# Büyük Veri Analizi

## Ders 5

Ali Mertcan KOSE Ph.D.
amertcankose@ticaret.edu.tr

İstanbul Ticaret Üniversitesi

# Giriş

Önceki bölümde, iş gereksinimlerinin nasıl belirlenip anlaşılacağı, bunları çözmeye yönelik bir yaklaşım ve veri hatlarının nasıl oluşturulacağı ve analizlerin nasıl gerçekleştirileceği gibi konuları içeren, veri bilimi perspektifinden oldukça yapılandırılmış bir yaklaşımla bir iş probleminin nasıl tanımlanacağını öğrendik.

Bu bölümde, günümüzde sektör genelinde ve akademisyenler tarafından karşılaşılan büyük bir zorluk olan hesaplamalı çalışmaların ve araştırma uygulamalarının tekrarlanabilirliğini inceleyeceğiz; özellikle de verilerin çoğuna, eksiksiz veri kümelerine ve ilgili iş akışına tam olarak erişilemeyen veri bilimi çalışmalarında. Günümüzde çoğu araştırma ve teknik makale, örnek veriler üzerinde kullanılan yaklaşım, kullanılan metodolojiye kısa bir değinme ve bir çözüme yönelik teorik bir yaklaşımla sonuçlanmaktadır. Bu çalışmaların çoğunda ayrıntılı hesaplamalar ve adım adım yaklaşımlar eksiktir. Bu, okuyan herkesin gerçekleştirilen aynı çalışmayı tekrarlayabilmesi için çok sınırlı bir bilgi birikimidir.

# Giriş

Bu, kodun kolayca tekrar üretilebilmesinin önemli olduğu tekrarlanabilir kodlamanın temel amacıdır. Genel olarak, ayrıntılı yorumlama için metin öğeleri içerebilen not defterlerinde gelişmeler olmuştur; bu, tekrar üretme sürecini iyileştirir. Jupyter'ın bir not defteri olarak veri bilimi ve araştırma topluluklarında ilgi gördüğü nokta burasıdır. Jupyter, Python, Spark ve R dahil olmak üzere düzinelerce programlama dilinde etkileşimli bilgi işlem için açık standartlara ve hizmetlere sahip açık kaynaklı bir yazılım olma amacıyla geliştirilmiştir.

# Jupyter Not Defterleriyle Yeniden Üretilebilirlik

Hesaplamalı tekrarlanabilirliğin ne anlama geldiğini öğrenerek başlayalım. Araştırmalar, çözümler, prototipler ve hatta geliştirilen basit bir algoritmanın, çözümü geliştirmek için kullanılan orijinal kaynak koduna erişim sağlandığı takdirde tekrarlanabilir olduğu söylenir ve ilgili herhangi bir yazılımı oluşturmak için kullanılan veriler aynı sonuçları üretebilmelidir. Ancak günümüzde bilim camiası, daha önce meslektaşları tarafından geliştirilen çalışmaları tekrarlamada bazı zorluklarla karşılaşmaktadır. Bu, esas olarak dokümantasyon eksikliğinden ve süreç iş akışlarını anlamadaki zorluktan kaynaklanmaktadır.

# Jupyter Not Defterleriyle Yeniden Üretilebilirlik

Dokümantasyon eksikliğinin etkisi, yaklaşımın kod seviyesine kadar her seviyede görülebilir. Jupyter, bu süreci daha iyi yeniden üretilebilirlik ve geliştirilen kodun yeniden kullanımı için en iyi araçlardan biridir. Bu, yalnızca her bir kod satırının veya kod parçasının ne işe yaradığını anlamakla kalmaz, aynı zamanda verileri anlamayı ve görselleştirmeyi de içerir.

# Jupyter Not Defterleriyle Yeniden Üretilebilirlik

Şimdi, Jupyter not defterini kullanarak etkili hesaplama tekrarlanabilirliğini nasıl koruyabileceğimize bakalım. Aşağıdaki ipuçları, Python'da bir Jupyter not defteri kullanarak tekrarlanabilirliği sağlamanın genel yollarıdır:

- İş sorununa ayrıntılı bir giriş sağlayın
- Yaklaşımı ve iş akışını belgelendirin
- Veri kanallarını açıklayın
- Bağımlılıkları açıklayın
- Kaynak kod sürüm kontrolünü kullanın
- Süreci modülerleştirin

Aşağıdaki bölümlerde, daha önce bahsedilen konuları kısaca inceleyip tartışalım.

# İşletme Problemine Giriş

Jupyter not defterini kullanmanın temel avantajlarından biri, bir iş akışı oluşturmak için kodla birlikte metinsel içerik de içermesidir. Tespit ettiğimiz iş problemine iyi bir girişle başlamalı ve problemin özünü sağlamak için Jupyter not defterinde bunun bir özeti belgelenmelidir. Belirlenen iş problemini veri bilimi perspektifinden ele alan, bu analizi neden yapmamız gerektiğini veya sürecin amacının ne olduğunu açıklayan bir problem ifadesi içermelidir.

# Yaklaşım ve İş Akışlarının Belgelenmesi

Veri biliminde, hesaplamalı çalışmalarda, örneğin yürütülen araştırmalar, kullanılan algoritma türleri ve ayarlanacak parametre değişiklikleri gibi konularda çok fazla ileri geri gidilebilir. Yaklaşımdaki değişiklikler tamamlandıktan sonra, işin tekrarlanmasını önlemek için bu değişikliklerin belgelenmesi gerekir. Yaklaşımın ve iş akışlarının belgelenmesi, bir sürecin kurulmasına yardımcı olur. Geliştirme sırasında koda yorum eklemek şarttır ve bu, yorum eklemek için son ana veya sonuçlara kadar beklemek yerine sürekli bir uygulama olmalıdır. Sürecin sonunda, ayrıntıları unutmuş olabilirsiniz ve bu, harcanan çabanın yanlış hesaplanmasına neden olabilir. Jupyter not defterini iyi bir dokümantasyonla sürdürmenin avantajları şunlardır:

# Yaklaşım ve İş Akışlarının Belgelenmesi

- Geliştirme çabasının takibi
- Her işlem için yorumlar içeren, kendini açıklayan kod
- Kod iş akışının ve her adımın sonuçlarının daha iyi anlaşılması
- Belirli görevler için önceki kod parçacıklarının bulunmasını kolaylaştırarak ileri geri çalışmalardan kaçınma
- Kodun tekrar tekrar kullanımını anlayarak aynı işi tekrar yapmaktan kaçınma
- Bilgi aktarımının kolaylığı

# Veri Boru Hattını Açıklamak

Sorunun tanımlanması ve nicelendirilmesi için gereken veriler, veritabanları, eski sistemler, gerçek zamanlı veri kaynakları vb. gibi birden fazla veri kaynağından üretilebilir. Bu süreçte yer alan veri bilimcisi, gerekli verileri çıkarmak, toplamak ve daha ileri analizler için analitik araçlara aktarmak üzere müşterinin veri yönetimi ekipleriyle yakın bir şekilde çalışır ve bu verileri elde etmek için güçlü bir veri hattı oluşturur.

Veri kaynaklarını ayrıntılı olarak belgelemek (önceki bölümde ele alınmıştır) önemlidir; böylece, dikkate alınan değişkenleri, neden dikkate alındıklarını, ne tür verilere sahip olduğumuzu (yapılandırılmış veya yapılandırılmamış) ve sahip olduğumuz veri türünü (yani, bir zaman serisi, çok değişkenli veya görüntü, metin, konuşma vb. gibi ham kaynaklardan ön işleme tabi tutulup oluşturulması gereken verilerimiz olup olmadığını) açıklayan bir veri sözlüğü tutulabilir.

# Bağımlılıkları Açıklayın

Bağımlılıklar, bir araçta bulunan paketler ve kütüphanelerdir. Örneğin, Python'da görüntü modelleme için bir kütüphane olan OpenCV'yi (https://docs.opencv.org/3.0-beta/doc/py/underline{ }tutorials/py_tutorials.html) veya derin öğrenme modellemesi için TensorFlow gibi bir API'yi kullanabilirsiniz. İşte başka bir örnek: Python'da görselleştirme için Matplotlib'i (https://matplotlib.org/) kullanıyorsanız, Matplotlib bağımlılıkların bir parçası olabilir. Diğer yandan, bağımlılıklar bir analiz için gereken donanım ve yazılım özelliklerini içerebilir. Bağımlılıklarınızı, paket/kütüphane sürümleri de dahil olmak üzere, tüm ilgili bağımlılıkları (pandas, NumPy vb. için bağımlılıklar hakkındaki önceki bölümlerde ele alınmıştır) listelemek için Conda ortamı gibi bir araç kullanarak en başından itibaren açıkça yönetebilirsiniz.

# Kaynak Kod Sürüm Kontrolünü Kullanma

Sürüm kontrolü, kod içeren her türlü hesaplamalı faaliyette önemli bir unsurdur. Kod geliştirilirken hatalar veya eksiklikler ortaya çıkar. Kodun önceki sürümleri mevcutsa, hatanın ne zaman tespit edildiğini, ne zaman çözüldüğünü ve ne kadar emek harcandığını tam olarak belirleyebiliriz. Bu, sürüm kontrolüyle mümkündür. Bazen ölçeklenebilirlik, performans veya başka nedenlerle eski sürümlere geri dönmeniz gerekebilir. Kaynak kod sürüm kontrol araçlarını kullanarak, kodun önceki sürümlerine her zaman kolayca erişebilirsiniz.

# Sürecin Modülerleştirilmesi

Tekrarlayan görevleri yönetmek, kodu korumak ve hata ayıklamak için yinelenen kodlardan kaçınmak etkili bir uygulamadır. Bunu verimli bir şekilde gerçekleştirmek için süreci modüler hale getirmelisiniz.

Bunu ayrıntılı olarak anlayalım. Diyelim ki bir görevi tamamlamak için kod geliştirdiğiniz bir dizi veri işleme süreci gerçekleştiriyorsunuz. Şimdi, aynı kodu kodun sonraki bir bölümünde kullanmanız gerektiğini varsayalım; aynı adımları tekrar tekrar eklemeniz, kopyalamanız veya çalıştırmanız gerekiyor ki bu da tekrarlayan bir görevdir. Giriş verileri ve değişken adları değişebilir. Bunu halletmek için, önceki adımları bir veri kümesi veya bir değişken için bir fonksiyon olarak yazabilir ve bu fonksiyonların tümünü ayrı bir modül olarak kaydedebilirsiniz. Buna bir fonksiyon dosyası (örneğin, bir Python dosyası olan functions.py) diyebilirsiniz. Bir sonraki bölümde, özellikle tekrarlanabilir bir şekilde verimli bir veri hattı toplama ve oluşturma konusunda buna daha ayrıntılı olarak bakacağız.

# Verilerin Tekrarlanabilir Bir Şekilde Toplanması

Sorun tanımlandıktan sonra, bir analiz görevinin ilk adımı veri toplamaktır. Veriler birden fazla kaynaktan elde edilebilir: veritabanları, eski sistemler, gerçek zamanlı veriler, harici veriler vb. Veri kaynakları ve verilerin modele nasıl aktarılacağı belgelenmelidir. Jupyter not defterinde Markdown ve kod bloğu işlevlerinin nasıl kullanılacağını anlayalım. Markdown hücreleri kullanılarak Jupyter not defterlerine metin eklenebilir. Bu metinler, herhangi bir metin düzenleyicide olduğu gibi kalın veya italik olarak değiştirilebilir. Hücre türünü Markdown olarak değiştirmek için Hücre menüsünü kullanabilirsiniz. Jupyter'da Markdown ve kod hücrelerindeki çeşitli işlevleri nasıl kullanabileceğinize bakacağız.

# Markdown ve Kod Hücrelerindeki İşlevler

- Jupyter'da Markdown: Jupyter'da Markdown seçeneğini belirlemek için açılır menüden Widget'lar ve Markdown'a tıklayın:
- Jupyter'da Başlık: Jupyter not defterinde iki tür başlık bulunur. Başlık ekleme sözdizimi, HTML'dekine benzer şekilde

  ve

  etiketleri kullanılarak yapılır:
- Jupyter'da Metin: Metni olduğu gibi eklemek için, metne herhangi bir etiket eklemiyoruz:

# Markdown ve Kod Hücrelerindeki İşlevler

- Jupyter'da Kalın: Metnin Jupyter not defterinde kalın görünmesi için metnin başına ve sonuna iki yıldız (**) ekleyin, örneğin **Kalın**:
- Jupyter'da İtalik: Jupyter not defterinde metnin italik görünmesi için metnin başına ve sonuna bir yıldız (*) ekleyin:

# Markdown'da İş Probleminin Açıklanması

Projenin amacını anlamak için iş problemine kısa bir giriş yapın. İş problemi tanımı, problem ifadesinin bir özetidir ve problemin bir veri bilimi algoritması kullanılarak nasıl çözülebileceğini içerir:

# Veri Kaynağına Ayrıntılı Bir Giriş Sağlama

Veri lisansının tekrarlanabilirlik ve daha fazla çalışma için anlaşılması amacıyla veri kaynağının doğru bir şekilde belgelenmesi gerekir.

# Markdown'daki Veri Niteliklerini Açıklayın

Verileri nitelik düzeyinde anlamak için bir veri sözlüğünün tutulması gerekir. Bu, niteliğin ne tür bir veri olduğunu tanımlamayı içerebilir:

Verileri öznitelik düzeyinde anlamak için info ve describe gibi fonksiyonları kullanabiliriz; ancak pandas_profiling, tek bir fonksiyonda çok sayıda tanımlayıcı bilgi sağlayan bir kütüphanedir ve bu kütüphaneden aşağıdaki bilgileri çıkarabiliriz:

# Markdown'daki Veri Niteliklerini Açıklayın

Veri Çerçevesi düzeyinde, yani dikkate alınan tüm sütun ve satırları içeren genel veriler için:

- Değişken sayısı
- Gözlem sayısı
- Toplam kayıp (%)
- Bellekteki toplam boyut
- Bellekteki ortalama kayıt boyutu
- Korelasyon matrisi
- Örnek veriler

# Markdown'daki Veri Niteliklerini Açıklayın

Belirli bir sütun için olan öznitelik düzeyinde, özellikler aşağıdaki gibidir:

- Ayrık sayım
- Benzersiz (%)
- Eksik (%)
- Eksik (n)
- Sonsuz (%)
- Sonsuz (n)
- Dağılım için histogram
- Uç değerler

# Veri Yeniden Üretilebilirliğini Gerçekleştirme

Bu alıştırmanın amacı, veri anlayışı açısından yüksek tekrarlanabilirliğe sahip kod geliştirmeyi öğrenmektir. Bu bağlantıdan alınan UCI Bankası ve Pazarlama veri setini kullanacağız:

https://raw.githubusercontent.com/mertcank1/BDA/refs/heads/main/bank.csv

# Veri Yeniden Üretilebilirliğini Gerçekleştirme

Veri tekrarlanabilirliğini sağlamak için aşağıdaki adımları uygulayalım:

1. Başlıklar ekleyin ve not defterine işaretleme kullanarak iş probleminden bahsedin:
2. Gerekli kütüphaneleri Jupyter not defterine aktarın:

```
import numpy as np
import pandas as pd
import time
import re
import os
import pandas_profiling
```

# Veri Yeniden Üretilebilirliğini Gerçekleştirme

3. Şimdi aşağıdaki komutta gösterildiği gibi çalışma dizinini ayarlayın:

```
os.chdir("/Users/svk/Desktop/packt_exercises")
```

4. Veri kümesinden pandas'ın read_csv fonksiyonunu kullanarak giriş veri kümesini df olarak içe aktarın ve okuyun:

```
df = pd.read_csv('bank.csv', sep=';')
```

# Veri Yeniden Üretilebilirliğini Gerçekleştirme

5. Şimdi head fonksiyonunu kullanarak veri setinin ilk beş satırını görüntüleyin:

```
df.head(5)
```

6. Jupyter not defterine Veri Sözlüğü ve Veri Anlama bölümlerini ekleyin:

7. `Veri özelliklerini anlamak için, açıklayıcı bilgileri oluşturmak üzere pandas profillemeyi kullanın:`

```
pandas_profiling.ProfileReport(df)
```

# Kod Uygulamaları ve Standartları

Kod tekrarlanabilirliği için, bir dizi uygulama ve standartla kod yazmak önemlidir; sürecin iş akışını adım adım açıklayıcı bir şekilde açıklamak da öyle. Bu, yalnızca Jupyter ile değil, kullanabileceğiniz tüm kodlama araçları için evrensel olarak geçerlidir. Bazı kodlama uygulamalarına ve standartlarına kesinlikle uyulmalıdır ve bunlardan birkaçı bir sonraki bölümde ele alınacaktır.

# Çevre Dokümantasyonu

Kurulum amacıyla, gerekli paketleri ve kütüphaneleri kurmak için bir kod parçası tutmalısınız. Aşağıdaki uygulamalar kodun yeniden üretilebilirliğine yardımcı olur:

- Kütüphaneler/paketler için kullanılan sürümleri ekleyin.
- Kullanılan paketlerin/kütüphanelerin orijinal sürümünü indirin ve yeni bir kurulumda kurulum için paketleri dahili olarak çağırın.
- Bağımlılıkları otomatik olarak kuran bir betikte çalıştırarak etkili uygulama.

# Yorumlarla Okunabilir Kod Yazma

Kod yorumlama önemli bir husustur. Jupyter'da bulunan işaretleme seçeneğinin yanı sıra, her kod parçası için yorum eklemeliyiz. Bazen kodda, hemen kullanılmayabilecek ancak sonraki adımlarda gerekli olacak şekilde değişiklikler yaparız. Örneğin, bir sonraki adım için hemen kullanılmayabilecek ancak sonraki adımlarda kullanılabilecek bir nesne oluşturabiliriz. Bu, yeni bir kullanıcının akışı anlaması açısından kafa karışıklığına neden olabilir. Bu tür ayrıntıları yorumlamak çok önemlidir.

# İş Akışlarının Etkili Segmentasyonu

Kod geliştirirken, nihai sonuçlara ulaşmak için tasarladığınız adımlar vardır. Her adım bir sürecin parçası olabilir. Örneğin, veri okuma, veri anlama, çeşitli dönüşümler gerçekleştirme veya bir model oluşturma. Bu adımların her biri, çeşitli nedenlerle açıkça ayrılmalıdır; birincisi, her aşamanın nasıl gerçekleştirildiğine dair kod okunabilirliği ve ikincisi, her aşamada sonucun nasıl üretildiği. Örneğin, burada iki grup aktiviteye bakıyoruz. Birincisi, normalleştirilmesi gereken sütunları belirlemek için döngünün oluşturulduğu, ikincisi ise önceki çıktıyı kullanarak normalleştirilmesi gerekmeyen sütunların oluşturulduğu:

# İş Akışı Belgeleri

Ürünler ve çözümler geliştirilirken, çoğunlukla bir deneme ortamında geliştirilir, izlenir, dağıtılır ve test edilir. Yeni bir ortamda sorunsuz bir dağıtım süreci sağlamak için, teknik ve teknik olmayan kullanıcılar için yeterli destek belgeleri sağlamalıyız. İş akışı belgeleri, gereksinimler ve tasarım belgeleri, ürün belgeleri, metodoloji belgeleri, kurulum kılavuzları, yazılım kullanıcı kılavuzları, donanım ve yazılım gereksinimleri, sorun giderme yönetimi ve test belgelerini içerir. Bunlar çoğunlukla bir ürün veya çözüm geliştirme için gereklidir. Bir müşteriye/kullanıcıya bir sürü kod verip çalıştırmalarını isteyemeyiz. İş akışı belgeleri, kodun yeniden üretilebilirliği için son derece önemli olan bir müşteri/kullanıcı ortamındaki dağıtım ve entegrasyon aşamalarında yardımcı olur. Veri bilimi proje belgeleri üst düzeyde iki bölüme ayrılabilir:

- Ürün belgeleri
- Metodoloji belgeleri

# Yüksek Üretilebilirliğe Sahip Eksik Değer Ön İşleme

Bu alıştırmanın amacı, eksik değer işleme ön işlemesi açısından yüksek tekrarlanabilirliğe sahip kodun nasıl geliştirileceğini öğrenmektir.

Eksik değer ön işleme tekrarlanabilirliğini bulmak için aşağıdaki adımları uygulayın:

1. Gerekli kütüphaneleri ve paketleri burada gösterildiği gibi Jupyter not defterine aktarın:

```
import numpy as np
import pandas as pd
import collections
import random
```

# Yüksek Üretilebilirliğe Sahip Eksik Değer Ön İşleme

2. Aşağıda gösterildiği gibi istediğiniz çalışma dizinini ayarlayın:

```
os.chdir("/Users/svk/Desktop/packt_exercises")
```

3. Veri setini back.csv dosyasından, read_csv fonksiyonunu kullanarak Spark nesnesine aktarın.Burada gösterildiği gibi:

```
df = pd.read_csv('bank.csv', sep=';')
```

4. Şimdi, head fonksiyonunu kullanarak veri kümesinin ilk beş satırını görüntüleyin:

# Yüksek Üretilebilirliğe Sahip Eksik Değer Ön İşleme

Veri setinde eksik değer bulunmadığından, veri setine bazı eksik değerleri eklememiz gerekiyor.

```
df.head(5)
```

5. İlk olarak, döngü parametrelerini burada gösterildiği gibi ayarlayın:

```
replaced = collections.defaultdict(set)
ix = [(row, col) for row in range(df.shape[0]) for
col in range(df.  shape[1])]
random.shuffle(ix)
to_replace = int(round(.1*len(ix)))
```

# Yüksek Üretilebilirliğe Sahip Eksik Değer Ön İşleme

6. Eksik değerleri üretmek için bir for döngüsü oluşturun:

```
for row, col in ix:
if len(replaced[row]) < df.shape[1] - 1:
df.iloc[row, col] = np.nan
to_replace -= 1
replaced[row].add(col)
if to_replace == 0:
break
```

# Yüksek Üretilebilirliğe Sahip Eksik Değer Ön İşleme

7. Verilerdeki eksik değerleri belirlemek için her sütunun eksik değerlerine bakarak aşağıdaki komutu kullanın:

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

8. Çeyreklik Aralığı (ÇA) aralığını tanımlayın ve aykırı değerleri belirlemek için bunları veri setine uygulayın:

```
num = df._get_numeric_data() Q1 = num.quantile(0.25)
Q3 = num.quantile(0.75) IQR = Q3 - Q1 print(num < (Q1
- 1.5 * IQR)) print(num > (Q3 + 1.5 * IQR))
```

# Tekrarı Önlemek

Hepimiz kodun tekrarlanmasının veya çoğaltılmasının iyi bir uygulama olmadığını biliyoruz. Hatalarla başa çıkmak zorlaşır ve kodun uzunluğu artar. Aynı kodun farklı sürümleri, hangi sürümün doğru olduğunu anlamak açısından bir noktadan sonra zorluklara yol açabilir. Hata ayıklama için, bir konumdaki değişikliğin koda yansıtılması gerekir. Kötü uygulamalardan kaçınmak ve üst düzey kod yazıp sürdürmek için, aşağıdaki bölümlerde bazı en iyi uygulamaları öğrenelim.

# Kodu Optimize Etmek İçin Fonksiyonları ve Döngüleri

Bir fonksiyon, tek bir girdiden tek veya birden fazla çıktıya kadar bir dizi adım gerektiren bir görevi sınırlar ve döngüler, farklı bir örnek veya alt küme veri kümesi için aynı kod bloğunda tekrarlayan görevler için kullanılır. Fonksiyonlar tek bir değişken, birden fazla değişken, bir Veri Çerçevesi veya birden fazla parametre girişi kümesi için yazılabilir. Örneğin, bir Veri Çerçevesi veya matristeki yalnızca sayısal değişkenler için bir tür dönüşüm gerçekleştirmeniz gerektiğini varsayalım. Bir fonksiyon tek bir değişken için yazılabilir ve tüm sayısal sütunlara uygulanabilir veya fonksiyonun sayısal değişkenler kümesini tanımlayıp çıktıyı oluşturmak için uyguladığı bir Veri Çerçevesi için yazılabilir. Bir fonksiyon yazıldıktan sonra, devam eden koddaki herhangi bir benzer uygulamaya uygulanabilir. Bu, tekrarlanan işi azaltır.

# Kodu Optimize Etmek İçin Fonksiyonları ve Döngüleri

Bir fonksiyon yazarken dikkate alınması gereken zorluklar şunlardır:

- Dahili parametre değişiklikleri: Giriş parametresinde bir görevden diğerine değişiklikler olabilir. Bu yaygın bir sorundur. Bunu ele almak için, bir fonksiyon için girdileri tanımlarken fonksiyon girdilerindeki dinamik değişkenlerden veya nesnelerden bahsedebilirsiniz.

# Kodu Optimize Etmek İçin Fonksiyonları ve Döngüleri

- Gelecekteki bir görev için hesaplama sürecindeki varyasyonlar: Herhangi bir varyasyonun yakalanması gerekirse çok fazla değişiklik gerektirmeyecek dahili fonksiyonlara sahip bir fonksiyon yazın. Bu şekilde, fonksiyonu yeni bir görev türü için yeniden yazmak kolay olacaktır.
- Fonksiyonlarda döngülerden kaçınma: Bir işlemin verilerin birçok alt kümesinde satır bazında gerçekleştirilmesi gerekiyorsa, fonksiyonlar her döngüde doğrudan uygulanabilir. Bu şekilde, fonksiyonunuz aynı veriler üzerindeki tekrarlayan kod bloklarıyla kısıtlanmaz.

# Kodu Optimize Etmek İçin Fonksiyonları ve Döngüleri

- Veri türü değişikliklerini ele alma: Bir fonksiyondaki dönüş nesnesi farklı görevler için farklı olabilir. Göreve bağlı olarak, dönüş nesnesi gerektiği gibi diğer veri sınıflarına veya veri türlerine dönüştürülebilir. Ancak, giriş veri sınıfları veya veri türleri görevden göreve değişebilir. Bunu başarmak için, bir fonksiyonun girdilerini anlamak amacıyla yorumları açıkça belirtmeniz gerekir.
- Optimize edilmiş fonksiyonlar yazmak: Diziler, döngüler veya fonksiyonlar gibi tekrarlayan görevler söz konusu olduğunda etkilidir. Python'da NumPy dizilerini kullanmak