiliriz. Pandas'ta, bu tür koşulları belirlemek için karşılaştırma operatörlerini (inequality operators)

kullanıyoruz. Sonuç olarak, bu koşula uyan verileri içeren yeni bir veri çerçevesi oluşturabiliriz ve bunu to_csv metodu ile bir CSV dosyası olarak kaydedebiliriz.

NumPy dizisi (ndarray) oluşturmak için önce NumPy kütüphanesini içe aktarmamız gerekiyor. Bir Python listesi, verileri saklamak ve erişmek için bir konteynerdir. Her bir eleman bir indeks ile ilişkilidir. NumPy dizisi, genellikle sabit boyutlu ve her elemanı aynı türdendir. Örneğin, bir listeyi NumPy dizisine dönüştürmek için şu şekilde yapabiliriz:

```
import numpy as np

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

Bu kodda, np.array() fonksiyonu ile bir NumPy dizisi oluşturuyoruz. Diziye erişmek için de indeks kullanabiliriz. Örneğin, a[0] ifadesi dizinin ilk elemanını döndürür. NumPy dizileri, aynı türdeki verileri sakladıkları için, dtype (data type) özelliği ile dizinin elemanlarının veri türünü öğrenebiliriz.

Numpy, çok boyutlu diziler oluşturmanıza olanak tanır. Bu bölümde, **2D arrays** üzerinde duracağız. Düşünün ki, bir **matrix** (matris) gibi, her bir **nested list** (iç içe liste) bir satırı temsil ediyor. Örneğin, üç iç içe liste içeren bir liste düşünün. Bu listeyi Numpy dizisine dönüştürdüğümüzde, her iç içe liste bir satır olarak görselleştirilebilir. Numpy'de dizinin boyutunu öğrenmek için **ndim** (boyut sayısı) ve **shape** (şekil) gibi özellikleri kullanabiliriz. **Shape** özelliği, dizinin kaç satır ve kaç sütun içerdiğini gösterir.

Örneğin, bir dizi tanımlamak için şu şekilde yazabiliriz:

```
import numpy as np


# 2D array oluşturma

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

Burada, array değişkeni 3 satır ve 3 sütun içeren bir **2D array** oluşturur. **Indexing** (indeksleme) ile dizinin elemanlarına erişebiliriz. Örneğin, array[1][2] ifadesi, ikinci satır ve üçüncü sütundaki değeri (yani 6) döndürür.