Göker Güner: Evet. Şimdi arkadaşlar, öncelikle "iyi görünmüyor" diyeceksiniz. Şöyle bir tur daha büyütebiliyor muyum? Böyle birazcık daha okunur oldu sanıyorum. Şimdi bugün iki bölüm olacak ders. Birisi Python, birazcık Python konuşacağız, epeyce bir Python konuşacağız. Sonra bu data tarafındaki kütüphanelere değineceğim kısaca. Neden "Python anlatacağım" değil de "Python konuşacağız" diyorum, ondan bahsedeyim birazcık. Esasen artık günümüzdeki bu generative AI teknolojileri ile beraber Python öğrenmek için herhangi bir canlı derse bile ihtiyacınız yok bana sorarsanız. Yani ChatGPT'ye "ben Python öğrenmek istiyorum, konu başlıklarını listele" deyip, sonra konu başlıkları ile ilgili teker teker örnek istediğinizde... Ama insan yine de birazcık kulak aşinalığı arıyor. O yüzden Python'ı öğrenirken, özellikle yapay zekaya yönelik öğrenirken nelere dikkat etmemiz lazım, neyi nereden öğrenirsiniz, ne önemli, neyi birazcık daha arka plana atarsınız gibi noktalara da değinip bir kulak aşinalığı yaratmak derdindeyim.

Bu yayınlar biliyorsunuz YouTube'da kalıcı olacak. Derdimiz aslında o. Bunu bu **Colab**'ı sizinle Lodos'tan paylaşacağım. Hem Lodos'ta hem WhatsApp'ta olan arkadaşlarımız zaten Lodos'tan alıp WhatsApp'taki diğer grubumuza paylaşacaklardır. Bu yayından sonra bu notebook'u alıp karşınıza açıp, buradaki örnekleri teker teker kendi ortamlarınızda deneyip, bir yandan yayını da açarsınız. Bu şekilde kalıcı bir öğrenme yolculuğu olmuş olur. Neden kısaca data kütüphanelerine? Çünkü bootcamp süresince zaten neredeyse her ders o kütüphaneleri kullanarak bir şeyler yapıyor olacaksınız. Dolayısıyla hem sürenizi çok yememek hem de artık iyice böyle sıkmamak için orayı birazcık daha kısa geçmeyi tercih ettim. Ama temel kütüphanelerin temel özelliklerine değinip, en sonda da gerçekten çok basit, çok minik bir Machine Learning projesiyle bir Machine Learning projesinin içinde aslında temelde hangi komponentler var, bunlardan bahsediyor olacağım.

Python'ı temelden orta seviye olarak isimlendirdim. İleri seviye olarak isimlendirmek birazcık iddialı olur. En temelinde, daha önce hiç yazılım dili ile uğraşmayan ya da başka dillerle de uğraştıysanız bildiğiniz şey değişkenler. Öncelikle en temel yapı taşımız değişkenler. Basitçe nedir değişkenler? Sayıları, metinleri saklamak için kullandığımız isimlendirilmiş kutular gibi. Başka dillerde eğer yazılım geliştirdiyseniz, daha önce okulda C, C++, C#, Java gibi dersler aldıysanız Python'da şöyle bir farklılık var, aynı zamanda bu bir kolaylık. Python'da bir değişkenin tipini önceden belirtmeniz gerekmiyor. Bir değişkene bir değer atadığınızda Python o değerin tipini otomatik olarak anlıyor. "Dynamically typed" deniyor buna. Yine kodumuza açıklama eklemek için de # (diyez) işaretini kullanıyoruz. # ile başlayan satırları Python görmezden geliyor. Bunlara yorum satırı deniyor. Burada gördüğünüz gibi bu bir yorum satırı. **print** fonksiyonu ekrana Merhaba Python yazdıralım. İşte bu yazılım dilleri geliştirildiğinde ilk programcının ekrana yazdırdığı ilk şey zannedersem Hello, world!'müş seneler seneler evvel. Ondan dolayı gelen böyle bir gelenek var. Bu notebookun içerisinde bu arada bu değişkenler nedir, zaten yeteri kadar değineceğimiz için burada ayrıca her şeyi böyle örneklemedim. Zaten yeri geldiğinde değişken nedir onu konuşuyor olacağız. Genelde de bölümleri birbirine bağlamak için her bölümde bir sonraki bölüme ya da sonraki bölümlere referans vererek ilerledim. Bir şeyi de açıklığa kavuşturayım. Ben burada anlatırken, özellikle ilk defa dinleyenleriniz varsa, birçok şey çok karışık gelecek, farkındayım. Ama bootcamp'in amacı zaten bu. Hem bu YouTube yayını, hem bu notebook'u size verip sizi bu öğrenme ortamıyla baş başa bıraktıktan sonra kendiniz çalışarak, takıldığınız yerde bize Lodos üzerinden sorular sorarak öğrenme yolculuğunuzu pekiştiriyor olacaksınız. O yüzden de ondan da endişe etmeyelim.

Python'da bir takım veri tipleri var. **integer**, **float** ve **complex** bizim sayısal veri tiplerimiz ki **complex**'i genelde çok kullanmıyoruz. Ben çok denk gelmedim kullananlara. Ama akademi tarafında ilerleyecek olanlarınız varsa, belki orada birazcık daha kullanılıyordur diye düşünüyorum.

Soru: Göker, bir böleceğim seni ama fontu biraz büyütebilir misin diye rica etmişler.

Göker Güner: Hemen büyütelim, büyütelim. Bir tık daha büyüttüm. Yeterli olmadıysa yazan olursa ona göre şey yaparız. Şu an gayet iyi görüyorum. Yayında da iyi görünüyor gibi. Okey, o zaman böyle devam ediyorum.

Daha önce benim verdiğim Python eğitimlerine denk gelen varsa aslında **string**'ler üzerine daha fazla duruyordum ama bunun da birazcık insanları boğduğunu, diğer özelliklerde eksik bıraktığını fark ettim. O yüzden bu notebook'ta çok fazla **string** anlatımı yok ama yapay zekada **string**'lerle çok uğraşıyorsunuz. En basitinden bir takım prosesleri log'larken, yani o proseslerin çıktılarını görüntülerken **string**'leri formatlıyorsunuz. **String** nedir, nasıl formatlanır, anlatacağım tabii. En çok kullandığınız veri tiplerinden birisi. Zaten kendinize yapay zeka alt dalı seçtiğinizde, metin verileri ile uğraşmayı tercih ederseniz, text'ler olabilir, işte PDF'ler olabilir vesaire, **string**'leri sıkça kullanıyor olacaksınız. Mantıksal tip dediğimiz **boolean** tipi var. Onlar da sadece **True** ve **False** değerlerini alıyorlar diyelim. Devam edelim. Burada bir gösterimimiz var. Tam sayı nedir, virgüllü sayı nedir, karmaşık sayı nedir, kısaca birer örnekle açıklamak istedim. Bunlar çok basit konseptler olduğu için çok üzerinde durmayacağım. Sadece burada şunun üzerinde durmak istiyorum. Şurada yaptığımız **print** gösterimi arkadaşlar, "**f-string**" diye geçiyor. **F-string** nedir? Metin içine süslü parantezle değişken yerleştirmenin modern ve kolay yoludur. Ben burada girdiğim tam sayıyı... Ki bunu birazcık daha karmaşık düşünün. Başka bir takım fonksiyonlar var bir yerlerde, bir takım işlemler var. Siz o işlemleri alıp kendi dosyanızda kullanmak istiyorsunuz, bir değişkene attınız. Sonra da bunu "acaba bu işlemlerin sonucunda ne gelmişti?" diye ekrana yazdırmak istiyorsunuz. Bu **f-string** size bunu sağlıyor. Çok kullanılan **string** formatlama yöntemlerinden biridir. Bu şekilde değişkenleri direkt olarak burada görüldüğü üzere ekrana yazdırabiliyorsunuz.

String ile ilgili yine kullanışlı olabilecek bir diğer nokta, tek tırnak, çift tırnak konusu. Hem tek tırnakları hem çift tırnakları Python birer metin olarak algılar. Yine bu C dil ailesinde kod yazdıysanız, Java, Go, C# gibi dilleri kullandıysanız, tek bir karakter için oralarda **char** denen, ya da "kar" diye mi okunur, karakterin kısaltılması, ayrı bir veri tipi var. Python'da bu yok. Dolayısıyla siz tek bir harf de girdiğiniz zaman bunu bir **string** olarak kabul ediyoruz arkadaşlar. Python'da aynı şekilde bu metinleri **f-string** olarak ekrana bastırıyoruz. Farklı tırnakları bir arada kullanmanın böyle bir avantajı var. Burada şu çift tırnak yerine tek tırnak kullanırsanız, zaten gördünüz oluşan şeyi, bu hata verir. Dolayısıyla şunu mesela çift tırnak yaparsak bu hata düzelir.

Bu arada şeyi anlatmayı unuttum. Böyle konuya daldık. Biz neredeyiz, şu an ne yapıyoruz diye. Bu gördüğünüz arkadaşlar, Google'ın **Colab** ortamı. **colab.research.google.com**. Uzantı... Buranın da linkini paylaşacağım. Python kodlarınızı ya da yapay zeka projelerinizi çalıştırabileceğiniz, web üzerinden kullanabileceğiniz bir ortam. Burada yine kendiniz de birazcık kurcalarsanız, derslerde bizler de bahsederiz. Daha önce bilmiyorum, **Kaggle** ortamında falan çalıştıysanız eğer, aşina olacaksınız oradan. **GPU**'lar falan kullanabiliyorsunuz. Bu derste yapacağımız küçük projeler için geçerli değil ama daha büyük veri setleriyle uğraştığınızda **GPU** üzerinde çalışmanın avantajları oluyor. Yine bu **Colab** ortamının ben linkini atarım. Birazcık kendiniz bir şeyler oluşturmaya çalışın, kurcalayın. Bir sorunuz olursa zaten Lodos'tan bize her zaman ulaşabilirsiniz. Bir takım ofis saatleri de yapacağız bootcamp'in ilerleyen süreçlerinde. Orada yine daha detaylı sorulara değiniyor oluruz.

Son olarak mantıksal tipe değinelim, **boolean**. Burada da **True** ve **False** kelimelerinin baş harfleri büyük olmalı. Dikkat etmeniz gereken tek şey bu. Zaten küçük yazarsanız, sağdaki soldaki değişken ismiyle aynı renkte olacak ve altına bir çizgi gelecek, dalgalı bir çizgi. Size bir uyarı verecek. O yüzden büyük harfle yazdığınız zaman zaten doğru yazdığınızı buradan da anlayabilirsiniz. Bunu ekrana **f-string** ile bastırdığınız zaman "durum_dogru"nun değeri **True**, diğerinin tipi **boolean** veri tipinde. **None** tipi... Arkadaşlar, **None** diye bir yaratık var. Bu sıfırla aynı şey değil yani genel olarak boşluk. Tamamen aslında, sıfır da bir değer çünkü. **None** da yani biz boşluğu ifade etmek için kullanıyoruz. Boşlukla sıfır arasında nasıl bir fark var? Özellikle yapay zeka projelerinde veri setini işlerken bir takım değerler **NaN** oluyor. Bu değerleri başka değerlerle uygun şekilde değiştirmek zorundasınız. Çünkü o şekilde bir yapay zeka modeli eğitmeye çalışırsanız kodunuz hata verir. **None**'un böyle bir önemi var. Burada son olarak şeyden de bahsedelim, işte değişkenin değeri "boş", değişken tipini yazdırdığım zaman **NoneType**. Yukarıda bahsetmiştim, Python dinamik bir dil. Bir değişkeni tanımladıktan sonra, bunun tipinden **NoneType** olsa bile, siz buraya bir sayı yazdığınız zaman değişkenin değeri ve tipi anında değişir diyelim arkadaşlar.

Burada da yine basitçe bir şöyle bir gösterim yapmak istedim. Bu **if else** nedir, dediğim gibi sonraki bölümlere atıfta bulunacağım ara ara demiştim. **bos_degisken2** diye bir değişken olarak buna eğer **None** dersem, **if bos_degisken2** yani **bos_degisken2**'nin bir değeri varsa demek aslında bu, ekrana "Selam" yaz, değilse "Merhaba". Ama bizim bos_degisken tamamen bomboş olduğu için ekrana "Merhaba" yazdırdı. **if** ifadeleri nasıl çalışıyor, buna daha sonra değineceğiz.

Şimdi bu notebook'un içinde temelde ihtiyacınız olmayan ama böyle bir tık daha kurcalamak isteyen arkadaşlara ilham vermesi amacıyla yerleştirdiğim şeyler var. Bunları "202" ile isimlendirdim. Yukarıda gördüğünüz gibi Python'da değişkenlerin tipini otomatik algılıyor ama şirketlerde, yani diyelim iş hayatına atıldığınızda, kodunuzu belli bir standartta yazmanız gerekiyor. Sizin değişkenlerinizi okuyan gören herkes bunun ne olduğunu anlasın diye. Bu gibi durumlarda ne tür bir veri tutulmasını beklediğinizi belirtmek isteyebilirsiniz. Burada tip belirteçleri devreye giriyor. Nedir hocam tip belirteçleri kısaca? Benim ismim **isim: str**. Bu bir **string** olmalı. Burada bir değer vermek zorunda değilsiniz, burayı boş da bırakabilirsiniz. Bunun **float** olmasını bekliyorum diyebilirsiniz, basitçe. Ancak Python bu kuralı zorunlu kılmaz. Yani bu sadece bir belirteç. İşte burada ben **integer** olacak bu dedim, 32 atadım ama bunu gelip burada **string** değerinde, **string** olarak 32, yazı olarak 32 yazarsam herhangi bir şekilde hata vermez. Sadece burada kullandığınız kod editörleri o altta gördüğünüz dalgalı çizgi ile sizi uyarır. İşte editör **isim.** yazdığınızda size **string** metotlarını önerir gibi avantajları var. Bunun dışında yoksa kodunuzun çalışmaması gibi bir şey burada olmuyor.

Geldik operatörlere. Bunlar da kısaca değişkenler ve değerler üzerinde işlemler gerçekleştirmek için kullandığınız semboller. Aritmetik, karşılaştırma, mantıksal, atama, üyelik ve kimlik operatörleri diye beşe ayrılıyor. Ya tam Türkçe adını bulmaya çalıştım, çok olmadı. "Üyelik" ve "kimlik" diye bunları görmezsiniz aslında, hani **in** ve **is** operatörü diye görebilirsiniz 5. bölümü. Ama bunun dışındakiler zaten şey, çoğunlukla lisedeki derslerimizden falan bildiğiniz şeyler. Yani yazılımla haşır neşir olmasanız bile bir bakışta ne işe yaradığını, ne yaptığını anladığınız şeyler. Ne gibi? Örnek, aritmetik operatörler. Yani toplama, çıkarma, çarpma, bölme. Tam bölme diye ayrı bir operasyon var burada. İşte mod diye geçiyordu galiba bu lisede, bölme işleminden kalanı veriyor. Çarpma işareti iki kere yan yana yazdığınız zaman da bu da bizi kuvvet alma aritmetik operasyonuna götürüyor. Burada işte iki tane değişken tanımladık, biri 10, biri 3. Toplama, çıkarma, çarpma ve kuvvet işlemleriyle... Şöyle şimdi küçük olduğu için biraz aşağı kayarak göstereceğim. Burada işte bu işlemlerin sonucu. Kuvvet aldığınız zaman 10'un 3. kuvveti. Bölme işlemi yaptığım zaman, bölme ile tam bölme arasında böyle bir fark var. Bölme işlemi yaptığınızda herhangi bir şey belirtmezseniz, hani "benim aldığım sonuç **integer** olsun" demezseniz, her zaman için size **float**

bir değer döner. Modül işlemi yaptığınız zaman da 10'un 3'e bölümünden kalan 1 olduğu için bize 1 değerini eşitliyor. Şunlardan da bahsedeyim, bu aritmetik operatörleri **string**'lerle de kullanabiliyorsunuz. Artı işlemini kullandığınızda, toplama işlemini kullandığınızda, o metinleri birleştiriyor. **Metin1** "Merhaba", **Metin2** "Dünya". Arada ben şu arada bir boşluk karakteri bırakırsam eğer, o zaman "Merhaba Dünya"yı ekrana basar. **Metin1**'i de 3 ile çarparsam, "Merhaba"yı arka arkaya üç kere ekrana basıyor.

Karşılaştırma operatörleri. Bunların sonuçları her zaman **boolean** oluyor, **True** ya da **False** değeri alıyor. Eşit mi, eşit değil mi, büyük mü, küçük mü, büyük veya eşit mi, küçük veya eşit mi gibi durumlar var burada. Bu durumları genelde **if-else** döngülerinde ya da işte "bu değer bundan küçükse şöyle yap, büyükse böyle yap, işte eşitse şunu kullan" gibi durumlar belirlemek için kullanıyorsunuz ki örneklerini zaten göreceğiz. **a == b**'yse bana direkt şu işlemin sonucunu bastır demiş oluyorum ben bu köşeli, süslü parantezler içinde. Bu değerin, bu işlemin değeri **False** olduğu için, çıktısında da **False** yazıyor. Buna da dikkat çekmek isterim. Yani bu **f-string**'i kullanarak... Sürekli sürekli değiniyorum çünkü gerçekten yaptığınız işlemin log'unu bastırmak için, log'u dedim hani yaptığınız işlemin sonucunu ekrana yazdırmak için diyeyim, log kelimesi yabancı olabilir bir kısmınıza, bu **f-string** oldukça kullanışlı bir metot.

Mantıksal operatörlere gelelim. Buna bir önceki yayında Özge Hocam da değinmiştir. Nasıl kullanılıyorsa, anlamsal olarak nasıl kullanılıyorsa, yazımı tabii ki farklı ama anlamsal olarak nasıl kullanıyorsanız Python'da da aslında o işe yarıyor. **and** koşulu, her iki koşul **True** ise **True** dönüyor. **or** koşulu, ikisinden biri **True** ise **True** dönüyor. **not** koşulu, durumu tersine çeviriyor kısaca. Burada işte 5 3'ten büyükse ve 5 10'dan küçükse **True** dönüyor. O **True**'nun da sonucu yine **True** olduğu için... Şurada göstereyim size, iki tane mantıksal operatörü **and** ile, bir karşılaştırma operatörünü özür diliyorum, mantıksal bir **and** operatörü ile birbirine bağlayabiliyorsunuz. Yani siz diyelim gittiniz, bir takım değerler çektiniz. Bu iki tablodaki değerler bir koşula eşitse eğer, bana **True** dön dediniz. Gibi bir kullanım senaryosu olabilir burada.

Atama operatörlerinden bir örneğini zaten yukarıda gördük. Kısaca bir değişkene bir değer atamak için kullanıyoruz. Eşittir, yukarıda gösterdik, değer atıyoruz. Artı eşittir, topla ve ata demek. Yani ben **x += 5** dersem, x'i 5 ile toplayıp değerini yine x'e ata demiş oluyorum. Eksi eşittir, çarpı eşittir, bölü eşittir benzer şekilde, önce soldaki işlemi yapıp sonra o değişkene yeni değerini atıyor. Bunun da örneklerini böyle görebiliriz. **sayi = 10** dedim, **sayi += 5** dersem sayının değeri 15 olur. **sayi *= 2** dersem, yukarıda bu sayının değeri 15 olduğu için **sayi *= 2** dediğim zaman yeni değerimiz bizim 30 olmuş oluyor.

"Üyelik ve Kimlik" operatörleri. Dediğim gibi bunun Türkçeleri biraz sevimsiz oldu, **in** ve **is** operatörü diye aklınızda tutmanız yeterli. **in** operatörü üyelik belirtiyor. Yani bir değer arıyorum ben, o değer aradığım yerin içinde mi? Bu ne demek olabilir? Bir veri seti var elinizde, o veri içerisinde bir tane kelimeyi arıyorsunuz. En az bir kere bile varsa bana **True** dön gibi bir yaklaşımınız varsa eğer, bu **in** operatörünü kullanabilirsiniz. Örneğin metin değişkeninin içerisinde "Python" olduğu için size **True** dönecek. Bunun kısaca kullanım alanı da bu şekilde arkadaşlar.

Burada yine bir "202" konusu var. Birazcık kafanızı karıştırabilir 101 seviyede. **==** dediğimiz operatörle **is** birbirinden farklı şeyler aslında. **==** ile biz iki değişkenin değerlerinin aynı olup olmadığına bakıyoruz. **is** kullandığımız zaman bellekte aynı yeri, aynı objeyi gösterip göstermediğine bakıyoruz. Bunu bir apartman metaforu ile açıklamaya çalıştım. **==** ile farklı bellek adresleri içindeki eşyaların değerinin birebir aynı olup olmadığına bakıyoruz **==** ile. **is** kullandığımız zaman iki farklı anahtarın aynı daireyi açıp açmadığına bakıyoruz gibi düşünebilirsiniz. Ama bunu biraz daha görselleştirmek isterseniz, benim **liste_c** dediğim şey **liste_a**'ya eşit olmuş. **liste_a**'nın değeri **liste_b**'nin değerine eşit, **==**. Peki **liste_a** ile **liste_b** bellekte aynı obje

mi? Hayır, çünkü bunlar farklı nesneler, farklı objeler. Dolayısıyla **liste_a is liste_c**'ye eşit. Aynı obje olduğu için, kopyalayarak oluşturduğumuz için bunlar aynı objeye eşit. Aynı objeye eşit olması bize ne getiriyor? Ben **liste_c**'ye yeni eleman eklediğimde **liste_a**'ya da aynı eleman eklenmiş oluyor. Ama dediğim gibi bu birazcık 202 seviye. Ama işte bu tip durumlarda böyle farklı bir liste atayarak ilerlemek, ilk listenin orijinalliğini korumak için önemli olabiliyor arkadaşlar.

Yine 202 kısmında bir **bitwise** operatörlerimiz var. Bu **bitwise** operatörler de yine C, C++ gibi dillerde kod yazmış olanlarınız için daha ziyade işe yarar olacaktır. Onlar birazcık daha aşinadır buraya. Burada **bitwise AND**, **bitwise OR**, **bitwise XOR**, sola kaydırma, sağa kaydırma gibi operatörlerimiz var. Bunlar işte ikili tabanlarda çalışıyorlar. **AND** 2 bitte de 1 varsa 1, yoksa 0 veriyor. Burada **a=5**, **b=6** dersek eğer, a'yı bu şekilde, b'yi de bu şekilde yazıyoruz ikilik tabanında. Bu 5 demek, bu da 6 demek. İkisinin kesiştiği sadece soldan ikinci şeyler olduğu için, soldan ikinci 1'ler olduğu için, **a AND b**'nin değeri 4 olmuş oluyor. **OR** kapısı, 2 bitten en az biri 1 ise 1, yoksa 0 veriyor. Yukarıda gösterdiğimiz **AND**'le aslında aynı şekillerde çalışıyorlar. Sola kaydırma, bitleri sola kaydırıyor. İkilik tabanda yazdığımız için sola kaydırınca hızlı yoldan 2 ile çarpma demek oluyor bu kısaca diyelim arkadaşlar.

Geldik veri yapılarına. En çok konuşacağımız noktalardan bir tanesi bu veri yapıları. Çünkü özellikle **liste** ve şu **sözlük** dediğimiz **dictionary**'leri yapay zeka operasyonlarında sıklıkla kullanıyor olacaksınız. **Demet**'ler, **küme**'lerden de bahsedeceğiz ama bunları nispeten daha az kullanıyorsunuz.

Soru: Python'ı öğrenmek isteyenlerin temelinde farklı dilleri bilme gereği var mı?

Göker Güner: Gerek yok, çünkü öğrenmesi en basit dillerden bir tanesi. Diğer dil bilmek gibi bir zorunluluğumuz bu anlamda yok. Sıfırdan başlayanlar için oldukça uygun diyelim.

Şimdi listeler, köşeli parantezle oluşturuyoruz. Sıralıdırlar. Yani hangi sırayla öğe eklediyseniz, siz modifiye, kendiniz bizzat modifiye etmediğiniz sürece o sırada kalıyor. Değiştirilebilirler. Bu değiştirilebilirlik önemli çünkü bazı veri yapılarını değiştirmediğimizi de göreceğiz. Ama listeleri oluşturduktan sonra öğeleri ekleyebilir, silebilir, yerini veya öğenin kendisini değiştirebilirsiniz. Böyle şeyler yapabiliyorsunuz. En güzel tarafı da farklı veri tiplerini bir arada tutabilmesi. Böyle bir liste oluşturabilirsiniz. Tamamen **string**'lerden oluşan, sayılardan oluşan bir liste oluşturabilirsiniz. Yine az önce bahsettiğimiz gibi karışık veri tiplerinden oluşan bir liste de oluşturabilirsiniz. Bir şey daha söyleyeyim, az önce değinmedim burada çalışırken size kolaylık olsun diye **meyveler**, **sayilar**, **karisik_liste** diye isimlendirdim bu değişkenleri ama yarın bugün hani yaptığınız projeleri GitHub'ınıza falan koyduğunuzda, bunlar sizin portfolyo projelerinizde globale açık olacaklar. O yüzden orada birazcık standardı takip edip değişken isimlerini İngilizce koymakta fayda var diye bir tavsiye vermiş olayım naçizane. Bu listeleri bastırdığınız zaman **meyveler** değişkenim benim "elma", "muz", "çilek" listesine eşit. Dolayısıyla ekrana bunu bu şekilde basıyor. Listenin uzunluğunu **len()** fonksiyonuyla alıyoruz. **len(meyveler)** dediğimiz zaman, parantez içinde "elma", "muz", "çilek" yani benim **meyveler** listemin uzunluğu 3 olduğu için eleman sayısı 3 diye ekrana basabiliyor bunu.

Geldik indeksleme ve dilimleme. İndeksleme nedir? Aslında yani bir adresleme gibi düşünebiliriz. Her öğenin bir indeks numarası vardır ve sıfırdan başlar. **liste[0]** ilk elemanı, **liste[1]** ikinci elemanı verir ve böyle devam eder. Negatif indeksleme de vardır. **liste[-1]** son elemanı, **liste[-2]** de sondan ikinci elemanı. Bunun nasıl bir önemi olabilir? Belki işte elinizde bir metin var ve bunun son kelimesini ya da son n tane kelmesini almak istemiyorsunuz diyelim. Bu gibi noktalarda bu negatif indekslemeyi kullanıyorsunuz. Bunların daha kullanışlı yolları da var tabii, orada da dilimleme devreye giriyor. Başlangıç, bitiş

ve adım. Tabii her kullanımda bunun üçünü de kullanmanız gerekmiyor ki örneklerimizde onları göreceğiz. Böyle 6 elemanlı bir listem var benim. **harfler[0]** bana ilk elemanı, **harfler[-1]** son elemanı f'yi döndürecek. Sondan ikinci elemanı döndürmek için de **harfler[-2]**. Buradan çıktılarını görebiliyoruz. Bir adım sayısı belirtmeden direkt 1'den 4'e kadar al dedim. Tekrar yukarıda hatırlatmam gerekir: Başlangıç, bitiş ve adım formatıyla aralık değiştirebiliyorum ama ben burada başlangıcı, bitişi kullandım, adımı kullanmadım. Adımı kullanmayınca direkt olarak 1'den 4'e kadar sıralıyor. Bitiş değerini dahil etmiyor. **harfler[1:4]** dediğim zaman 1, 2, 3 numaralı indeksleri alıp bitir demiş oluyorum. Bu yüzden 4'ü tercih etmiyor. Baştan üçüncü indekse kadar al dediğim zaman, başlangıç indeksi vermiyorum. Bu indeksi vermediğimde 3'ü dahil etmiyor. Geldik adım sayısına. Tüm listeyi ikişer atlayarak al dediğimde, tüm listeyi dediğim için başlangıç ve bitiş indeksi vermedim. Sadece kaçar atlaması gerektiğini söyledim. Bir sayıyı tersten yazdırmak isterseniz de ne başlangıç ne bitiş indeksi vermeden -1 diyerek listeyi tersten yazdırabilirsiniz.

Geldik listenin metotlarına. Burada tabii listenin çok metodu var aslında ama temellerine değinmek istedim sadece. **append()** listenin sonuna öğe ekler. **insert()** belirtilen indekse bir öğe ekler. **pop()** belirtilen indeksteki öğeyi siler ve döndürür. İndeks verilmezse son öğeyi siler. **remove()** dediğinizde değeri belirtilen ilk öğeyi listeden siler. **sort()** listeyi sıralar. **reverse()** listeyi ters çevirir diye anlatabiliriz kısaca. Burada hayvanlardan oluşan bir listemiz var. "balık"ı **append()** etmek istersek, direkt burada **append("balık")** sonrasına gidelim. "kedi", "köpek", "kuş", en sona "balık" ekliyoruz. **insert()** dediğimiz zaman 1. indekse "hamster" ekle dediğimizde, 1. indeks dediğimiz neresi? Listenin 2. sırası. "hamster" sonrasına baktığımız zaman "kedi", "hamster", "köpek", "kuş", "balık" diye devam ediyor. "hamster" oraya eklendi, listeyi kaydırdı yani herhangi bir elemanı kaybetmedi orada. **pop()**'u kullanırken buraya 2 dersem, 2. indeksteki "köpek" öğesini artık çıkarıyor. "hamster" ekledikten sonra köpeğin 2 numaralı indekse kaydığına dikkat edin. **pop(2)** sonrası liste dediğim zaman artık köpeği listede görmüyoruz. "Çıkarılan hayvan: köpek" diye zaten bunu, **hayvanlar.pop(2)** işleminin çıktısını bir değişkene atarsanız eğer, bu değişkeni ekrana yazdırabiliyorsunuz. **hayvanlar.remove("kedi")**, "kedi" değerini sadece, direkt o değer neredeyse gidip onu bulup onu siliyor. Kediyi de sildiğiniz zaman "hamster", "kuş", "balık" diye listemiz kalıyor elimizde arkadaşlar. Sıralamada, sayıları karışık verdiniz, **.sort()** dediğiniz zaman küçükten büyüğe sıralar diyelim.

Şimdi burada en çok kullandığımız **list comprehension**. Bunu nerede kullanıyoruz? Bize daha okunaklı kodlar yazmamızı sağlıyor. O da yarın bugün şirkette çalışmaya başladığımızda bir yerlerde kodumuzun daha kolay anlaşılmasını sağlıyor diyebiliriz aslında. Nedir **list comprehension** kısaca bakalım. **for sayi in [1, 2, 3, 4, 5]**... Az sonra demiştim, sonraki bölümlere atıfta bulunacağım ara ara, **for** döngüsü diye bir şey göreceğiz. Nedir bu **for** döngüsü? Benim burada bir kümem olacak, bir değerler kümem. Burada verdiğim değişken, solda verdiğim değişken yani bu örnek için sayi değişkeni, bu değerlerin içinde dönecek, bu değerleri tek tek tarayacak gibi düşünebiliriz. Ne yapıyormuş tarayıcı? **kareler.append()** etmek istiyorum. Neyi? **sayi**2**. Şimdi ben bunu burada iki satırda yaptım. Tabii basit bir örnek bu. İki satır yerine tek satıra düşürmek büyük bir şey değil ama burada daha uzun süren operasyonlarımız da olacak belki, çalıştığınız veriler üzerinde. Bunu daha okunaklı yazmak için direkt tek satıra yazıp geçebiliyorsunuz arkadaşlar. **kareler = [sayi**2 for sayi in [1, 2, 3, 4, 5]]**... 1, 2, 3, 4, 5 listesinde sayi diye bir değişkenle dön, her bir değişkenin karesini alıp bana kareler diye bir liste oluştur demiş oluyorum. Burada hatta koşullu da, yani **if** de kullanabilirim ben bunun içinde. **if**'i nasıl kullanırım? Elimde bir sayılar listesi var. Eğer bu sayılar çift ise... O da nasıl oluyor? Bana sayının karesini al. Neye göre sayının karesini alacağım ben? **if sayi % 2 == 0**. Sayının 2'ye bölümünden kalan 0 ise, yani benim sayım çift sayıysa bana bir cift_kareler listesi oluştur. Bu notebook'un sonunda bir takım kaynaklardan bahsedeceğiz ki Özge kendi yayınında da bahsetti galiba o kodlama alıştırması platformlarında, bu **list**

comprehension size epeyce bir vakit kazandırabilir olacak. Ama dediğim gibi burası 202 kısmı yani temelini anladıktan bir adım sonra çalışacak kısım.

Geldik demetlere (**tuple**). Listelere çok benzerler. Pek önemli farkları, listelerde bahsetmiştim, değiştirilebilir özelliği önemli diye. Demetleri değiştiremiyoruz. Bazen böyle değiştirmek istemediğimiz, değiştirilmemesi gereken veriler kullandığımız zaman, işte nedir? Koordinatlar, veritabanı kayıtları gibi noktalarda bazen bazı kayıtların değişmesini istemeyiz. O noktada bunu Python içerisinde kullanırken **tuple** olarak yani demet olarak kullanırsanız, değiştirmeden hayatımıza devam edebiliyor olursunuz arkadaşlar. Değiştirilemezlik nedir peki? İşte benim bir koordinatım var veya bir RGB renk kodum var. İndeksleme, dilimleme listeler ile birebir aynı. **koordinatlar[0]** dediğimde 10.0, 0. indeksteki. Renkleri görmek istersem işte 1'den 3'e kadar renklerim 0, 0. Demetler değiştirilemez. Bu satırı kendiniz açarsanız, açtığınızda **TypeError** verdiğini göreceksiniz. **koordinatlar[0] = 5.0**... Çalıştır... **TypeError: 'tuple' object does not support item assignment**. "Ben bunu desteklemiyorum, sen bir item assign edemezsin buraya" dedi. "**Tuple unpacking**" dediğimiz, x ve y değerlerini, yani bir koordinatın x değerini başka bir işlemde, y değerini başka bir işlemde kullanmak isterseniz bu şekilde değerleri dışarıya alabiliyorsunuz arkadaşlar.

Kümeler (**set**). Burada en az kullandığımız şey. Belki ben hiç yani denk bile gelmemiş olabilirim. Matematikteki kümeler gibidir ve iki temel özellikleri vardır. Öğelerin belirli bir sırası yoktur. Bu sebeple indeksleme yapamıyorsunuz. Bir öğeyi birden fazla kez barındırmıyor. Kümede hani her elemanın maksimum bir kere olabilmesi gibi düşünün bunu. Süslü parantez veya **set()** fonksiyonuyla oluşturabiliyorsunuz. Boş bir küme oluşturmak için süslü parantez kullanmıyorsunuz, boş sözlük oluşturuyor. Daima **set()** kullanıyoruz burada. Ne zaman kullanılır? Bir listedeki tekrarlayan öğeleri silmek isteyebiliriz bazen. Öğelerin tekrarlandığından da şüphe edebiliriz. Çok uzun bir listeyse teker teker kontrol etmek yerine **set** kullanırım, tekrarlıyorsa silsin demek için yaparım bunu. Bir öğenin bir koleksiyonda olup olmadığını çok hızlı kontrol etmek için ve matematiksel küme işlemleri yapmak için diyelim. Listenin tekrarlayan öğelerini kaldırmak için kullanıyoruz. **set(tekrarlayan_liste)** dediğiniz zaman bu listem benim artık bir kümeye dönüşür. Bu birleşim, kesişim, fark konuları tamamen matematikteki ile aynı.

Geldik bir diğer önemli veri yapısı, sözcükler ya da **dictionary** İngilizcesi ile. Bu da yine Python'da çok kullanacağımız veri tiplerinden bir tanesi. Nasıl kullanıyoruz bunu? Biraz daha böyle bir işte veri seti işleme aşamasına geldiğinizde, ilerlediğinizde, yapay zeka projelerinizde, hatta ve hatta machine learning için değil belki, generative AI projelerinde daha çok **JSON** denen bir veri tipi kullanacaksanız arkadaşlar. Bu veri tipinin yazımı bu sözcüklerle çok benzer olacak. Dolayısıyla bu süslü parantez içinde **key: value**, **ad: "Ahmet"**. ad bizim key'imiz, "Ahmet" bizim value'muz. Anahtarımız ve değerlerimiz yani. Bu gibi yazım tipine aşina olmanız gerekiyor. En çok kullanacağımız gösterme yöntemlerinden bir tanesi. Anahtarlar benzersiz olmalıdır ve değiştirilemez bir tipte olmalıdır. Genellikle **string** veya **integer** olarak belirleriz. Değerler herhangi bir tipte olabilir ve tekrarlanabilir. Ne zaman kullanılır? Verileri bir etiketle hızlıca bulmak istediğimizde kullanılır. Ne demek? Örneğin ben **ogrenci** diye bir tane sözlük oluşturdum, **dictionary** oluşturdum. Ahmet diye 25 yaşında bir öğrenci arkadaşımız var, mühendislik okuyor. **ogrenci['ad']**... Az önce şurada bahsettiğimiz işte "hızlıca bir şeyleri bulmak istediğimizde kullanılır" dediğimiz nokta bu. Yapısının ad key'ine ulaşmak istediğim zaman bunu bu şekilde yazabiliyorum ve bana Ahmet'in adını dönüyor. **.get()** metodu, anahtar yoksa hata vermek yerine **None** veya istediğiniz bir değeri döndürür. Okulunu döndürmek istediniz, **ogrenci.get('okul')** dediğiniz zaman, yukarıda benim aslında 'okul' diye bir değerim yok ama **get('okul')** dediğiniz zaman **None** değerini dönecek. Değer ekleme, güncelleme kısmına geldiğimiz zaman, yaşı 25'ti Ahmet'in yukarıda. Ben burada yas anahtarına 26 değerini atadığım zaman artık değerimi güncellemiş oluyorum. Ahmet bir yıl yaşlanmış, 26 yaşına

gelmiş oluyor. 'okul' anahtarını da böyle bu şekilde ekleyebilirim. **ogrenci['okul'] = "XYZ Üniversitesi"** dediğim zaman 'okul' özelliği gelmiş oluyor. Sadece anahtar ya da sadece değerlerin arasında gezinmek için de bir takım böyle altta **.keys()**, **.values()** gibi metotları kullanabiliyoruz. **.items()** hem anahtarı hem de değeri **tuple** olarak döndürüyor.

List comprehension'a benzer şekilde **dictionary comprehension**. Bu da çok kullanılmayan yine bir metodu aslında. **List comprehension** gibi mevcut bir yapıdan yeni bir sözlük oluşturmak için aslında kullanılıyor. Bunu **dictionary comprehension** ile tek satırda bu şekilde yapabiliriz. **List comprehension**'dan farklı olarak bu sefer sözlük oluşturduğumuz için süslü parantez kullanıyoruz.

Geldik sürekli bahsettiğimiz **if**, **elif**, **else** anahtar kelimelerine. Burada özellikle "kodumuzun düşünmesini ve karar almasını sağlarlar" diye belirttim. Böyle şeyler görürseniz, geyik muhabbetlerini görürseniz yapay zeka ile ilgili, "aslında arkada **if-else**'ler çalışıyor" diye. Tam olarak böyle değil ama bir takım otomasyon projelerini "**if** bu, bunu yap. O değil, buysa bunu yap. **elif**'i kullanarak... Hiçbiri değilse, **else** ile şunu yap" diye böyle programlayabiliyorsunuz. Aşağıda bunların kullanım örneklerini göreceğiz. Bir yapay zeka değil ama otomatik, kendi kendinize ufak bir düşünme sistemi tabii tasarlamanızı sağlayabilir. **if-else** yapısı, yukarıda bir örneğini zaten göstermiştik. **if kosul**, koşul **True** ise burası çalışıyor. Koşul **False** ise **else**'deki blok çalışır. **elif** tarafında da yine, hatta direkt şöyle örneğinden göstereyim, birkaç tane koşulu check ettiğimizde... İşte benim bir notum var, notum 75. 50'den küçükse... Ama 40'tan küçük olma koşulunu zaten yukarıda kontrol etmiştik. Dolayısıyla aslında 40 ile 50 arası gibi algılıyor sistemimiz. O da değilse 60'tan küçük mü diye bakıyor. O da değilse 70'ten küçük mü diye bakıyor. Artık hiçbiri değilse, **else** bloğunun içindeki 'A'yı ekrana benim notum olarak basıyor arkadaşlar. Bu **if** durumunu, iç döngüsünü, **if** koşulunu diyeyim, iç içe olarak da kullanabiliyorsunuz. Örneğin benim yaşım 22, ehliyet durumumu "var" olarak işaretledim. İlk baktığım şey doğal olarak yaş 18'den büyük mü? 18'den büyük ise "kişi reşittir" ekrana basıyor. Eğer 18'den büyükse ikinci bir kontrol yapıyor: Ehliyet durumu "var" mı? Eğer "var"sa "araç kullanabilir", değilse "araç kullanamaz, ehliyeti yok". Ama burada ilk kontrol ettiğim şey ne? 18'den büyük olup olmama durumu. Eğer 18'den büyük değilse, bu arayı komple atlıyor. Dolayısıyla size zamandan tasarruf sağlıyor ve direkt olarak "kişi reşit değil, araç kullanamaz" yazdırıyor. Başka condition'ları kontrol etmenize gerek kalmıyor.

Şimdi burada yine bir 202 seviye bir durum var. "**Truthiness**" (doğruluk değerleri) diye geçiyor bunlar. **if** ifadeleri herhangi bir Python değerini kontrol edebiliyor ama o değerler bazıları default olarak **True** ya da **False** olarak kabul edilebiliyor. Örneğin siz 0 verdiğinizde, boş metin, boş liste, boş **tuple**, falan filan verdiğinizde, bunların hepsini Python **False** olarak kabul ediyor. **True** olarak kabul edilenler de aslında temelde bunun dışındaki her şey. **if liste** olarak daha sade, daha okunaklı bir kod yazmış oluyorsunuz arkadaşlar. **Ternary operations**, tek satırlık koşul diye geçiyor bu. Bu da yine bir diğer ileri seviye bir kullanım yöntemi Python'da.

Geldik döngülere. Döngülerle ilgili sorularınızı görüyorum. Bir yarım saate 1. bölümü bitiririz. Dediğim gibi, zaten asıl uzun olan kısmı burasıydı. Bu bölümün sonunda burayla ilgili birazcık soru cevap yaparız diyeyim. Şimdi programlamada döngü... "Don't repeat yourself", kısaca DRY diye bir prensip var. Bunu duymazsınız ama yarın bugün duyarsanız da bir yerlerde "böyle bir prensip var" diye aşina olun diye örnek vermek istedim. Bir işlemi 100 kez yapmak istiyorsanız o kodu kopyalayıp yapıştırmak yerine döngüleri kullanıyorsunuz. İki ana döngü türü var, **for** ve **while**. Zaten **for** döngüsü ile ilgili örnek yaptık, az buçuk neye benzediğini biliyorsunuz. Birazcık daha detayına ineceğiz. Yukarıda gördüğünüz gibi bir tane koleksiyonumuz var elimizde. Bu koleksiyonun içinde verdiğiniz bir isimle dönüyorsunuz sürekli ve içeride bu döndüğünüz değişkene, değişken

adına bir işlem yaptırıyorsunuz. Yaptırmayadabilirsiniz. **range()** fonksiyonumuz var. Bazen elinizde bir liste olmayabilir ama bu işlemi belirli bir sayıda, örneğin 10 kere yapmak istersiniz. O durumda **range()** fonksiyonu imdadımıza yetişiyor. Bize sanal bir sayı dizisi oluşturuyor. **while** döngüsü, bir koşul **True** olduğu sürece tekrarlanan kod bloğudur. Burada bir koşulumuz var. Bu blokta koşulu **False** yapacak bir değişikliği genelde biz yapıyoruz arkadaşlar. Yani bir şekilde bu döngünün sonsuz döngü olmaması için bu döngüyü bir yerde kırmanız gerekiyor. **break** ile döngüyü kırabiliyoruz, **continue** ile o adımı atlayıp devam edebiliyoruz. **pass**'e geçersek, hiçbir şey yapmaz. **enumerate()** bizi nereye getiriyor? Bir listede gezerken hem indeksi hem de indeksteki değeri almak için. **zip** fonksiyonu, iki veya daha fazla listeyi fermuar gibi birleştirir ve her adımda her iki listeden de birer eleman verir. Döngülerde **else** bloğu. Genelde böyle **if-else** diye kullanıyoruz biz ama **for**'da da **else** bloğu kullanabiliyoruz.

Geldik fonksiyonlara. Fonksiyonlarla ilgili bu arada şunu söyleyim, **Colab** ortamında hücre hücre çalıştığınız için, az sonra göstereceğim, fonksiyon yazmaya **Colab** ortamında çok ihtiyaç duymayabilirsiniz ama büyük bir kolaylıktır. Kendi lokalinizde Python dosyaları ile, **.py** uzantılı dosyalarda çalıştığınızda hayatımızı kolaylaştırır. Zaten bir noktadan sonra artık fonksiyon yazmadan çalışamadığınızı fark ediyorsunuz. O yüzden bu fonksiyon tanımlama işleri birazcık önemli diyelim. Burada da yine döngüler gibi arkadaşlar, girintiler önemli. Bir selamla fonksiyonu tanımlamışım, ne yapıyor bu selamla fonksiyonu? Ekrana "Merhaba" yazdırıyor. Parametreler ve argümanlar. Fonksiyonu tanımlarken parantez içine yazdığımız değişken adıdır. Örneğin, **def topla(a, b):**. a ve b değişkenleri benim parametrelerim. Bu fonksiyonu çalıştırırken burada girdiğim 5 ve 3 de benim argümanlarım olmuş oluyor. **lambda** ifadeleri, minik böyle küçük fonksiyonlar olarak tanımlayabiliriz. **lambda**'yı. Bu da yine 202 seviye bir konu.

Hata yakalama kısmı. Bu da yine çok uzun bir kod bloğu çalıştırdığınız zaman bir hata verdiğinde, "bu hata neden oldu, bu hatayı nasıl düzeltebilirim?" sorusuna kendiniz yanıtlamak istediğinizde kullandığınız **try-except**.

Geldik dosya işlemlerine. Burada aslında çok uzun uzadıya bahsetmeyeceğim bundan çünkü genelde text verilerini işlemek için ikinci kısımda göstereceğim başka metotlar var. Ama bazı keyword'ler önemli burada. Yani bir dosyayı açmak istediğiniz zaman **with open()** açmanız önemli.

Nesne yönelimli programlama. Bunu da yapay zeka operasyonlarında genelde çok kullanmadan yol alabiliyoruz. Özellikle notebook'ların içerisinde hiç **class** tanımlamadan çalıştığınız case'ler olacak ama öğrenirseniz, naçizane, biraz böyle yapay zeka uygulaması haricinde bir backend tasarladığınız zaman daha derli toplu kod yazmanızı sağlar. O yüzden birazcık önemli diyebilirim açıkçası bu kısım.

Sonunda arkadaşlar, şöyle 1-2 tane tavsiye ekledim diyelim. Öğrenmenin tek yolu öğrendiklerinizi kullanmaktır. Bu bu arada burası bir ödev değil, lütfen bootcamp kapsamında bir değerlendirme falan yapmıyorum. Bir takım projeler geliştirebilirsiniz. Projelerle alakalı bazı fikirler verdim ama bu fikirleri bir tık aşağıda anlatacağım. Kodlama platformlarının linklerini eklemedim, Özge Hocam sunumunda eklemişti. Bir alan seçin ve derinleşin. Birazcık Python'da ilerledikten sonra yani veri bilimine gireceğiz zaten bootcamp kapsamında. Web development tarafında da ilerleyebilirsiniz. Machine learning AI kapsamında gideceğiz. Python'da istediğiniz neredeyse her şeyi yapabilirsiniz arkadaşlar. Çok kısaca şunu göstereyim, sonra ufak bir ara verelim. **FreeCodeCamp** diye benim çok sevdiğim bir platform. Bunun böyle bir tane linkini ekledim. Python'a yeni başlayanlar için 25 farklı proje fikri var burada. Bunları incelerseniz, kendiniz yapmaya çalışırsınız, baktınız olmuyor, gidin ChatGPT'ye "bana bu kodu anlat" deyin. Size projeyi yapsın, kodu anlatsın. Anladıktan sonra dönüp başka bir örnek seçin, bu

sefer onu kendiniz yapmayı deneyin. Bu şekilde kendinizi geliştirebilirsiniz diyeyim. Bu YouTube kanalından da özellikle bahsetmek isterim. **FreeCodeCamp**'ta arkadaşlar, ilginizi çeken neredeyse her şeyle alakalı böyle, atıyorum Data Structures & Algorithms, 49 saat mesela. Videolar böyle yani. Ne ilginizi çekiyorsa, burada onunla ilgili mutlaka tek parçalık, farklı farklı uzunluklardaki, bazıları bunun çok uzun olabiliyor, böyle videolar var. Geneli İngilizce tabii ama hani kodlama temelli bir şey olduğu için zaten ekranda yazıyor ne yaptığını, İngilizcesi de sizi çok zorlamaz diye düşünüyorum açıkçası. Ufak bir ara verelim.

Göker Güner: Buraya da lambda ile ilgili yine örnekler bıraktım. Teker teker hani hepsini böyle "çalıştırınca böyle oluyor" diye anlatmayacağım ama kendiniz deneyerek görebilirsiniz. Zaten dediğim gibi, burada bir lambda'nın ya da herhangi bir fonksiyonun kullanımı ile ilgili bir sorunuz olduğunda Lodos sunucusunda kullanıyor oluruz diyeyim.

Devam edeyim. Hata yakalama kısmı. Bu da yine çok uzun bir kod bloğu çalıştırdığınız zaman bir hata verdiğinde, "bu hata neden oldu, bu hatayı nasıl düzeltebilirim?" sorusuna kendiniz yanıtlamak istediğinizde kullandığınız **try-except**. Genelde **try-except**'i kullanırız burada. Yani **else**, **finally** ile ilgili de örnekler verdim ama burada **else**, **finally** çok da kullanmıyoruz açıkçası. Kullanım alanları var tabii ki. Neden olur diye baktığımızda, bazı yaygın hata örneklerini verdim burada. Bir sayıyı sıfıra bölmek isterseniz, hatalı bir tip dönüşümü yapmak isterseniz, uyumsuz tiplerle işlem yaparsanız, tanımlamadığınız bir değişkeni kullanırsanız ya da olmayan bir indekse erişmek isterseniz, bu tip hatalar Python'da olur. Programın çökmesini engellemenin temel yolu bu **try-except** konusudur. Bir fonksiyonun acaba hata verecek mi diye düşünüyorsanız **try**'ın içine yazarsınız. Eğer fonksiyonunuz hata veriyorsa, "işte benim şu isimli fonksiyonum hata verdi" diye **print** bloğu koyarsınız. Eğer ki ekranda o yazdığınız **print** bloğunu görüyorsanız demek ki orada bir hata var demektir.

Tabii bunu daha güzel bir yolla yapmak da var, daha güzel formatlama yolları. Burada mesela "Merhaba"yı integer'a dönüştürmeye çalıştığım zaman, bu tam sayı olamayacağı için kodunuz hata verecek. "Hata oluştu, geçerli bir sayı girmediniz" yazacak. **try-except** olmasaydı, yukarıdaki hatadan dolayı bu satır asla çalışmazdı. Ben ne oldu? Bir kodu denedim burada, çalışmadı. Çalışmadığı için **except**'e girdim. **except** kısmında bir sorun olmadığı için kodum çalışmaya devam etti ve "program devam ediyor" log'unu ekrana bastı. Yukarıda bahsettiğimiz hataları hani böyle spesifik olarak tanımlamak istersek, ben burada ne yaptım? **try** dedim, **except** dedim, herhangi bir şey vermedim, hata belirtmedim. Ama **except ValueError** dediğimde sadece **ValueError**'u yakalar ya da **except ZeroDivisionError** yazabilirim altına, bu da sadece 0'a bölünmeleri yakalar. Bunun dışında altına yine **except** yazıp hiçbir tip vermezsem, bu ikisi dışında bir hata gelirse onu da buraya atar. Son olarak burada **except exception as e** göstereyim. Yani burada e yazmak zorunda değiliz tabii. Gördüğünüz gibi **except Exception**... Bu ilk **Exception**'ın E'si büyük olmalı... **as hata** dediğimiz zaman hata değişkenim benim artık hata mesajımı ifade ediyor. **hata.__class__.__name__**'i **ValueError**, hata mesajında da böyle bir hata mesajı varmış. Bu mesajı ekrana bastırmak bazen hatanın neyden kaynaklı olduğunu anlamanızı kolaylaştırır. İnceleyelim: "can only concatenate str (not 'int') to str". "Strange" diyor. Yani siz string'e yalnızca string'i concatenate edebilirsiniz. Uç uca birleştirebilirsiniz demek concatenate. Integer'la bunu yapamazsınız. Böyle bir hata aldığınız zaman buradaki işlemdeki hatanızın neyden kaynaklandığını anlamanız da bir tık daha kolay oluyor.

else ve **finally** blokları **try-except** yapısını geliştirmenizi sağlıyor. Burada yine **for** döngüsüne benzer şekilde, **try** bloğu hatasız biterse bu bloğa atıyor sizi. **finally** her zaman o bloğu çalıştırmanızı sağlıyor. Yani siz bu **except**'lerin sonucunda bir hata yakalasanız da yakalamasanız da **finally**'yi çalıştırmanızı sağlıyor bu. **raise** de aynı şekilde. Bazen bazı noktalarda, benim atıyorum veri setimde bir takım böyle non-type'lar falan varsa, bazı noktalarda aldığım hatalar benim kodumun çalışmasını durdurmasını istiyorsam, bana hataları, hata fırlatırım. Bunun için de **raise**'i kullanırım kısaca. **raise ValueError**. Ya bu ne demek? **yas_kaydet(25)**. Nedir abi bu 25? Yaşım sıfırdan küçükse bu mantıksal bir hatadır. Bunu bilinçli olarak fırlatıyoruz. "Yaş negatif olamaz". Yaşım 50 olabilir, yaşım -10 olamaz. Yaşım 150 olamaz, niye? Yaş çok yüksek. Bunu mesela incelemek istediğimizde 25 ve 50'de başarıyla çalışıyor ama -10 girdiğimizde **raise ValueError** fırlatıyor: "Yaş negatif olamaz. Girilen değer: -10". Benim bir takım işte kullanıcıları check ediyorum diyelim. Birisi -10 girmişse benim artık bununla işlem yapmamam gerekiyor. Bu -10 değerinde bir anomali mi var, burada ne oldu diye dönüp elimdeki veriye bakmam gerekiyor. -10 girdiğimde ne oldu? Burada dikkat ederseniz bir **yas_kaydet(150)** de demiştim. 150 ile hiçbir işlem yapmadı. Niye? Çünkü burada ben hata fırlatıp çalıştırmayı durdurdum.

Geldik dosya işlemlerine. Burada aslında çok uzun uzadıya bahsetmeyeceğim bundan çünkü genelde text verilerini işlemek için ikinci kısımda göstereceğim başka metotlar var. Ama bazı keyword'ler önemli burada. Yani bir dosyayı açmak istediğiniz zaman **with open()** açmanız önemli. Bunu kullanmadan da açabiliyorsunuz. Dosya adınızı veriyorsunuz ve dosyanızı hangi modda açtığınızı söylüyorsunuz. Mod ne demek? **w** ile açarsanız yazma modunda açar. Ama **w** ile açtığınız zaman var olan dosyanın içeriğini tamamen silersiniz. Yani siz bunu tekrar tekrar çalıştırdığınızda her defasında dosyanın içeriğini en baştan silecek, ilk yazdığınız hiçbir şeyi dosyada tutmayacak. **r** ile açarsanız sadece okumak için açar. Okumak için açtığınızda içeriye herhangi bir şey yazamazsınız o blok içerisinde. **open()** fonksiyonunun varsayılan modudur bu arada. Yani içeriye **r** yazmadan açarsanız default olarak read modunda açarsınız. Read modunu açtıktan sonra, işte burada bir değişken ismi tanımlıyorsunuz, f olarak tanımlamak zorunda değilsiniz, file'ın kısaltması değil, genelde f kullanılıyor. **f.read()** ile bütün içeriği icerik'e atmış oluyorsun. **a** ile, write'ın daha güvenli hali, dosya mevcut değilse yeni bir dosya oluşturur, içindeki verileri silmez. Append operasyonlarına benzer şekilde imleci dosyanın sonuna götürür, yeni verileri oraya ekler.

Neden **with** kullanıyoruz? Yukarıda bahsetmiştim, **with** kullanmadan da bunu yapabiliriz ama **with** kullanmayı tercih ederiz. Sadece **open** kullanıp ben file diye bir değişken oluşturabilirim ama öyle ya da böyle yaptığım operasyonun sonunda bunu mutlaka **close()** ile kapatmak zorundayım. **with** keyword'ünü kullanırsanız **.close()**'u kendisi arka planda kendi kendine çağırıyor diyeyim arkadaşlar. Burada biz Türkçe metinlerle çalıştığımız için kullanışlı bir şeyden daha bahsetmek istiyorum. **encoding='utf-8'** değerini atarsanız, içinde Türkçe karakterleri kullanabilmenizi sağlar.

Geldik nesne yönelimli programlama. Bunu da yapay zeka operasyonlarında genelde çok kullanmadan yol alabiliyoruz. Özellikle notebook'ların içerisinde hiç **class** tanımlamadan çalıştığınız case'ler olacak ama öğrenirseniz, naçizane, biraz böyle yapay zeka uygulaması haricinde bir backend tasarladığınız zaman daha derli toplu kod yazmanızı sağlar. O yüzden birazcık önemli diyebilirim açıkçası bu kısım. Burada **class** keyword'ü ile **Ogrenci** diye bir class tanımladım. Şu an için içini de hiçbir şey yazmadım. İçine hiçbir şey yazmadan böyle bir **ogrenci1** değişkeni oluşturduğum zaman, **Ogrenci** class'ını kullanarak bir öğrenci objesi oluşturmuş oluyorum.

Geldik bu **__init__** metoduna. **__init__** metodu nedir? Benim başlangıçta olmasını istediğim bir takım değişkenler var. Elimde bir öğrenci var, bu öğrencinin bir adı olmalı, bir yaşı olmalı diyelim ki. Bunu burada **self** keyword'ü ile böyle bu şekilde

tanımlıyorum. **self** keyword'ü bana neyi getiriyor? Bu adı ve yaşı alıp, içeride **self.ad**, **self.yas** dediğim zaman class'ın içinde her yerden **self.ad**, **self.yas** diye bu arkadaşımızın adını ve yaşını kullanabiliyorum.

Miras alma (inheritance). Çok sık kullanmamakla beraber kullanışlı bir yöntem yine. Burada neye getiriyor bizi? Benim **Insan** diye bir class'ım var bu sefer. Bu insanın bir adı var, bir yaşı var ve bu insanın bilgilerini göstermemi sağlayan bir **bilgileri_goster** metodum var. Ben bu **Insan** tanımından sonra bir **Ogrenci** tanımlamak istediğimde içeriye **Insan** yazdığımda artık o **Insan** objesine ait her şey, yani adı ve yaşı alabiliyorum. Bunu da **super()** keyword'ü ile yapıyorum. **super().__init__(ad, yas)** dediğimde, bir daha böyle **self.ad = ad**, **self.yas = yas** diye tanımlamama gerek kalmadan direkt olarak **Insan** sınıfındaki ad ve yas değişkenlerini kullanabiliyorum.

Geldik sonuna. Sonunda arkadaşlar, şöyle 1-2 tane tavsiye ekledim diyelim. Öğrenmenin tek yolu öğrendiklerinizi kullanmaktır. Bu bu arada burası bir ödev değil, lütfen bootcamp kapsamında bir değerlendirme falan yapmıyorum. Bir takım projeler geliştirebilirsiniz. Projelerle alakalı bazı fikirler verdim. Kodlama platformlarının linklerini eklemedim, Özge Hocam sunumunda eklemişti. Bir alan seçin ve derinleşin. Birazcık Python'da ilerledikten sonra, yani veri bilimine gireceğiz zaten bootcamp kapsamında. Web development tarafında da ilerleyebilirsiniz. Machine learning AI kapsamında gideceğiz. Python'da istediğiniz neredeyse her şeyi yapabilirsiniz arkadaşlar. Çok kısaca şunu göstereyim, sonra ufak bir ara verelim. FreeCodeCamp diye benim çok sevdiğim bir platform. Bunun böyle bir tane linkini ekledim. Python'a yeni başlayanlar için 25 farklı proje fikri var burada. Bunları incelersemiz, kendiniz yapmaya çalışırsınız, baktınız olmuyor, gidin ChatGPT'ye "bana bu kodu anlat" deyin. Size projeyi yapsın, kodu anlatsın. Anladıktan sonra dönüp başka bir örnek seçin, bu sefer onu kendiniz yapmayı deneyin. Bu şekilde kendinizi geliştirebilirsiniz diyeyim.

Saat 21:41, 21:50'de ikinci kısma geçeceğim. Sorusu olan var mı?

Soru: Colab çalışınca veri çaldırma riski yok mu?

Göker Güner: Var tabii yani, yok onu diyemem. Ama burada atıyorum, ben sizinle bu linki paylaştığımda, bu link benim. Ben sizinle bunu viewer modda paylaşıyorum. Siz sadece görüntüleyebiliyorsunuz gibi düşünebilirsiniz. Onun yani güvenliği orada. Buradaki herhangi bir değeri siz kendiniz değiştiremiyor olacaksınız arkadaşlar.

Yorum: Colab'ın güzel bir yorumu oldu. Colab bulutta çalıştığı için bilgisayar gücüne göre daha verimli ve kısa sürebiliyor, özellikle eğitim tarafı.

Soru: Colab mı, Jupyter kullanmak mı iyi? Colab daha mı iyi, kod editörü açmak mı iyi?

Göker Güner: Bizim yapacağımız projede çok bir şey fark etmiyor çünkü zaten çok küçük verilerle çalışacağız. Ama uzakta böyle deep learning falan çalıştığınız zaman, daha büyük verilerle, Colab'ı lokale tercih edin bence.

Soru: Inheritance konusuna başka nasıl bir örnek verebilirim?

Göker Güner: Hiçbir şey kalmıyor aklıma. Yani çok kullanırsınız diye değil de belki hani lazım olur diye verdim burada. İnsan değil de işte bu bir araba olabilirdi. Arabaların da işte spor araba olanı var, 4 kapılı olanı, 2 kapılı olanı var, ne bileyim. Özellikleri değişebilir arabanın yani. Bir tane burada temel bir sınıf tanımladığımız zaman, en basit tabiriyle, benim bir tane insanım var, bu insanın her türlü adı, yaşı olacak. Ya da ne bileyim, bu üretim bandındaki ürünler olabilir. Bu ürünlerin temel bir takım özellikleri olabilir. Farklı farklı ürünlerin farklı özellikleri olabilir. Ben şimdi burada benim öğrencimin de

öğretmenimin de adı, yaşı olacaksa, bu ikisi ortaksa eğer, bu ikisini ayrı ayrı tanımlamak yerine gidip hepsini özet olarak burada tanımlayıp sonra gelip öğrenci ve öğretmenin içinde miras verdirerek kullanıyorum. Kullanım olarak bir farkı var mı? Bir farkı yok. Sadece işte az önce bahsettiğim gibi, daha böyle okunaklı bir kod yazmamızı sağlıyor.

O zaman bir 3 dakika sonra görüşürüz. Ufak bir kamerayı kapatıp çay doldurup geliyorum. Sonra ikinci kısma devam ediyoruz gençler. Ufaktan geçelim mi 2. bölüme?

Temel veri kütüphanelerinden bahsedeceğiz. Yayının başında dediğim gibi, bunu bootcamp içinde zaten sürekli kullanacağımız için öyle ilk bölümdeki kadar detaylı bahsetmeyeceğim her şeyden. Sadece hangi kütüphaneyi ne için kullanıyoruz, işte ne işe yarıyor ve nasıl kullanıyoruz gibi böyle temel bir kaç metotla gösteriyor olacağım.

NumPy, nedir? Açılımı Numerical Python. Sayısal hesaplamalar için kullanacağınız temel kütüphanedir. Neden Python listeleri varken **NumPy**'a ihtiyacımız var? Az sonra göreceğiz ama büyük matematiksel hesaplamaları için Python birazcık yavaş kalıyor. **NumPy** burada biraz daha özelleştirilmiş, bu tip işlemler için özel yazılmış bir kütüphane. O yüzden sizin daha hızlı çalışmanızı sağlıyor daha büyük verilerle. Bootcamp'te en çok **Pandas** ve **Scikit-learn** kullanacaksınız kütüphane olarak. Arka planda tamamen **NumPy**'ı kullanıyor olacak. Yüklemek için **!pip install numpy**... Şu başındaki **!** (ünlem) işareti ile birlikte çalıştırırsanız yükler. Colab ortamında, işte Google gibi ortamlarda zaten ön tanımlı olarak yüklü geliyor. Dolayısıyla sizin ayrıca yüklemenize gerek kalmıyor. **NumPy**'ı **np** olarak tanımlayıp kullanıyoruz projelerde. Buna mecbur değiliz ama bu bir standarttır. **NumPy**'ı **np** diye içeriye import etmek, içeriye dahil etmek.

Gerilim **NumPy**, ufak tefek farklarına. Benim bir tane böyle 4 elemanlı bir listem var diyelim. Ben bu listeye bu haliyle **+ 5** yazamıyorum. Bir listeyi 5 ile toplayamam çünkü. İstediğim şeyi yapmak için, bunu 1. bölümde gördüğüm şekilde bir for döngüsü lazım bana. Bütün elemanları tek tek dönüp bu işlemleri yapmam için. Ama **NumPy** bizim yerimize bunu hallediyor. Ben listeyi **np.array()** diyerek bir **NumPy** listesine dönüştürüyorum. Ben bunu istediğim gibi toplayabilirim çünkü vektörel bir işlem yapar. Vektörel işlem yapması ne demek? Bütün elemanlara aynı işlemi uygulayabilmesi demek. Genelde makine öğrenimi için en çok iki boyutlu array'lere ihtiyacımız var. Matrix deniyor buna. Tek boyutlu olunca vektör diyoruz. Özel matrisler oluşturmak istediğimde, örneğin sıfırlardan oluşan bir array oluşturmak için, matris oluşturmak istediğimde özür diliyorum, böyle **3, 4** yazıp 3 satır 4 sütunlu sıfırdan oluşan bir matris oluşturmak isterseniz... Az önce bahsettiğim gibi, bazen bazı case'lerde bizim elimizdeki array'in şeklini bilmek önem taşıyor. Burada **.shape** dediğiniz zaman size şeklini verir. **.ndim** dediğiniz zaman dimension, size boyut sayısını verir, kaç boyutlu olduğunu. Ve son olarak da veri tipi ile ilgili bazen hatalar aldığınızda bunu kullanmanız gerekebiliyor. **.dtype**... Veri tipim benim **int64**.

Scikit-learn, tüm verilerinizi **NumPy** array'leri olarak istiyor. Tüm verilerinizi **NumPy** array'i olarak istiyor. Bizim bootcamp'imizde bu **NumPy**'a dönüştürme işini zaten **Pandas** bizim için yapıyor olacak.

Veriyi manipüle etmek için kullandığımız **Pandas**'a gelelim. Bootcamp'te **Pandas**'ı epeyce bir kullanacağız. Neden **NumPy** varken **Pandas**? Çünkü az önce işte oluşturduğum 1, 2, 3 gibi isimsiz bir veri dizileri verdik. Ama **Pandas** bize **DataFrame** diye bir tane özellik sunuyor. Burada işte siz Excel ya da SQL benzeri bir tablo gibi sunuyor size bu verileri. Aşağıda zaten örneklerini vereceğiz. Burada yapacağımız projede bir oylama kayıtları kullanacağız. Oylama kayıtları dediğim de, onu da kısaca açıklayım. Bir veri seti var elimizde, Amerikan kongresinde bir takım konularla alakalı verilen oylar. Bu oylara göre bu oy veren insanlar Demokrat mı, Cumhuriyetçi mi diye sınıflandıracağız. Gerçekten çok minik, basit bir proje ekledim ki hani

bir machine learning projesinde ne gibi adımlar var, bu adımları nasıl çalışırız, yeni gelecek olan bootcamp projesine nasıl hazırlanırız, bunu görmeniz için bir ön izleme vermek istedim. Yine eğer yüklü değilse **!pip install pandas** diyerek yükleyebiliyorsunuz ama Colab'da yüklü geliyor. **NumPy**'a benzer şekilde **Pandas**'ı da **pd** diye içeriye import ediyorsunuz. Bu da yine global bir standart arkadaşlar. **Pandas**'ın temel iki tane veri yapısı var. Bir tanesi az önce yukarıda bahsettiğimiz **DataFrame**, bir tanesi de Seri şeklinde isimlendirdiği bir veri tipi. Excel'deki tek bir sütun gibi düşünün. **pd.DataFrame()**... Az sonra göreceğiz, verimizi import ettiğimizde direkt **pd.DataFrame()** içeriye o veriyi yazdığınız zaman otomatik olarak **DataFrame**'e dönüşüyor olacak. Burada Jupyter, Colab'de şu anda gördüğünüz arayüzde **display** kullanırsanız daha güzel bir biçimde size formatlar. Yukarıda verdiğimiz değişkenlerin hepsini bir tablo formatında bize sunuyor.

Pandas'ta en çok kullandığımız 3 metot. Bunu Enes'in bir sonraki yayınında, Exploratory Data Analysis, EDA diye kısaltılır, keşifsel veri analizi demek, orada daha detaylı inceleyeceksiniz. Ama **.head()**, **.info()**, **.describe()** en çok kullandığımız **Pandas** metotları olacak aslında. **.head()** ilk n satırına hızlıca bakar. Default'u bunun 5'tir ama isterseniz **.head()**'in içine 3 yazarsanız ilk 3 veriyi incelemiş olursunuz. **.info()** kısmında veri setinin özetini gösterir size. Eksik verileri tespit etmenin aslında varsa en hızlı yolu. Hangi sütun var elimizde, kaç tanesi dolu, **dtype**'ları ne, tipleri ne, bunları gösterir. **.describe()** da adının üstünde, tanımlar. Ama sadece sayısal sütunları tanımlar ve temel istatistiklerini gösterir. Yani sütundaki değerlerin ortalamasını, minimumunu, maksimumunu gibi.

DataFrame'lerden de belirli sütunları almak benzer bir işlemle oluyor. **.isna().sum()** dediğin zaman hangi sütunda kaç adet eksik veri olduğunu saymanın en hızlı yolu aslında. **fillna()** adının üstünde, **NaN** olan değerleri dolduracak diyelim. Ne ile dolduracak? Buradaki değerle dolduracak. **.dropna()**, **NaN** olan değerleri drop'lamaya, o veri setinden atmaya yarıyor ama elimizdeki veriyi neden silelim, neden durduk yere daha az veriyle çalışıyor olalım? O yüzden bunları çok kullanmıyoruz diyelim.

Veri tablosunu nasıl yükleyeceğiz, nasıl anlayacağız? ML modelleri için en kritik adım olan eksik verileri nasıl temizleyeceğinizi gördük. Tabii ki eksik veri temizlemenin falan daha fazla yolu elbette var ama burada bu anlatımı yaparken maksadımız nedir? Bizim elimizde eksik veriler olabilir, bu verileri temizlememiz gerekiyor. **Pandas**'ta da bunu yapabiliyoruz. Bu temayı vermekti derdim. Bu temayı verdikten sonra bu verileri acaba başka nasıl temizleyebilirim diye kendiniz artık araştıracak noktaya gelmiş oluyorsunuz.

Veri görselleştirme. Veriyi temizledik ama anlamak için görselleştirmek faydalı oluyor. Geldik kütüphanelere. **Matplotlib**, **Seaborn**. **Matplotlib** en temeli. **Seaborn** birazcık daha böyle güzel görsellerin yer aldığı başka bir kütüphane. Bunların ikisini birlikte kullanmak zorunda değilsiniz ama genelde birlikte kullanıyoruz. Onları da bu arada varsayılan olarak **plt** ya da **sns**, Seaborn ile kısaltmaları ile projelerinize dahil ediyorsunuz arkadaşlar. Bu da yine evrensel bir gösterimdir diyelim. Burada **Seaborn**... Bu arada şey, görselleştirme metodu olarak da burada göstereceğim ben, fazlası var. **Seaborn**'un orijinal kütüphanesini açarsanız, bu dökümanda yok galiba ama linkini bulur eklerim, daha fazla sayıda görselleştirme göreceksiniz orada. Ben sadece en temellerine biraz göz aşinalığınız olsun diye göstereceğim. Yukarıda dediğim gibi bir Amerikan Kongresi'ndeki temsilcilerin Demokrat mı, Cumhuriyetçi mi olduğunu ayırt edeceğiz aslında burada. Model eğitiminden önce sormamız gereken ilk soru: Veri setim dengeli mi? Yani benim burada 450 Cumhuriyetçi, 50 Demokrat'ım varsa, herkese Cumhuriyetçi deyip %90 başarı sağlayabilirim. **countplot** metodu, bir sütundaki her bir kategorinin kaç kez geçtiğini sayan bir bar grafiği.

Görselleştirme, sonuç olarak modelimizi eğitmeden önce verimizle tanışıp olası problemleri tespit etmemizi sağlıyor.

Scikit-learn'e gelelim. Ne yaptık? Verimizi önce **DataFrame**'e dönüştürdük, oluşturduk. Görselleştirdik. Veriyi artık eğitme kısmına geliyoruz. **Scikit-learn** nedir? Python'da machine learning, makine öğrenmesi yapmak için kullanılan endüstri standardı bir kütüphane. Bu hakikaten gerçekten endüstri standardı. Şirketlerde de ML projelerinde bunlar kullanılıyor. Bizim ML projemizin 5 adımlık temel iş akışını öğrenmeye odaklanacağız. Bu akış, hangi modeli kullanırsanız kullanın, %99 aynı kalacak. Nedir onlar? Verimizi hazırlayacağız, verimizi böleceğiz, modelimizi seçeceğiz, modelimizi eğiteceğiz ve modelimizi değerlendireceğiz. Hangi veri seti üzerinde çalışsanız çalışın, hangi modeli seçerseniz seçin, nasıl bir proje... İşte sınıflandırma, supervised, unsupervised falan filan, hangi algoritmayla, nasıl bir proje yaparsanız yapın, bu akışlar %99 aynen kalır arkadaşlar. **Scikit-learn**'i nasıl kullanıyoruz? **sklearn.model_selection**'dan **train_test_split** fonksiyonunu kütüphaneden direkt çağırabiliyorsunuz. Geldik modelimizi nasıl train ediyoruz. Burada bir tane **KNeighborsClassifier** diye bir modelimiz var. K-En Yakın Komşu modeli. Bu modeli kısaca **neighbors** sınıfından import ettikten sonra, gittim işte "en yakın 3 komşusuna bak" dedim. **model** değişkenime atadım. **model.fit()** dediğimde train verilerini verdiğimde ben modelimi eğitmiş oluyorum arkadaşlar. Yani nedir? Modelimi veriye fit etmiş oluyorum. Modelim bu veriyi öğrenmiş oluyor. Değerlendir dediğimizde de, değerlendirirken tabii test verisiyle değerlendireceğiz. **model.predict(X_test)** dediğim zaman, bunu **y_tahminleri**'ne atarsanız eğer...

Geldik temel bir ML projesi. Bu son modül, bu arada burada bitiriyorum. Bu modülde, 2. kısımda öğrendiğimiz her şeyi basit bir ML projesi için kullanacağız. Çok bunun da detayına girmeyeceğim çünkü zaten hem yukarıda bahsettim hem de zaten çalıştırınca "okey, böyle çalışıyormuş" gibi bir kenara koyacaksınız. Sonra proje yayını geldiğinde, yavaş yavaş proje yapma kısmına geçtiğinizde dönüp tekrar tekrar o aşamaların üstünden geçiyor olursunuz.

Uçtan uca çalıştırmak için tabii yani siz buranın dışında çalıştığınızı varsayıyorum. **pandas**, **os**, **matplotlib**, **seaborn**, bunların hepsini import etmek durumundasınız ama biz zaten yukarıda import ettiğimiz için ben tekrar import etmedim. **train_test_split**, çalıştıracağınız modelinizi, **accuracy_score**'u, **confusion_matrix**'i import ediyorsunuz. Yukarıda yaptığımız adımların her birini direkt tekrar ediyoruz aslında. Önce veri setini çekiyoruz... Adım 4'e geldik, artık direkt iş akışımızı uygulamak. Burada ne yapıyoruz? **parti** değişkenini **DataFrame**'den drop ediyorum. Hedef değişkenim bu çünkü. Kalan herkesi **X**'e atıyorum. **parti** değişkenini de **y**'ye atıyorum. Veriyi böldüm, 70-30 böldüm. Modeli seçerken bu sefer en yakın 5 komşuya bakıyorum. Modelim datamı fit ediyor. Ardından test verilerini predict ediyorum. Tahminlerimi **y_tahminleri**'ne atıyorum. Ve tahminlerimi **y_test** ile kıyasladığımızda neymiş? %93.1 başarıyla tahmin edebiliyormuşuz.

Proje odasında kısaca ne yaptığımızdan bahsettim. Sonra iki tane kaynak koydum ama bu kaynakları artırabilirim. Bu **Scikit-learn**'ün dökümanını falan ekleyeceğim oraya.

Özge Usta: Bayağı dolu dolu bir eğitim oldu umarım, faydalı olmuştur.

Göker Güner: Vallahi ben de dinledim, güzel oldu. Herkesin kafasını karıştırdık, umarım herhalde...

Özge Usta: Ya dediğin gibi, aslında hani biz bunları tabii birkaç saate sığdırmaya, anlatmaya çalışıyoruz ama mutlaka özet geçmek kaçınılmaz oluyor.

Göker Güner: Evet. Biz sorulara bakabiliriz istiyorsan.

Soru: Gelişmiş fonksiyonel EDA hakkında...

Göker Güner: Enes'in yayınında bu keşifsel veri analizine daha detaylı değineceğiz. Orada hatta Enes şeyi de göstersin. Kendisi ChatGPT ile, şu anda Ahmet'in de EDA ile ilgili bir projesi var. Böyle otomatik o süreçleri yapan, "auto EDA" kütüphanesi... Yayında ondan da bahsetsin.

Soru: Kütüphaneler, daha kütüphaneler dediğimiz bu ikinci kısımda gösterdiklerin, misal daha detaylı olacak mı?

Göker Güner: Çünkü ya şöyle, daha detaylı olacak... Zaten hem derslerde hem de yapacağınız projede daha detaylı kullanıyor olacağız bu kütüphaneleri. O yüzden her ders neredeyse bu kütüphanelerin üzerinden tekrar tekrar geçiyor olacağız.

Soru: Sıkça kullandığınızı söylemişsiniz, nasıl kullandığınızı anlatabilir misiniz?

Göker Güner: Kod editörü olarak Cursor ve Kira kullanıyorum. Bunun dışında asistan olarak da ChatGPT'yi kullanıyorum. Şirkette Claude üyeliğimiz var. Orada işte bir proje yapacağım zaman Claude'a bir proje session'u oluşturup, işte bu context engineering dediğimiz olay var ya, biraz ona giriyor. Dökümanlarımı yüklüyorum, ne yapmasını istediğimi anlatıyorum, sonra beraber bu projeyi hayata geçiriyoruz.

Soru: Eğitim sonundaki proje hakkında bilgi verir misiniz?

Göker Güner: Orada bunu asıl sonraki bir yayınımızda zaten detaylı anlatacağız. Aynen öyle, yani Enes anlatacak o yayında. Oylama projesi, burada bahsettiğimiz halinden yola çıkarak bir proje yapacaksınız.

Yavaştan kapatalım, benim sesim iyice kısıldı isterseniz. O zaman herkese iyi akşamlar.

Özge Usta: Teşekkürler, ağzına sağlık. Görüşmek üzere, iyi akşamlar.