İçindekiler

[Intro to Machine Learning 3](#_Toc45127876)

[How Models Work (Modeller Nasıl Çalışır?) 3](#_Toc45127877)

[Giriş 3](#_Toc45127878)

[Decision Tree’nin Geliştirilmesi 4](#_Toc45127879)

[**Basic Data Exploration (Basit Veri Keşfi)** 6](#_Toc45127880)

[Verilerinizi Tanımak için Pandas Kullanımı 6](#_Toc45127881)

[Interpreting Data Description (Verilerin Yorumlanması) 7](#_Toc45127882)

[Excercise: Explore Your Data 8](#_Toc45127883)

[**Your First Machine Learning Model** 10](#_Toc45127884)

[Selecting Data for Modeling (Modelleme için Veri Seçmek) 10](#_Toc45127885)

[Choosing "Features" (Özellik Seçimi) 11](#_Toc45127886)

[Building Your Model (Model Oluşturma) 13](#_Toc45127887)

[Exercise: Your First Machine Learning Model 15](#_Toc45127888)

[**Model Validation (Model Geçerliliği)** 19](#_Toc45127889)

[Model Validation Nedir? 19](#_Toc45127890)

[The Problem with "In-Sample" Scores 22](#_Toc45127891)

[Coding It 22](#_Toc45127892)

[Wow! 23](#_Toc45127893)

[Exercise: Model Validation 23](#_Toc45127894)

[**Underfitting and Overfitting** 28](#_Toc45127895)

[Farklı Modellerle Deneme 28](#_Toc45127896)

[Examples 30](#_Toc45127897)

[Sonuç 31](#_Toc45127898)

[Exercise: Underfitting and Overfitting 32](#_Toc45127899)

[**Random Forests** 34](#_Toc45127900)

[Introduction 34](#_Toc45127901)

[Example 34](#_Toc45127902)

[Sonuç 35](#_Toc45127903)

[Exercises: Random Forest 36](#_Toc45127904)

[**Exercises: Machine Learning Competitions** 38](#_Toc45127905)

[Introduction 38](#_Toc45127906)

[Creating a Model For the Competition 40](#_Toc45127907)

[Make Predictions 40](#_Toc45127908)

[Quiz: Intro to Machine Learning 41](#_Toc45127909)

[Intermediate Machine Learning 47](#_Toc45127910)

[Introduction 47](#_Toc45127911)

[Exercises 48](#_Toc45127912)

[Step 1 : Eveluate Several Models (Birkaç modeli değerlendirin) 48](#_Toc45127913)

[Step 2: Generate Test Prediction (Test tahminleri oluşturun) 50](#_Toc45127914)

[Missing Values (Eksik Veriler) 50](#_Toc45127915)

[Üç Yaklaşım 50](#_Toc45127916)

[Example 51](#_Toc45127917)

[Conclusion 54](#_Toc45127918)

[Exercises 55](#_Toc45127919)

[Categorical Variables 61](#_Toc45127920)

[Kaynaklar 62](#_Toc45127921)

# Intro to Machine Learning

Makine öğrenmesindeki temel fikirleri öğrenin ve ilk modellerinizi oluşturun.

## How Models Work (Modeller Nasıl Çalışır?)

Giriş

Makine öğrenimi modellerinin nasıl çalıştığına ve nasıl kullanıldıklarına genel bir bakışla başlayacağız. Daha önce istatistiksel modelleme veya makine öğrenimi yaptıysanız bu temel görünebilir. Endişelenmeyin, yakında güçlü modeller oluşturmaya devam edeceğiz.

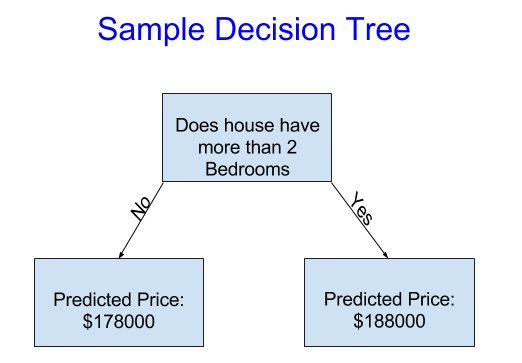
Bu mikro kurs, aşağıdaki senaryodan geçerken modeller oluşturmanızı sağlayacaktır:

Kuzeniniz gayrimenkul konusunda spekülasyonlarla milyonlarca dolar kazandı. Veri bilimine gösterdiğiniz ilgi nedeniyle sizinle iş ortağı olmayı teklif etti. Parayı tedarik edecek ve çeşitli evlerin ne kadar değerli olduğunu tahmin eden modeller sunacaksınız.

Kuzeninize geçmişte gayrimenkul değerlerini nasıl tahmin ettiğini soruyorsunuz. Ve bunun sadece sezgi olduğunu söylüyor. Ancak daha fazla sorgulama, geçmişte gördüğü evlerden fiyat örüntülerini belirlediğini ve bu kalıpları düşündüğü yeni evler için tahminler yapmak için kullandığını ortaya koyuyor.

Makine öğrenimi de aynı şekilde çalışır. Decision Tree adlı bir modelle başlayacağız. Daha doğru tahminler veren meraklı modeller var. Ancak Decision Tree’lerin anlaşılması kolaydır ve bunlar veri bilimindeki en iyi modellerin bazıları için temel yapı taşıdır.

Basitlik için, mümkün olan en basit karar ağacıyla başlayacağız.



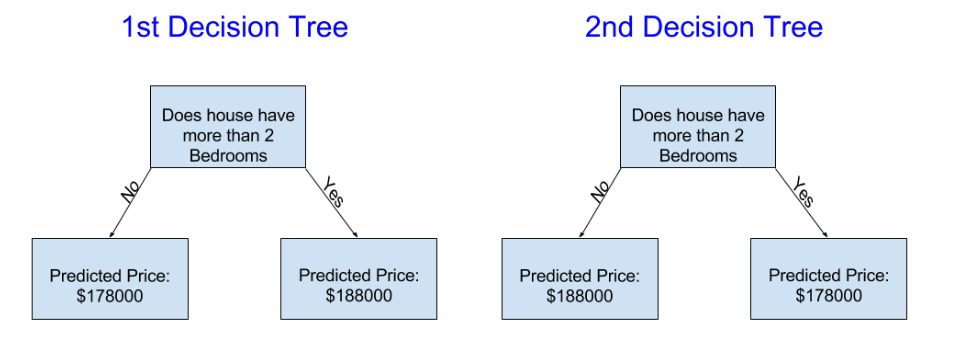
Evleri sadece iki kategoriye ayırır. Dikkate alınan herhangi bir ev için tahmini fiyat, aynı kategorideki evlerin tarihsel ortalama fiyatıdır.

Verileri, evlerin iki gruba nasıl ayrılacağına karar vermek için ve sonra her grupta öngörülen fiyatı belirlemek için kullanıyoruz. Verilerden pattern yakalamanın bu adımına, modelin fit edilmesi(**fitting**) veya train edilmesi(**training**) denir. Modelin **fit** edilmesi için kullanılan verilere **training data** denir.

Modelin nasıl **fit** edildiğine dair ayrıntılar (örneğin, verilerin nasıl bölüneceği) daha sonra kullanmak üzere kayıt edeceğimiz kadar karmaşıktır. Model **fit** edildikten sonra, yeni evlerin fiyatlarını **predict** edebilmek için yeni verilere uygulayabilirsiniz.

Decision Tree’nin Geliştirilmesi

Aşağıdaki iki karardan hangisinin gayrimenkul eğitim verilerinin fit edilmesinden kaynaklanması daha olasıdır?



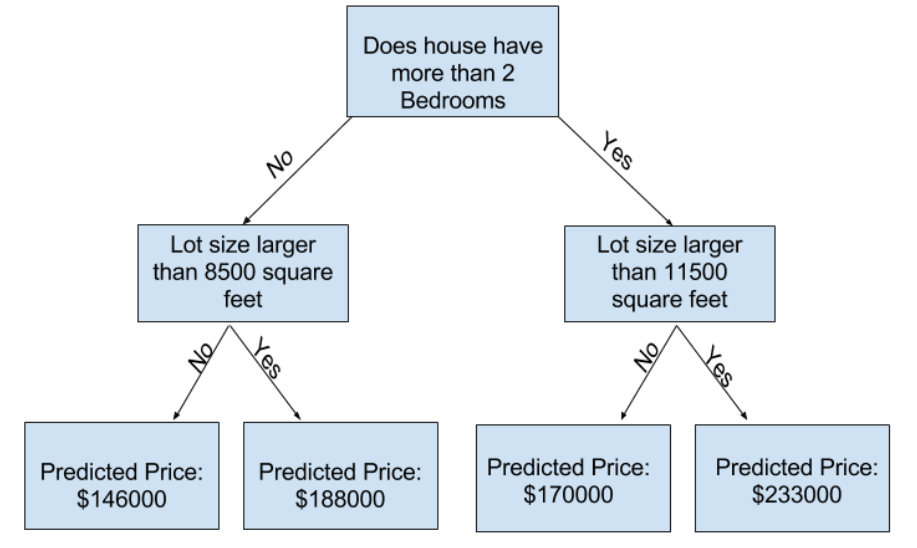
Soldaki karar ağacı (Decision Tree 1) muhtemelen daha mantıklıdır, çünkü daha fazla yatak odası olan evlerin daha az yatak odası olan evlerden daha yüksek fiyatlarla satılma eğiliminde olduğu gerçeğini yakalar.

Bu modelin en büyük eksikliği, banyo sayısı, lot büyüklüğü, konum vb. gibi ev fiyatını etkileyen çoğu faktörü yakalamamasıdır.

Daha fazla "splits(bölme)" olan bir ağaç kullanarak daha fazla faktör yakalayabilirsiniz.

Bunlara "deeper(daha derin)" ağaçlar denir.

Her evin toplam lot büyüklüğünü de dikkate alan bir karar ağacı şöyle görünebilir:



Herhangi bir evin fiyatını karar ağacından takip ederek, her zaman o evin özelliklerine karşılık gelen yolu seçerek tahmin edersiniz.

Ev için tahmini fiyat ağacın altındadır.

Altta tahmin yaptığımız noktaya **leaf**(yaprak) denir.

Yapraklardaki splits(bölünmeler) ve values(değerler) veriler tarafından belirlenecektir, bu nedenle çalışacağınız verileri kontrol etmenin zamanı geldi.

**Basic Data Exploration (Basit Veri Keşfi)**

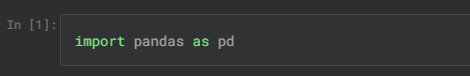
Verilerinizi Tanımak için Pandas Kullanımı

Herhangi bir makine öğrenimi projesinin ilk adımı, verileri tanımaktır.

Bunun için Pandas kütüphanesini kullanacaksınız.

Pandas, bilim insanlarının verileri keşfetmek ve işlemek için kullandığı temel araç verisidir.

Çoğu kişi kodlarında pandas’ı **pd** olarak kısaltır. Bunu şu komutla yapıyoruz:



Pandas kütüphanesinin en önemli kısmı DataFrame'dir.

Bir DataFrame, tablo olarak düşünebileceğiniz veri türünü tutar. Bu, Excel'deki bir sayfaya veya SQL veritabanındaki bir tabloya benzer.

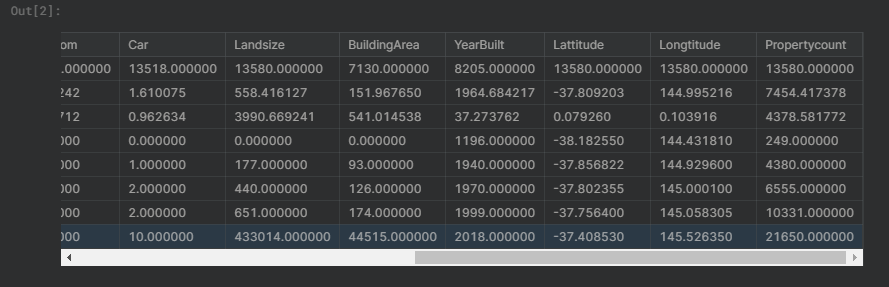
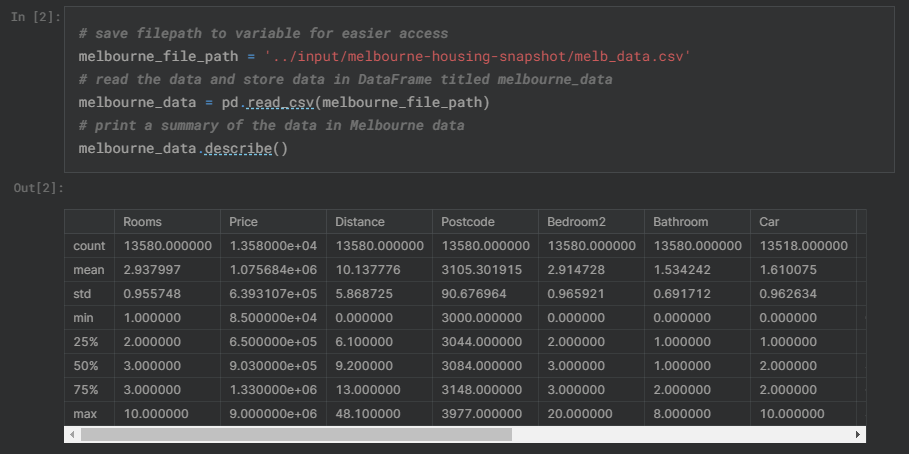
Pandas, bu tür verilerle yapmak isteyeceğiniz birçok şey için güçlü yöntemlere sahiptir.

Örnek olarak, Avustralya, Melbourne'daki ev fiyatları hakkındaki verilere bakacağız. (<https://www.kaggle.com/dansbecker/melbourne-housing-snapshot>)

Uygulamalı alıştırmalarda, aynı işlemleri Iowa'da ev fiyatları olan yeni bir veri kümesine uygulayacaksınız.

Örnek (Melbourne) verileri ../input/melbourne-housing-snapshot/melb\_data.csv dosya yolundadır.

Verileri aşağıdaki komutlarla yükler ve keşfederiz:



Interpreting Data Description (Verilerin Yorumlanması)

Sonuçlar, orijinal veri kümenizdeki her column(sütun) için 8 sayı gösterir.

İlk sayı, **count**, kaç satırın eksik olmayan değerleri olduğunu gösterir.

Eksik değerler birçok nedenden dolayı ortaya çıkar.

Örneğin, 1 yatak odalı bir ev araştırılırken 2. yatak odasının boyutu toplanmaz.

Eksik veriler konusuna geri döneceğiz.

İkinci değer, **mean** olan ortalamadır.

Bunun altında **std**, değerlerin sayısal olarak ne kadar yayıldığını ölçen standart sapmadır.

**Min**, **% 25, % 50, % 75 ve max** değerlerini yorumlamak için, her sütunu en düşükten en yüksek değere doğru sıraladığınızı düşünün.

İlk (en küçük) değer min.

Listenin dörtte birini geçerseniz, değerlerin % 25'inden daha büyük ve değerlerin % 75'inden daha küçük bir sayı bulacaksınız.

Bu **% 25** değeridir ("25. percentile" olarak telaffuz edilir). 50. ve 75. yüzdelikler benzer şekilde tanımlanır ve **max** en büyük sayıdır.

Excercise: Explore Your Data

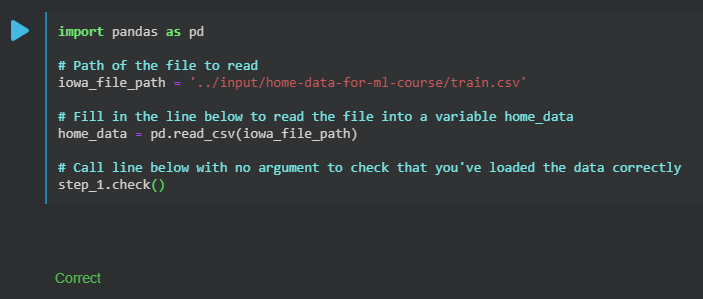
Bu alıştırma, bir veri dosyasını okuma ve verilerle ilgili istatistikleri anlama yeteneğinizi test edecektir.

Daha sonraki alıştırmalarda, verileri filtrelemek, bir makine öğrenme modeli oluşturmak ve modelinizi yinelemeli olarak geliştirmek için teknikler uygulayacaksınız.

Kurs örnekleri Melbourne'den gelen verileri kullanır. Bu teknikleri kendi başınıza uygulayabilmeniz için, bunları yeni bir veri kümesine (Iowa'dan konut fiyatları) uygulamanız gerekecektir.

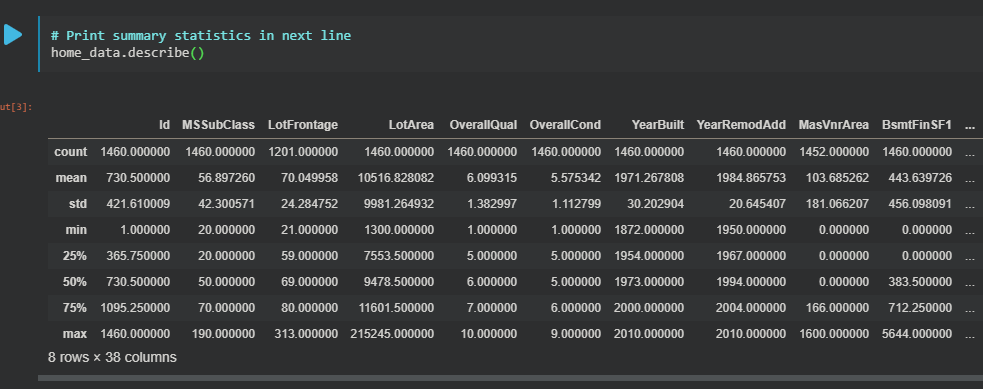
**Step 1: Loading Data (Veri Yükleme)**

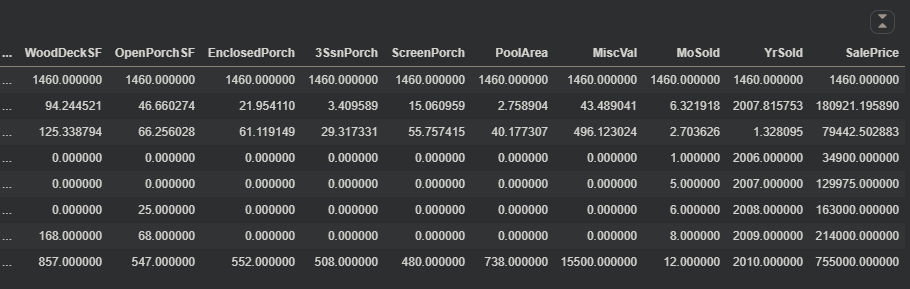
Iowa veri dosyasını home\_data adlı bir Pandas DataFrame'de okuyun.

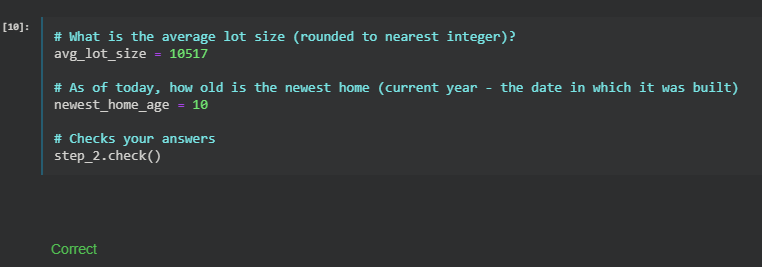


**Step 2: Review The Data (Verileri Gözden Geçirme)**

Verilerin özet istatistiklerini görüntülemek için öğrendiğiniz komutu kullanın. Ardından aşağıdaki soruları cevaplamak için değişkenleri doldurun







**Verilerinizi Düşünün**

Verilerinizdeki en yeni ev o kadar yeni değil. Bunun için birkaç potansiyel açıklama:

1- Bu verilerin toplandığı yeni evler inşa etmediler.

2- Veriler uzun zaman önce toplanmıştır. Veri yayımından sonra inşa edilen evler görünmezdi.

Nedeni yukarıdaki 1. açıklama ise, bu, bu verilerle oluşturduğunuz modele olan güveninizi etkiler mi? 2. neden ise ne olur?

Hangi açıklamanın daha mantıklı olduğunu görmek için verileri nasıl inceleyebilirsiniz?

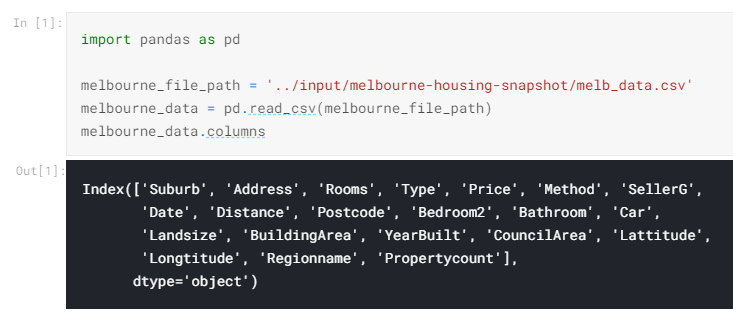
**Your First Machine Learning Model**

Selecting Data for Modeling (Modelleme için Veri Seçmek)

Veri kümenizin, kafanızda canlanması veya güzelce ekrana yazdırmak için çok fazla değişkeni vardı. Bu başa çıkılamaz veri miktarını anlayabileceğiniz bir şeye nasıl ayırabilirsiniz?

Sezgimizi kullanarak birkaç değişken seçerek başlayacağız. Daha sonraki kurslar, değişkenleri otomatik olarak önceliklendirmek için istatistiksel teknikleri gösterecektir.

Değişkenleri / sütunları seçmek için veri kümesindeki tüm sütunların bir listesini görmemiz gerekir. Bu, DataFrame’in **columns** özelliği ile yapılır. (Aşağıdaki kodun alt satırı.)



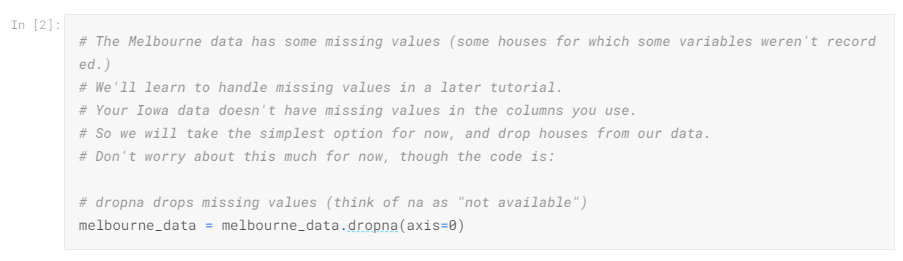
# Melbourne verilerinin bazı eksik değerleri vardır (bazı değişkenlerin kaydedilmediği bazı evler.)

# Daha sonraki bir derste eksik değerleri ele almayı öğreneceğiz.

# Iowa verileriniz, kullandığınız sütunlarda eksik değerlere sahip değildi.

# Şimdilik en basit seçeneği alacağız ve verilerimizden eksik değere sahip evleri düşüreceğiz.

# dropna eksik değerleri düşürüyor (na'yı "mevcut değil" olarak düşünün)



Verilerinizin bir alt kümesini seçmenin birçok yolu vardır. Pandas Micro-Course (<https://www.kaggle.com/learn/pandas>) bunları daha derinlemesine ele alıyor, ancak şimdilik iki yaklaşıma odaklanacağız.

1. "Prediction Target(Tahmin hedefi)"’ni seçmek için kullandığımız nokta gösterimi(dot notation)
2. "Features(Özellikleri)" seçmek için kullandığımız bir sütun listesiyle seçim yapma

**Selecting The Prediction Target (Tahmin Hedefini Seçme)**

**dot-notation** ile bir değişkeni(column) veri setinden çekebilirsiniz. Bu tek sütun, genel olarak yalnızca tek bir column’a sahip DataFrame benzeri bir **Seri**de depolanır.

Tahmin etmek istediğimiz column’u seçmek için dot-notation kullanacağız, buna **prediction target** (tahmin hedefi) denir.

Kural olarak, prediction target (tahmin hedefi) **y** olarak adlandırılır.

Melbourne'deki ev fiyatlarını (price) kaydetmek için gereken kod.



Choosing "Features" (Özellik Seçimi)

Modelimize girilen sütunlara (ve daha sonra tahminlerde kullanılan sütunlara) "features (özellikler)" denir.

Bizim durumumuzda, bunlar ev fiyatını belirlemek için kullanılan sütunlar olacaktır.

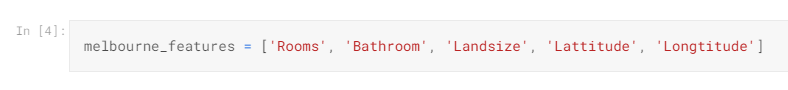
Bazen, target(hedef) hariç tüm sütunları feature(özellik) olarak kullanırsınız. Diğer zamanlarda daha az özellik ile daha iyi olacaksınız.

Şimdilik, sadece birkaç özelliğe sahip bir model oluşturacağız.

Daha sonra, farklı özelliklerle oluşturulan modellerin nasıl tekrarlanacağını ve karşılaştırılacağını göreceksiniz.

Köşeli parantez içine sütun adlarının listesini yazarak birden fazla özellik seçiyoruz. Bu listedeki her öğe bir string (tırnak işaretli) olmalıdır.

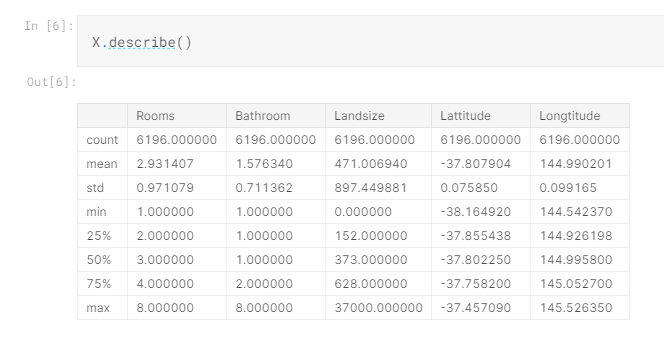
Here is an example:

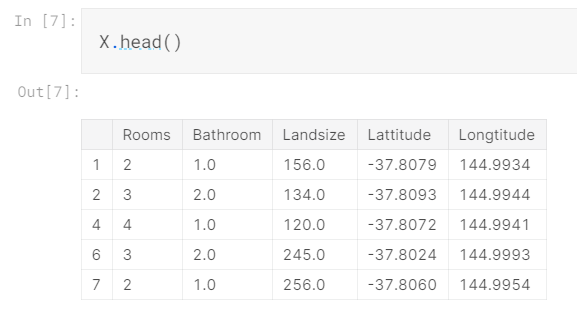


Kural olarak, bu verilere X denir.



En üstteki birkaç satırı gösteren **head** yöntemini ve **describe** yöntemini kullanarak konut fiyatlarını tahmin etmek için kullanacağımız verileri hızlı bir şekilde inceleyelim.





Verilerinizi bu komutlarla görsel olarak kontrol etmek, bir veri bilim insanının işinin önemli bir parçasıdır. Veri kümesinde sıklıkla daha fazla incelemeyi hak eden sürprizler bulacaksınız.

Building Your Model (Model Oluşturma)

Modellerinizi oluşturmak için **scikit-learn** kütüphanesini kullanacaksınız.

Kodlama yaparken, bu kütüphane örnek kodda göreceğiniz gibi **sklearn** olarak yazılır.

Scikit-learn, tipik olarak DataFrames'da depolanan veri türlerini modellemek için en popüler kütüphanedir.

Bir model oluşturma ve kullanma adımları:

* **define :** Ne tür bir model olacak? Karar ağacı mı? Başka bir model mi? Model tipinin diğer bazı parametreleri de belirtilir.
* **fit :** Sağlanan verilerden pattern(desen) yakalayın. Bu modellemenin kalbidir.
* **predict :** Tahmin
* **evaluate :** Modelin tahminlerinin ne kadar doğru olduğu belirleyin.

İşte **scikit-learn** ile bir **Decision Tree**(Karar Ağaçları)modelini tanımlama ve modeli feature’lara ve target değişkene **fit** etme örneği.

* Modeli tanımlayın. Her çalıştırmada aynı sonuçları sağlamak için random\_state için bir sayı belirtin



**random\_state:** Kodu her çalıştırdığımızda aynı çıktıyı alabilmek için girdiğimiz bir ifade. Örneğin, validation ve training olarak datayı ayırırken Python her seferinde datayı farklı yerlerinden böler, bir random state değeri belirlediğimizde de her çalıştırdığımızda aynı şekilde bölmüş olur ve aynı sonucu vermiş olur. Farklı değerler verdiğinde farklı sonuçlar aldığını göreceksin.

En iyi karar ağacını bulma problemi NP-Complete olarak sınıflandırılan problemlerdendir. Bu tip problemlerin çözümlerinde sezgisel algoritmalar kullanılır. Sezgisel algoritmalarda her kullanıldıklarında en iyi çözümü bulabileceklerini garanti etmezler ve her seferinde farklı sonuçlar üretirler. Dolayısıyla her ağaç inşa ettiğinde ağaç yapısı değişiklik gösterecektir. Modeli her çalıştırdığında aynı ağacı elde etmek istersen **random\_state** parametresini bir tamsayıya eşitlemen gerekir. Hangi tamsayıya eşitlediğinin bir önemi yok .

Birçok makine öğrenimi modeli, model eğitiminde bazı rasgeleliklere izin verir.

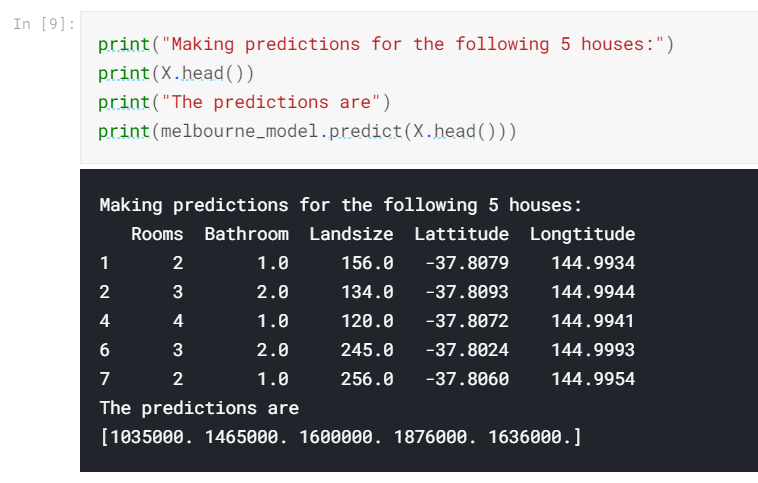
**Random\_state** için bir sayı belirtmek, her çalıştırmada aynı sonuçları almanızı sağlar. Bu iyi bir uygulama olarak kabul edilir.

Herhangi bir sayı kullanabilirsiniz ve model kalitesi tam olarak hangi değeri seçtiğinize bağlı olmayacaktır.

Şimdi tahminler yapmak için kullanabileceğimiz uygun bir modelimiz var.

Uygulamada, halihazırda fiyatlarımız olan evler yerine piyasaya çıkan yeni evler için tahminler yapmak isteyeceksiniz.

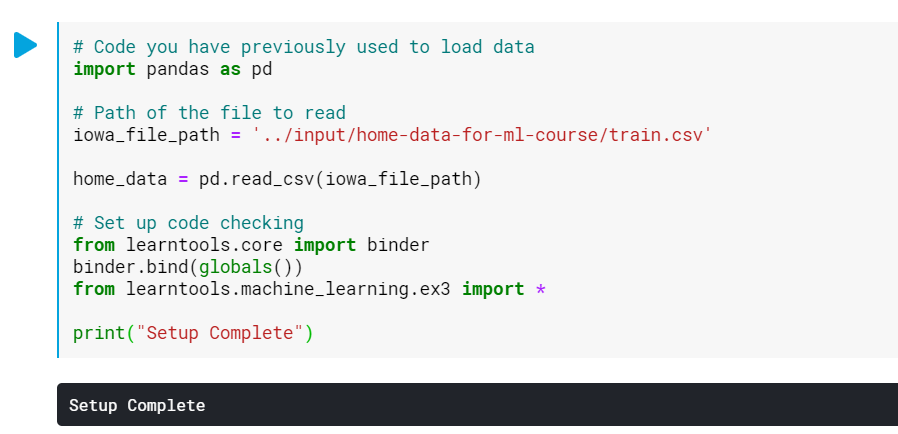
Ancak, tahmin işlevinin nasıl çalıştığını görmek için egzersiz verilerinin ilk birkaç satırı için tahminler yapacağız.



Exercise: Your First Machine Learning Model

**Özet**

Şimdiye kadar, verilerinizi yüklediniz ve aşağıdaki kodla incelediniz. Önceki adımı bıraktığınız yerde kodlama ortamınızı ayarlamak için bu hücreyi çalıştırın.



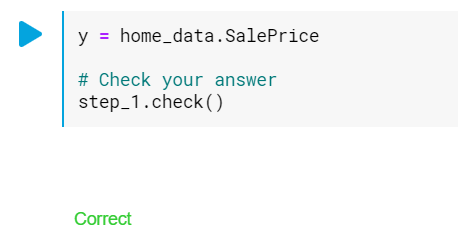
**Exercises**

**Step 1: Prediction Target Belirleme**

Satış fiyatına karşılık gelen hedef değişkeni seçin. Bunu y adlı yeni bir değişkene kaydedin. İhtiyacınız olan sütunun adını bulmak için sütunların bir listesini yazdırmanız gerekir.



Prediction Target’i y’ye tanımladık.

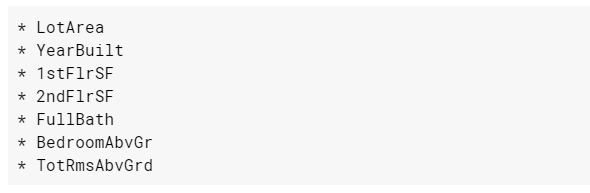


**Step 2: X Oluştur**

Şimdi, predictive feature’ları (tahmin özelliklerini) tutan X adında bir DataFrame oluşturacaksınız.

Orijinal verilerden yalnızca bazı sütunlar istediğiniz için, önce X'de istediğiniz sütunların adlarını içeren bir liste oluşturacaksınız.

Listede yalnızca aşağıdaki sütunları kullanacaksınız :

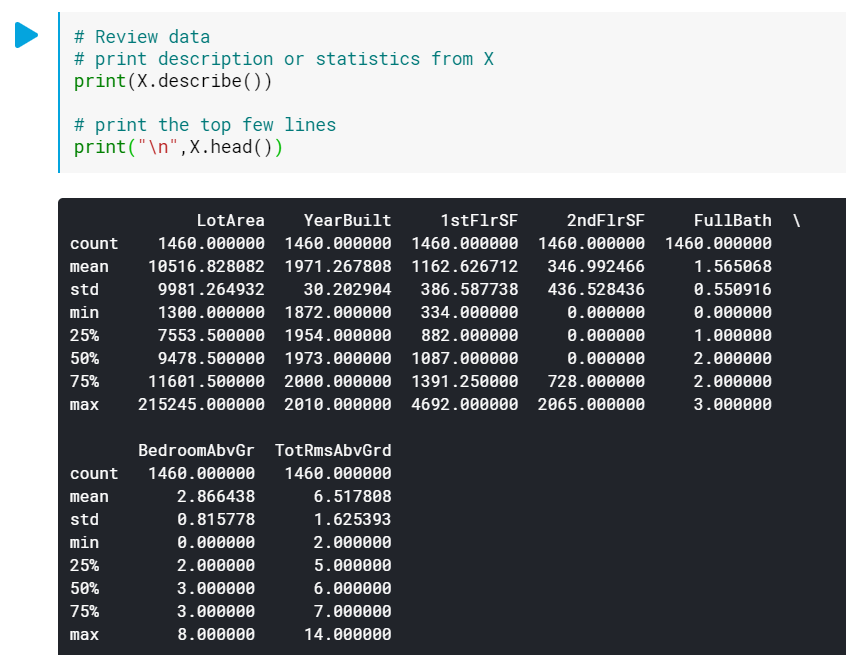


Bu özellik listesini oluşturduktan sonra, modeli fit etmek için kullanacağınız DataFrame'i oluşturmak için kullanın.



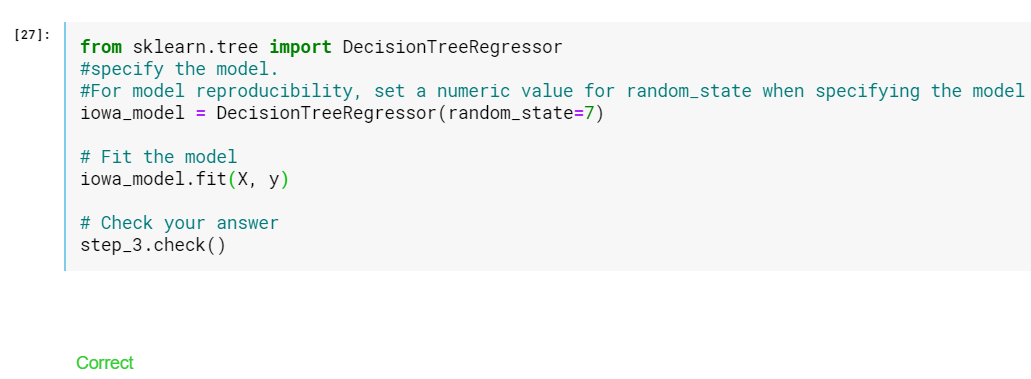
**Verinin İncelenmesi**

Bir model oluşturmadan önce, mantıklı göründüğünü doğrulamak için X'e hızlı bir göz atın.

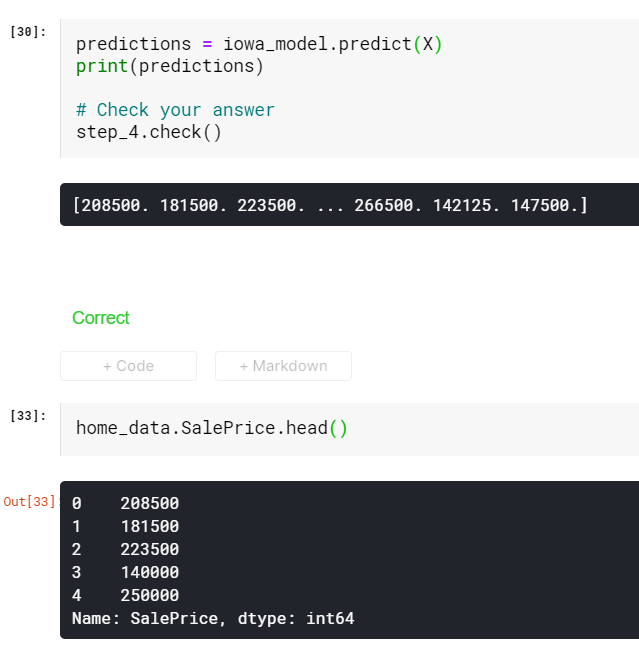
**Step 3: Modelin belirlenmesi ve fit edilmesi**

**DecisionTreeRegressor** oluştur ve iowa\_model’e kaydet. Bu komutu çalıştırmak için **sklearn’de** ilgili import işlemini yaptığınızdan emin olun.



**Step 4: Tahmin Yapma**

Veri olarak **X**'i kullanarak modelin **predict** komutuyla tahminler yapın. Sonuçları **predictions** adı verilen bir değişkene kaydedin.



**Model Validation (Model Geçerliliği)**

Bir model oluşturdunuz. Ama bu model ne kadar iyi?

Bu derste, modelinizin kalitesini ölçmek için model validation(model doğrulamayı) kullanmayı öğreneceksiniz. Model kalitesini ölçmek, modellerinizi tekrar tekrar geliştirmenin anahtarıdır.

Model Validation Nedir?

Oluşturduğunuz hemen hemen her modeli değerlendirmek isteyeceksiniz.

Çoğu uygulamada, model kalitesiyle ilgili ölçü **predictive accuracy**(tahmini doğruluk)’dir.

Başka bir deyişle, modelin tahminleri gerçekte olana yakın olacak mı?

Birçok kişi, tahmin doğruluğunu ölçerken büyük bir hata yapar.

Training data ile tahmin yaparlar ve bu tahminleri training data’daki hedef değerlerle karşılaştırırlar.

Bu yaklaşımla ilgili sorunu ve bir anda nasıl çözüleceğini göreceksiniz, ancak önce bunu nasıl yapacağımızı düşünelim.

Önce model kalitesini anlaşılır bir şekilde özetlemeniz gerekir.

10.000 ev için tahmini ve gerçek ev değerlerini karşılaştırırsanız, muhtemelen iyi ve kötü tahminlerin bir karışımını bulacaksınız.

10.000 tahmini ve gerçek değerin listesine bakmak anlamsız olacaktır. Bunu tek bir metrikte özetlememiz gerekiyor.

Model kalitesini özetlemek için birçok metrik var, ancak **Mean Absolute Error** (Ortalama Mutlak Hata) (MAE olarak da adlandırılır) ile başlayacağız.

Son sözcükten başlayarak bu metriği inceleyelim, error.

Her ev için tahmin hatası:



hata = gerçek değer – tahmin edilen değer

Yani, bir ev 150.000 dolara mal olduysa ve 100.000 dolara mal olacağını tahmin ederseniz, hata 50.000 dolar olacaktır.

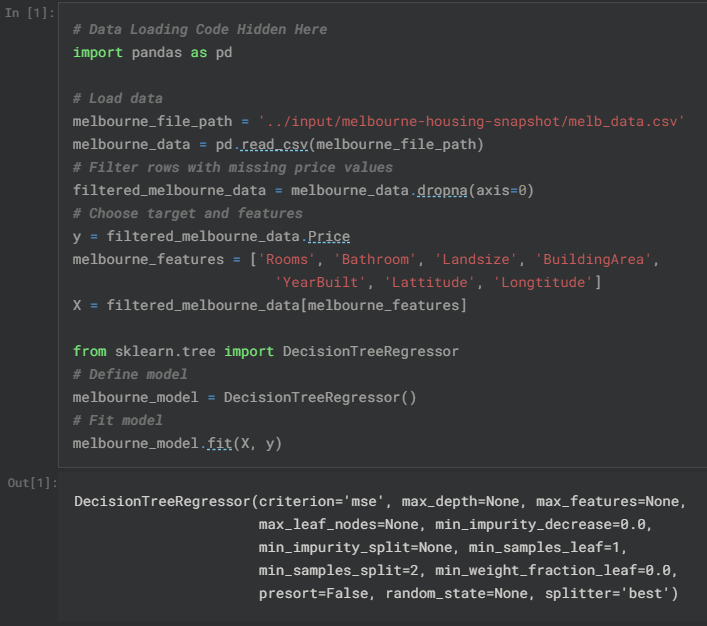
MAE metriğiyle, her bir hatanın mutlak değerini alırız. Bu, her hatayı pozitif bir sayıya dönüştürür.

Daha sonra bu mutlak hataların ortalamasını alırız.

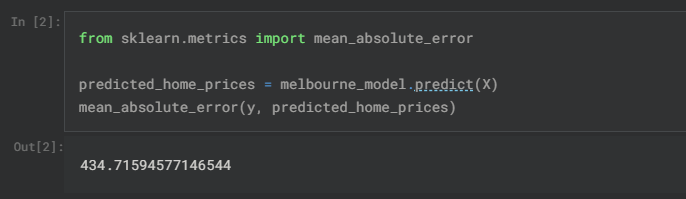
Bu bizim model kalitesi ölçümüzdür. Sade bir dille şöyle denilebilir ;

*Ortalama olarak, tahminlerimiz yaklaşık X civarında.*

MAE'yi hesaplamak için önce bir modele ihtiyacımız var.



Bir modelimiz olduğunda, ortalama mutlak hatayı şu şekilde hesaplıyoruz:



The Problem with "In-Sample" Scores

Yeni hesapladığımız ölçüme "in-sample" score’u denilebilir. Hem modeli oluşturmak hem de değerlendirmek için tek bir "sample (örnek)" ev kullandık. Bu yüzden bu kötü bir tercihti.

Büyük emlak piyasasında kapı renginin ev fiyatıyla ilgisi olmadığını düşünün.

Ancak, modeli oluşturmak için kullandığınız veri örneğinde, yeşil kapıya sahip tüm evler çok pahalıydı.

Modelin işi, ev fiyatlarını tahmin eden pattern’ler bulmaktır, bu yüzden bu pattern’i görecek, ve her zaman yeşil kapılı evler için yüksek fiyatları tahmin edecektir.

Bu model training data’dan türetildiği için, model training datalarında doğru görünecektir.

Ancak, model yeni veriler gördüğünde bu pattern(örüntü) tutmazsa, model pratikte kullanıldığında çok inaccurate(yanlış) olur.

Modellerin pratik değeri yeni veriler üzerinde tahminler yapmaktan geldiğinden, modeli oluşturmak için kullanılmayan verilerdeki performansı ölçeriz.

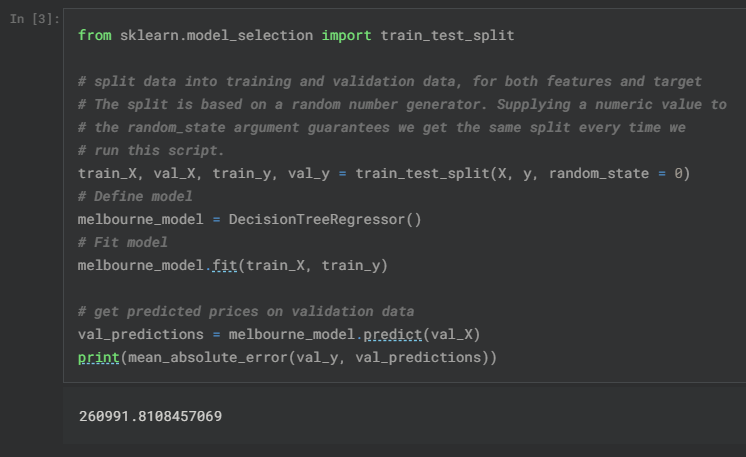
Bunu yapmanın en basit yolu, bazı verileri model oluşturma sürecinden hariç tutmak ve daha sonra bunları, daha önce görmediği veriler üzerinde modelin doğruluğunu test etmek için kullanmaktır.

Bu verilere **validation data** (doğrulama verisi) denir.

Coding It

Scikit-learn kütüphanesi, verileri iki parçaya bölmek için **train\_test\_split** fonksiyonuna sahiptir.

Bu verilerin bir kısmını modeli fit etmek için *training data* olarak kullanacağız ve diğer verileri **mean\_absolute\_error** değerini hesaplamak için *validation data* (doğrulama verileri) olarak kullanacağız.



Wow!

in-sample veriler için mean absolute error değerimiz yaklaşık 500 dolardı. out-of-sample verilerde ise 250.000 dolardan fazla.

Bu, neredeyse tamamen doğru olan bir model ile en pratik amaçlar için kullanılamayan bir model arasındaki farktır.

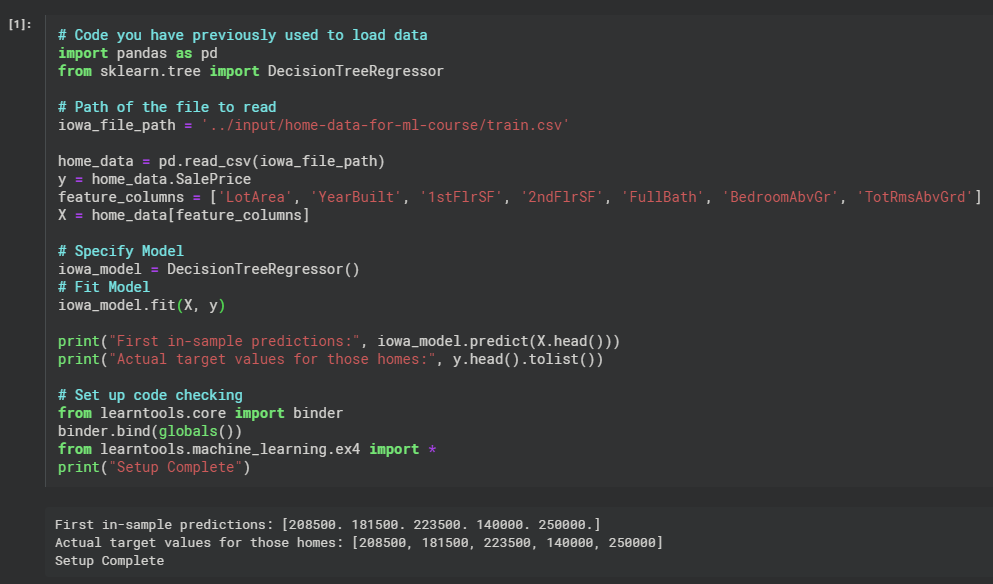
Bir referans noktası olarak, validation data’daki (doğrulama verilerindeki) ortalama ev değeri 1,1 milyon dolar.

Yani yeni verilerdeki hata ortalama ev değerinin dörtte biri kadardır.

Bu modeli geliştirmenin daha iyi feature’lar bulmak veya farklı model türleri bulmayı denemek gibi birçok yolu vardır.

Exercise: Model Validation

Bir model oluşturdunuz. Bu alıştırmada modelinizin ne kadar iyi olduğunu test edeceksiniz.

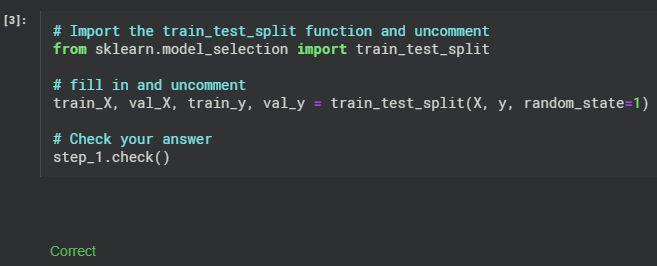


**Exercises**

**Step 1: Split Your Data (Verinizi Ayırın)**

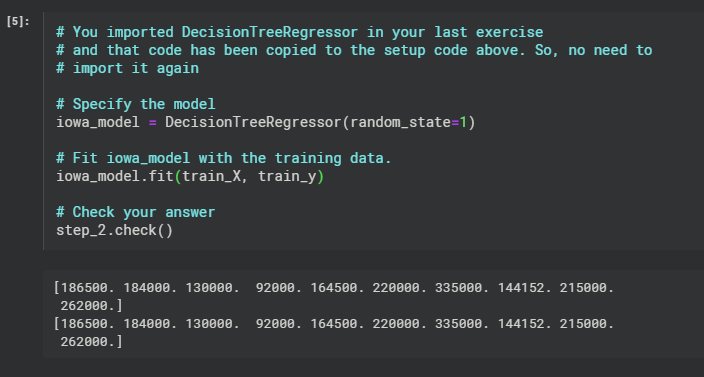
Verilerinizi bölmek için **train\_test\_split** işlevini kullanın.

Hatırlayın, feature’larınız DataFrame X'e yüklenir ve target(hedefiniz) y olarak yüklenir.

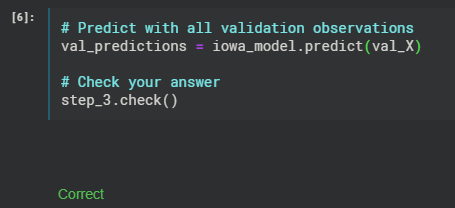


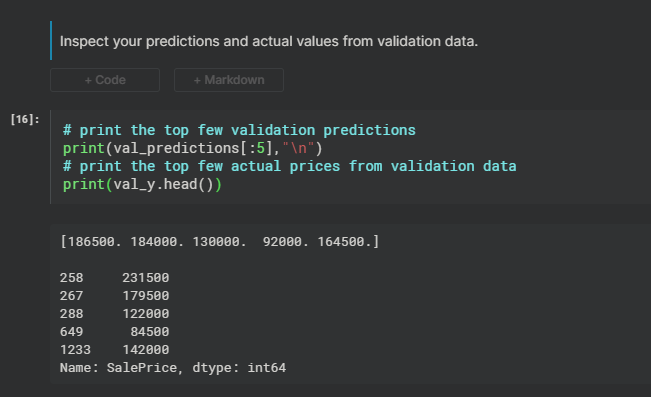
**Step 2: Specify and Fit the Model (Modeli belirleme ve fit etme)**

**DecisionTreeRegressor** modeli oluşturun ve modeli ilgili veriler ile fit edin.



**Step 3: Make Predictions with Validation Data**

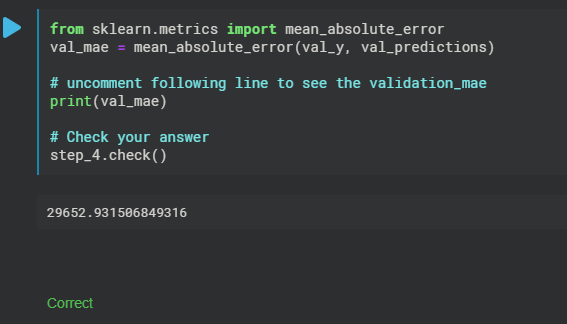




Bu gördüğünüz çıktıların in-sample tahminlerden neden farklı olduğunu anladınız mı?

Validation predictions’ların neden in-sample (veya train) predictions’larından farklı olduğunu hatırlıyor musunuz?

**Step 4: Calculate the Mean Absolute Error in Validation Data**



MAE sonucu iyi mi? Uygulamalar arasında geçerli olan değerlerin genel bir kuralı yoktur. Ancak bir sonraki adımda bu sayının nasıl kullanılacağını (ve geliştirileceğini) göreceksiniz.

**Underfitting and Overfitting**

Bu adımın sonunda, **underfitting**(uygun olmayan) ve **overfitting**(fazla uygunluk) kavramlarını anlayacak ve modellerinizi daha doğru hale getirmek için bu fikirleri uygulayabileceksiniz.

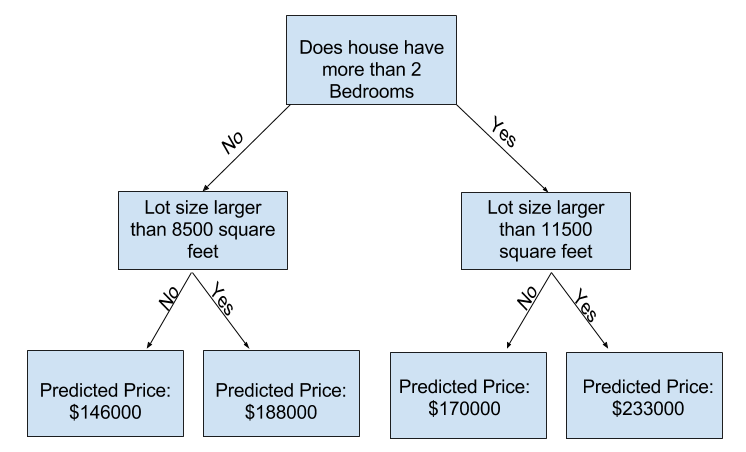
Farklı Modellerle Deneme

Artık model doğruluğunu ölçmenin güvenilir bir yoluna sahip olduğunuza göre, alternatif modelleri deneyebilir ve hangisinin en iyi tahminleri verdiğini görebilirsiniz.

Peki modeller için hangi alternatifleriniz var?

Scikit-learn'un dökümantasyonunda, Decision Tree modelinin birçok seçeneğe sahip olduğunu görebilirsiniz (isteyeceğinizden veya ihtiyacınız olandan daha fazla).

En önemli seçenekler ağacın derinliğini belirler. Bu mikro kursta ilk dersten, bir ağacın derinliğinin bir tahmine gelmeden önce kaç bölünme yaptığının bir ölçüsü olduğunu hatırlayın. Bu nispeten sığ bir ağaçtır:



Uygulamada, bir ağacın en üst seviyesi (tüm evler) ve bir leaf(yaprak) arasında 10 bölünme olması nadir değildir.

Ağaç derinleştikçe, veri kümesi daha az ev içeren yapraklara dilimlenir.

Bir ağacın sadece 1 bölünmesi varsa, verileri 2 gruba ayırır.

Her grup tekrar bölünürse, 4 grup ev alırdık. Bunların her birini tekrar bölmek 8 grup oluşturacaktır.

Her seviyede daha fazla bölme ekleyerek grup sayısını ikiye katlamaya devam edersek, 10. seviyeye ulaştığımızda 210 ev grubumuz olacak. Bu 1024 yaprak yapar.

Evleri birçok yaprak arasında böldüğümüzde, her yaprakta da daha az ev olur.

Çok az evi olan yapraklar, o evlerin gerçek değerlerine oldukça yakın tahminler yapacak, ancak yeni veriler için çok güvenilir olmayan tahminler yapabilirler (çünkü her tahmin sadece birkaç eve dayanmaktadır).

Bu, bir modelin train(eğitim) verileriyle neredeyse mükemmel şekilde eşleştiği, ancak validation(doğrulama) ve diğer yeni verilerde yetersiz olduğu, **overfitting** takma adı verilen bir fenomendir.

Flip tarafında, eğer ağacımızı çok sığ yaparsak, evleri çok farklı gruplara ayırmaz.

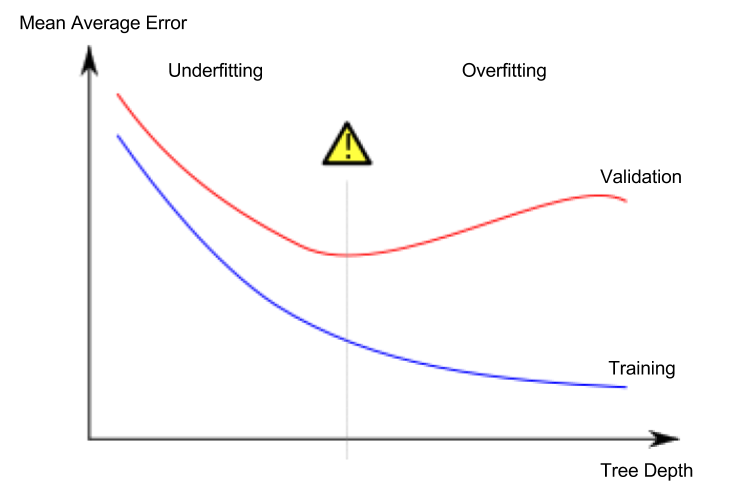
Extreme olarak, bir ağaç evleri sadece 2 veya 4'e ayırırsa, her grubun hala çok çeşitli evleri vardır.

Sonuç tahminleri(predictions), train verilerinde bile çoğu ev için çok uzak olabilir (ve aynı nedenden dolayı validation(doğrulama) da kötü olacaktır).

Bir model verilerdeki önemli ayrımları ve pattern’leri(desenleri) yakalayamadığında, train verilerinde bile yetersiz performans gösterir, buna **underfitting** denir.

Validation data’mızdan(doğrulama verimizden) predict(tahmin) ettiğimiz yeni verilerdeki accuracy’i(doğruluğu) önemsediğimiz için, **underfitting** ve **overfititng** arasındaki tatlı noktayı bulmak istiyoruz.

Görsel olarak, (kırmızı) doğrulama eğrisinin(validation curve) düşük noktasını bulmak istiyoruz.



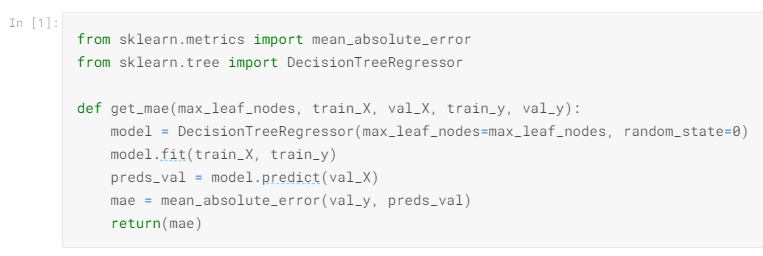
Examples

Ağaç derinliğini kontrol etmek için birkaç alternatif vardır ve birçoğu ağaçtaki bazı yolların diğer yollardan daha fazla derinliğe sahip olmasına izin verir.

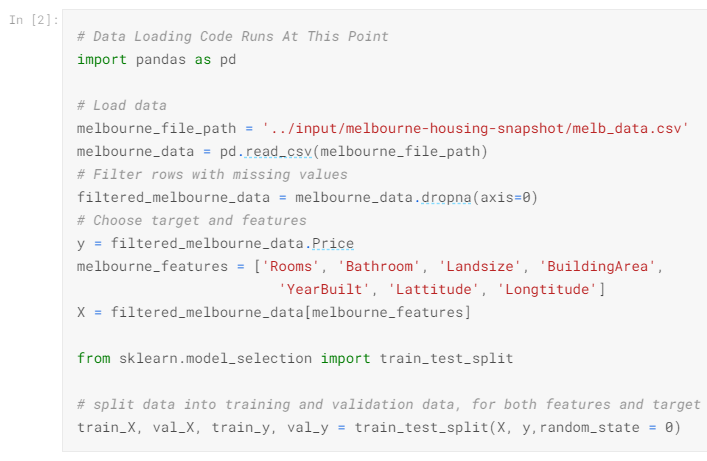
Ancak max\_leaf\_nodes argümanı, overfitting ve underfiting’i kontrol etmek için çok mantıklı bir yol sağlar.

Modelin ne kadar fazla leaf(yaprak) yapmasına izin verirsek, yukarıdaki grafikteki underfitting alanından overfitting alanına o kadar fazla hareket ederiz.

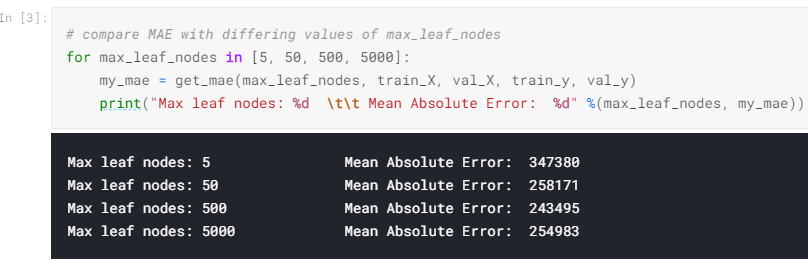
Max\_leaf\_nodes için farklı değerlerden MAE puanlarını karşılaştırmaya yardımcı olması için bir yardımcı program işlevi kullanabiliriz:



Veriler, daha önce gördüğünüz (ve daha önce yazdığınız) kodu kullanarak train\_X, val\_X, train\_y ve val\_y içine yüklenir.



Max\_leaf\_nodes için farklı değerlerle oluşturulan modellerin doğruluğunu karşılaştırmak için bir for-loop kullanabiliriz.



Listelenen seçeneklerden 500, en uygun yaprak sayısıdır.

Sonuç

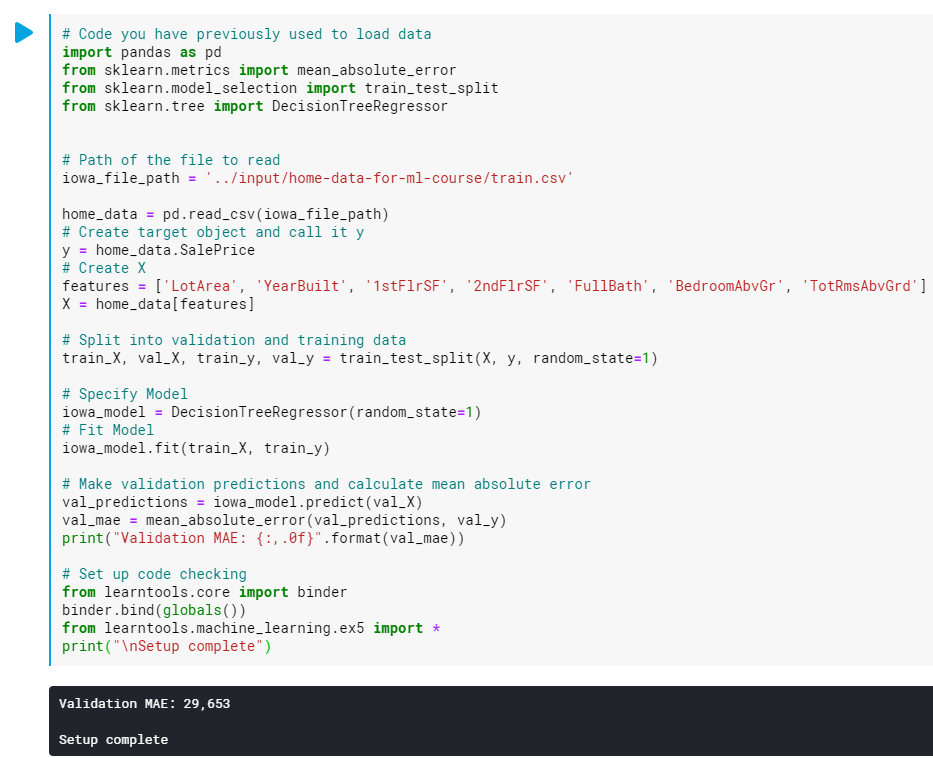
Modeller şunlardan herhangi birine sahip olabilir:

* **Overfitting:** gelecekte tekrarlamayacak sahte pattern(desen)leri yakalamak, daha az doğru tahminlere yol açmak veya
* **Underfitting:** alakalı pattern’leri yakalayamama, yine daha az doğru tahminlere yol açma.

Bir aday modelin doğruluğunu(accuracy) ölçmek için model eğitiminde(train) kullanılmayan **doğrulama(validation)** verilerini kullanıyoruz. Bu, birçok aday modeli denememizi ve en iyisini elde etmemizi sağlar.

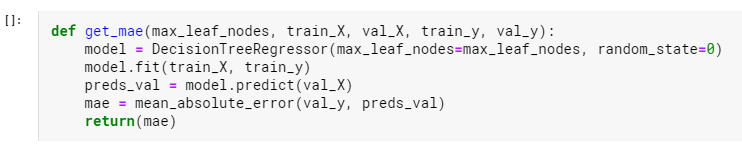
Exercise: Underfitting and Overfitting

İlk modelinizi oluşturdunuz ve şimdi daha iyi tahminler yapmak için ağacın boyutunu optimize etme zamanı. Önceki adımı bıraktığınız yerde kodlama ortamınızı ayarlamak için bu hücreyi çalıştırın.



**Exercises**

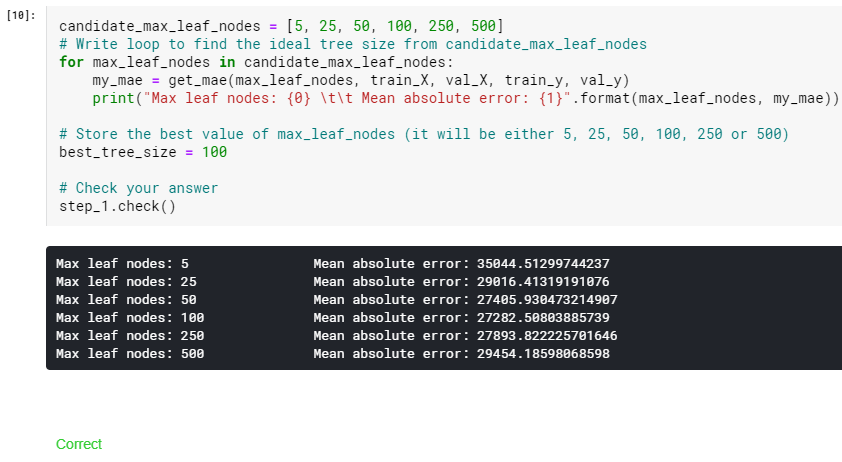
*Get\_mae* fonksiyonunu kendiniz yazabilirsiniz. Şimdilik tedarik edeceğiz. Bu, bir önceki derste okuduğunuz işlevle aynıdır. Aşağıdaki hücreyi çalıştırmanız yeterlidir.



**Step 1: Compare Different Tree Sizes (Farklı ağaç boyutlarını karşılaştırın)**

Bir dizi olası değerden ***max\_leaf\_nodes*** için aşağıdaki değerleri çalıştıran bir döngü yazın.

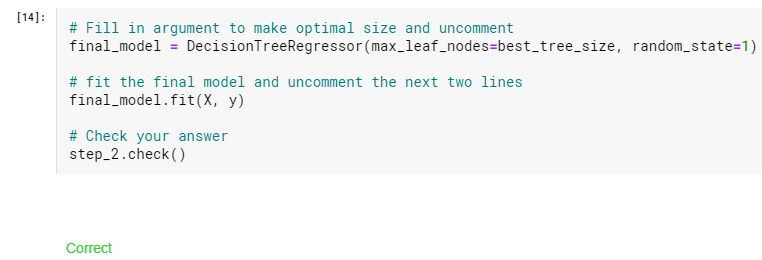
Her max\_leaf\_nodes değerinde get\_mae işlevini çağırın. Çıktıyı, verilerinizde en doğru modeli veren max\_leaf\_nodes değerini seçmenize izin verecek şekilde saklayın.



**Step 2: Fit Model Using All Data**

En iyi ağaç boyutunu biliyorsun. Bu modeli pratikte deploy edecek olsaydınız, tüm verileri kullanarak ve bu ağaç boyutunu koruyarak daha da doğru hale getirirsiniz.

Yani, tüm modelleme kararlarınızı verdiğiniz için doğrulama verilerini saklamanız gerekmez.



Bu modeli ayarladınız ve sonuçlarınızı geliştirdiniz. Ancak hala modern makine öğrenimi standartlarına göre çok karmaşık olmayan *Decision Tree* modellerini kullanıyoruz. Bir sonraki adımda, modellerinizi daha da geliştirmek için **Random Forest** kullanmayı öğreneceksiniz.

**Random Forests**

Introduction

Decision Tree sizi zor bir kararla baş başa bırakır. Çok sayıda yapraklı derin bir ağaç, her tahmin, yaprağındaki sadece birkaç evden gelen tarihsel verilerden geldiğinden fazla olacaktır. Ancak, az yapraklı sığ bir ağaç kötü performans gösterecektir, çünkü ham verilerdeki birçok farklılığı yakalayamaz.

Günümüzün en sofistike modelleme teknikleri bile, underfitting ve overfitting arasındaki bu gerilim ile karşı karşıyadır.

Ancak, birçok model daha iyi performans sağlayabilecek akıllı fikirlere sahiptir. Örnek olarak **Random Forest**’a bakacağız.

Random Forest birçok ağaç kullanır ve her bileşen ağacının tahminlerini ortalayarak bir tahmin yapar.

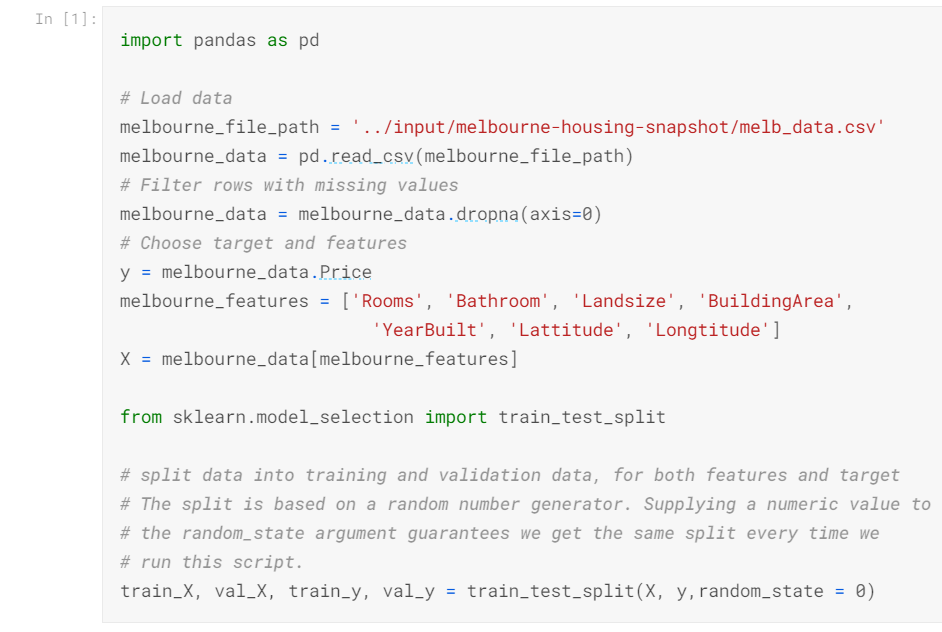
Genellikle tek bir karar ağacından çok daha iyi tahmin doğruluğu(predictive accuracy) vardır ve varsayılan parametrelerle iyi çalışır.

Modellemeye devam ederseniz, daha iyi performansa sahip daha fazla model öğrenebilirsiniz, ancak bunların çoğu doğru parametreleri almaya duyarlıdır.

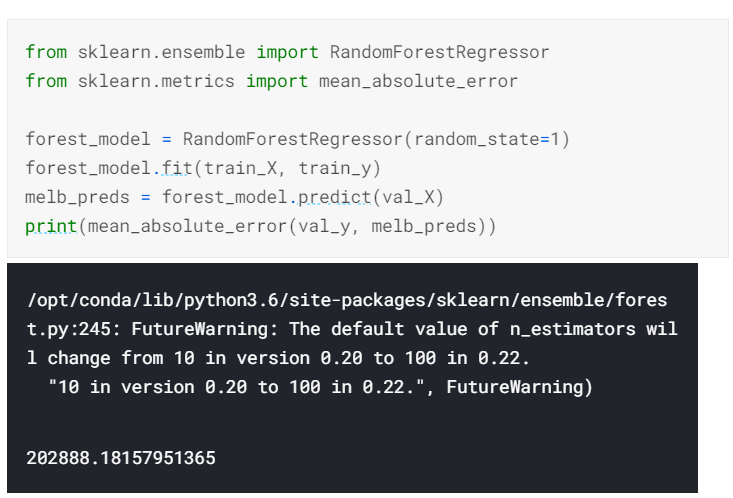
Example

Verileri yüklemek için gereken kodu zaten birkaç kez gördünüz. Veri yüklemenin sonunda aşağıdaki değişkenler bulunur:

* train\_X
* val\_X
* train\_y
* val\_y



scikit-learn kütüphanesinde decision tree modeli oluşturduğumuz gibi bu kez **random forest** modeli oluşturacağız. – **DecisionTreeRegressor** yerine **RandomTreeRegressor** kullanacağız.



Sonuç

Daha da iyileştirilmesi muhtemeldir, ancak bu 250.000 olan en iyi karar ağacı hatası üzerinde büyük bir gelişmedir.

Single decision tree’nin maksimum derinliğini değiştirdiğimiz gibi Random Forest'ın da performansını değiştirmenize izin veren parametreler var.

Ancak Random Forest modellerinin en iyi özelliklerinden biri, bu ayarlama olmadan bile genellikle makul bir şekilde çalışmasıdır.

Yakında, doğru parametrelerle iyi ayarlandığında daha iyi performans sağlayan (ancak doğru model parametrelerini elde etmek için biraz beceri gerektiren) XGBoost modelini öğreneceksiniz.

Exercises: Random Forest

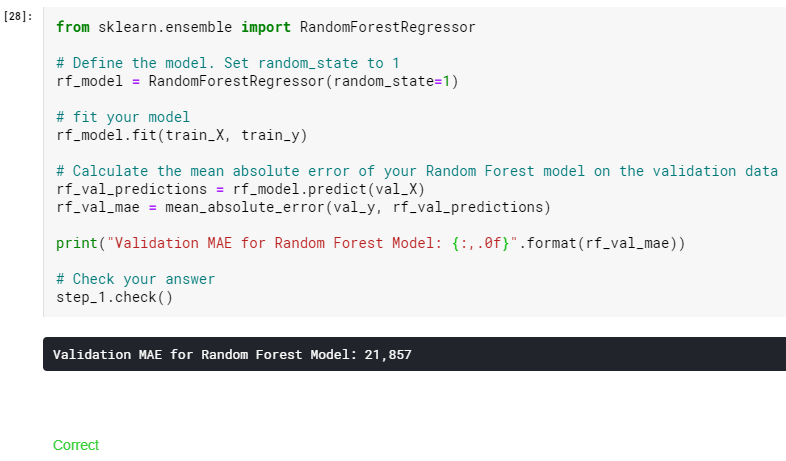
Şimdiye kadar yazdığımız kod:



**Exercises**

Veri bilimi her zaman bu kadar kolay değildir. Ancak Decision Tree’yi Random Forest ile değiştirmek kolay bir kazanç olacaktır.

**Step 1: Use a Random Forest**



Şimdiye kadar, projenizin her adımında belirli talimatları izlediniz. Bu, temel fikirleri öğrenmeye ve ilk modelinizi oluşturmaya yardımcı oldu, ancak şimdi işleri kendi başınıza denemek için yeterince bilgi sahibisiniz.

Machine Learning yarışmaları, bağımsız olarak bir machine learning projesinde gezinirken kendi fikirlerinizi denemek ve daha fazla bilgi edinmek için harika bir yoldur.

**Exercises: Machine Learning Competitions**

Introduction

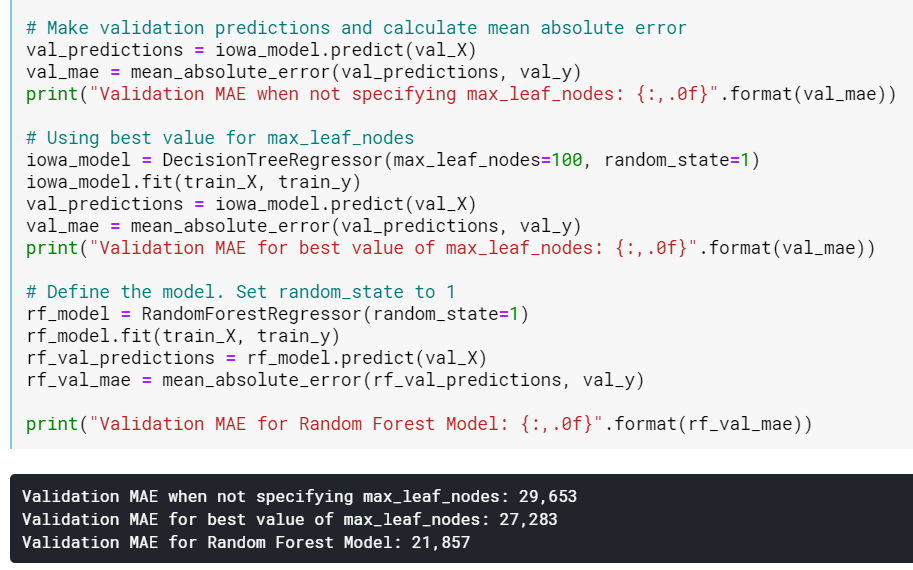
Makine öğrenimi yarışmaları, veri bilimi becerilerinizi geliştirmenin ve ilerlemenizi ölçmenin harika bir yoludur.

Bu alıştırmada, bir Kaggle yarışması için tahminler oluşturacak ve sunacaksınız.

Bu notebook’daki adımlar:

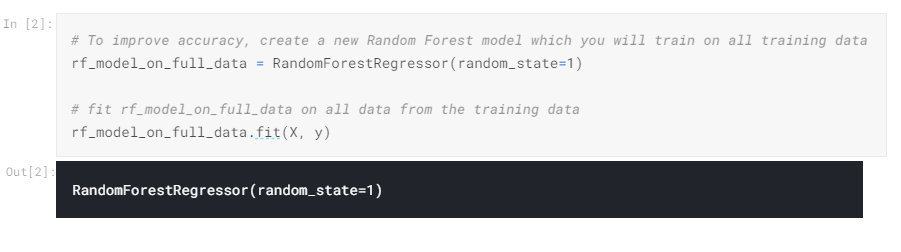
* Tüm verilerinizle Random Forest modeli oluşturun. (X ve y)
* Target(hedef) içermeyen “test” verilini okuyun. Random Forest modelinizle test verilerindeki ev fiyatlarını tahmin edin.
* Bu tahminleri yarışmaya gönderin ve puanınızı görün.
* İsteğe bağlı olarak, feature’lar ekleyerek veya modelinizi değiştirerek modelinizi geliştirip geliştiremeyeceğinizi görmek için tekrar deneyin. Daha sonra bunun rekabet lider panosunda nasıl etkilediğini görmek için yeniden gönderebilirsiniz.

Şimdiye kadar yazdığımız kod:



Creating a Model For the Competition

Random Forest modeli oluşturun ve tüm X ve y ile modeli eğitin.



Make Predictions

"Test" verileri dosyasını okuyun. Tahmin yapmak için modelinizi uygulayın.

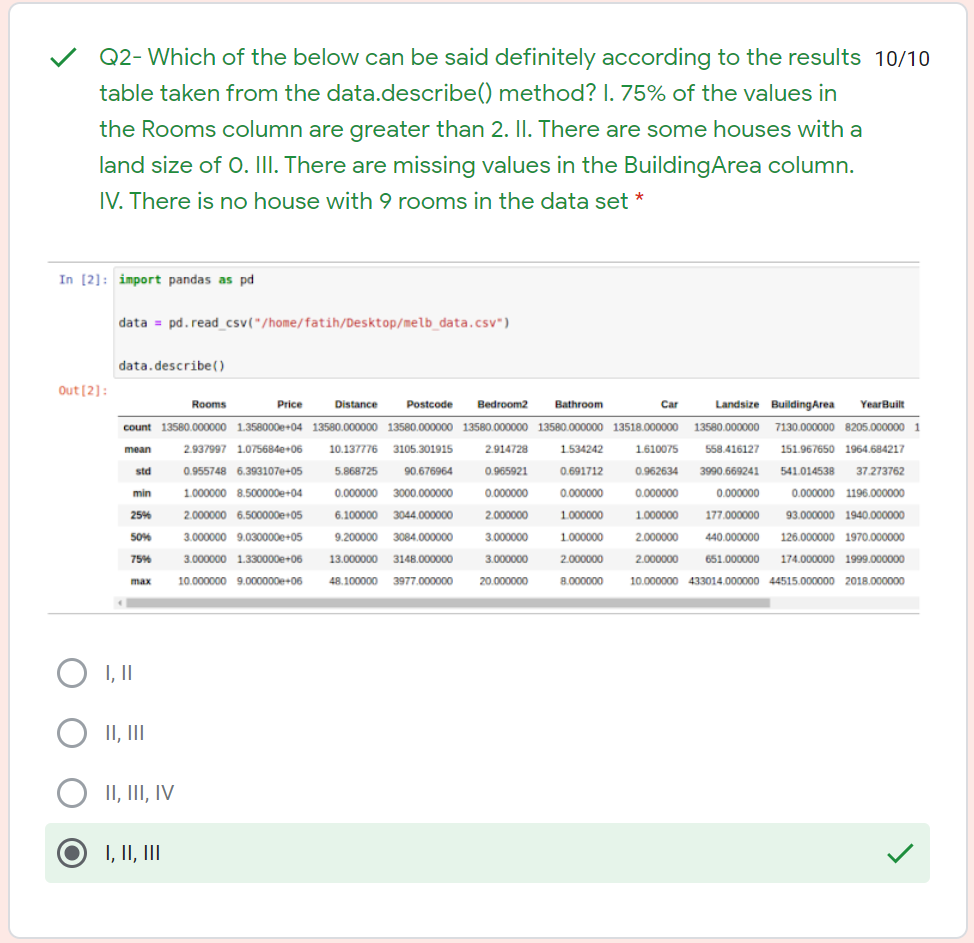
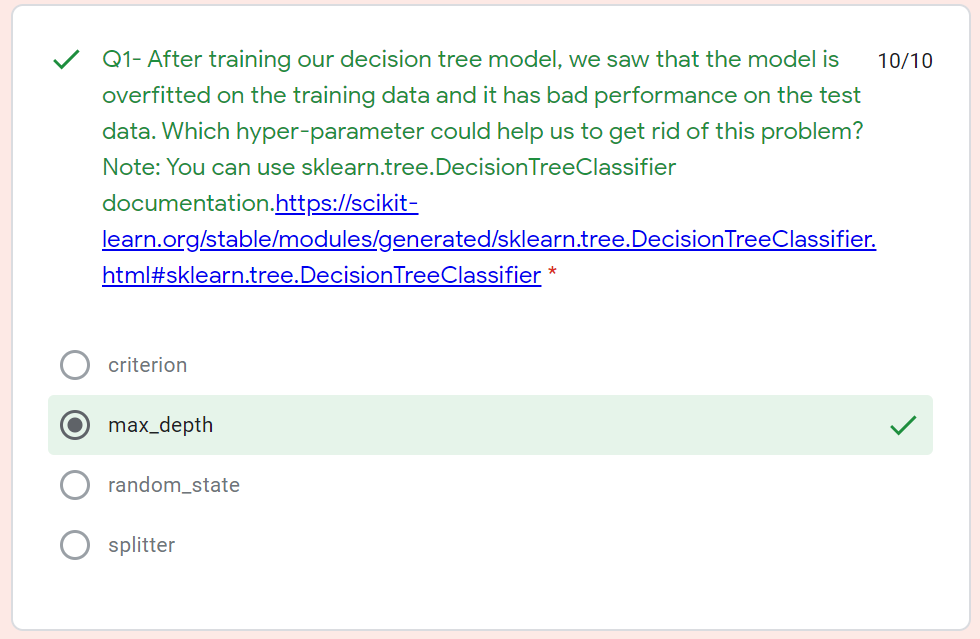


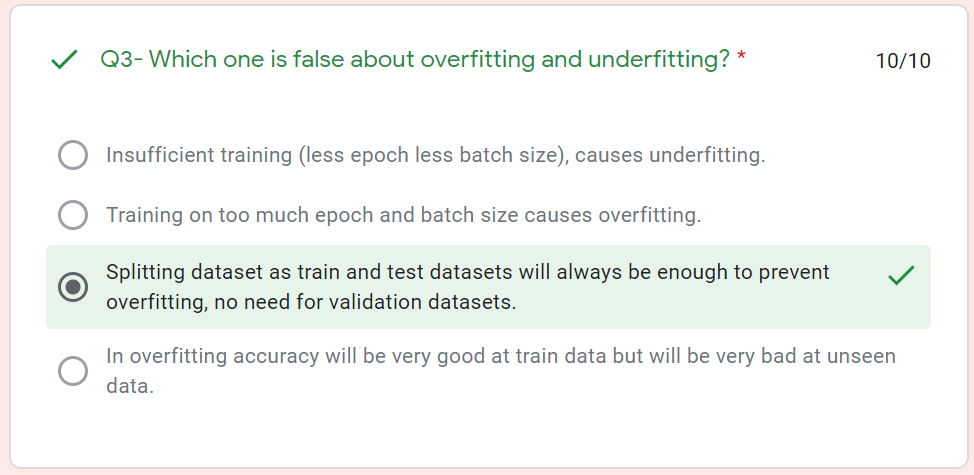
Modelinizi geliştirmenin birçok yolu vardır ve deneme yapmak bu noktada öğrenmenin harika bir yoludur.

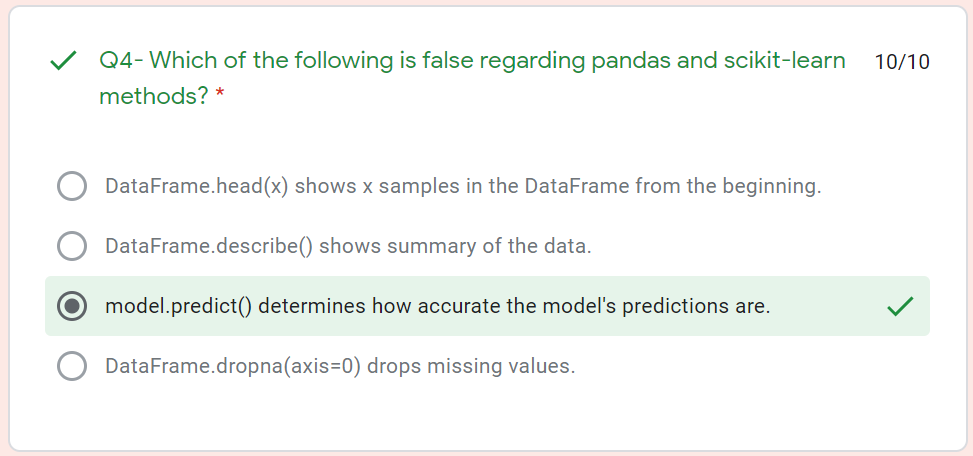
Modelinizi geliştirmenin en iyi yolu özellikler eklemektir. Sütun listesine bakın ve konut fiyatlarını nelerin etkileyebileceğini düşünün.

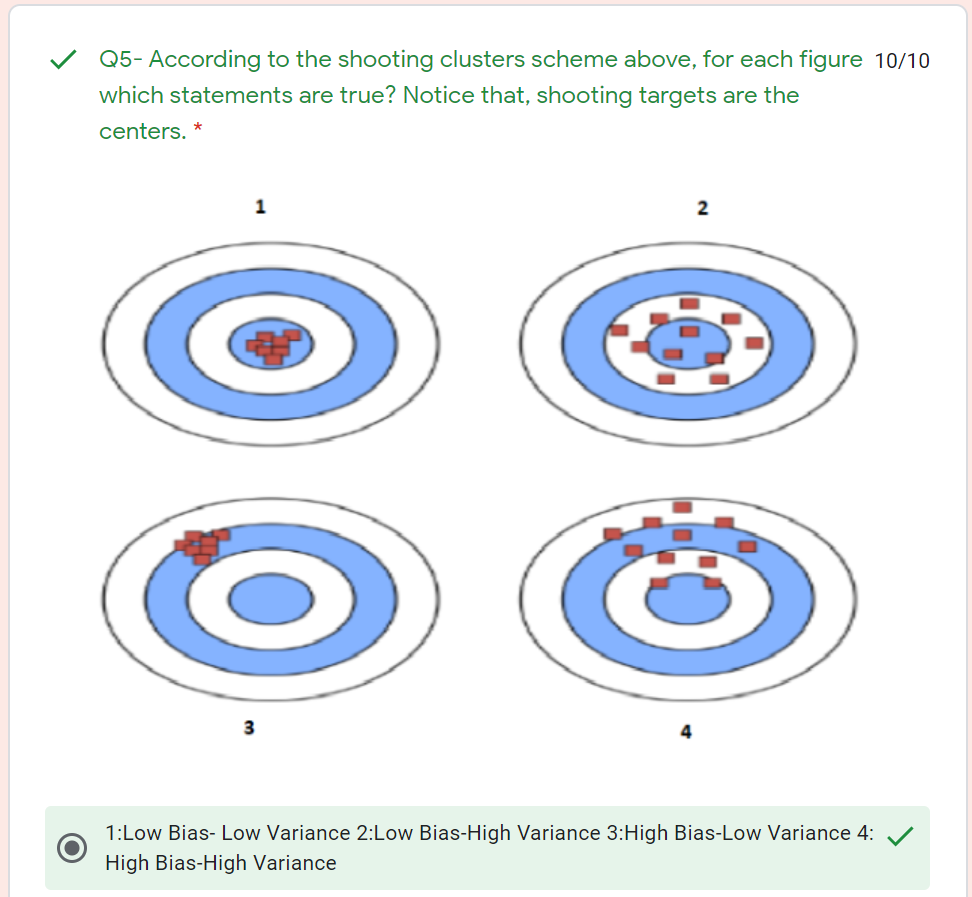
Bazı özellikler, eksik değerler veya sayısal olmayan veri türleri gibi sorunlar nedeniyle hatalara neden olur.

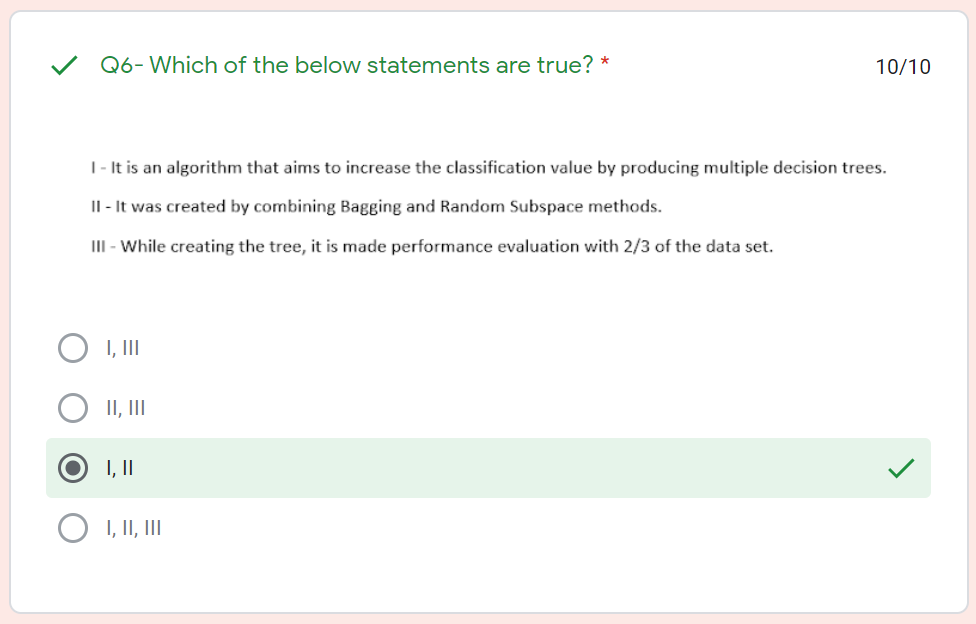
## Quiz: Intro to Machine Learning

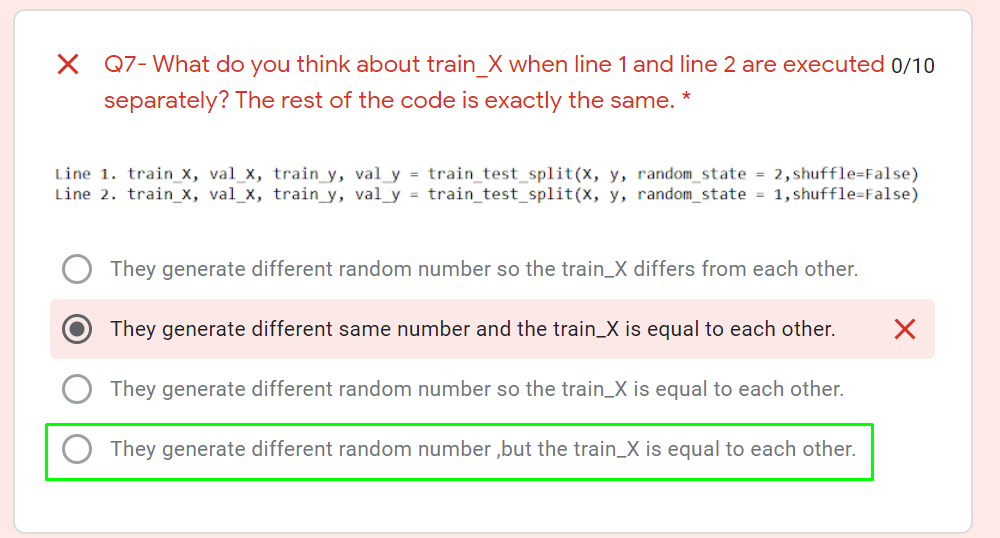


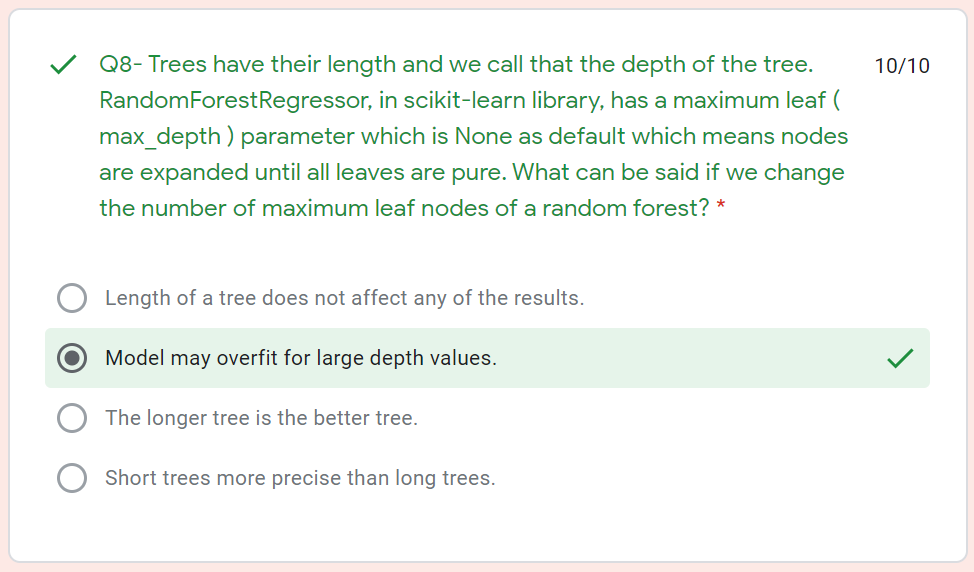


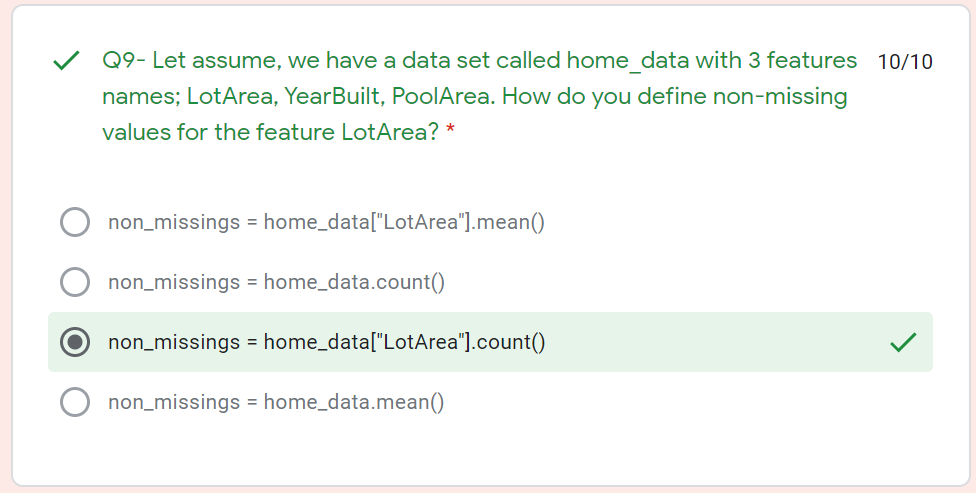


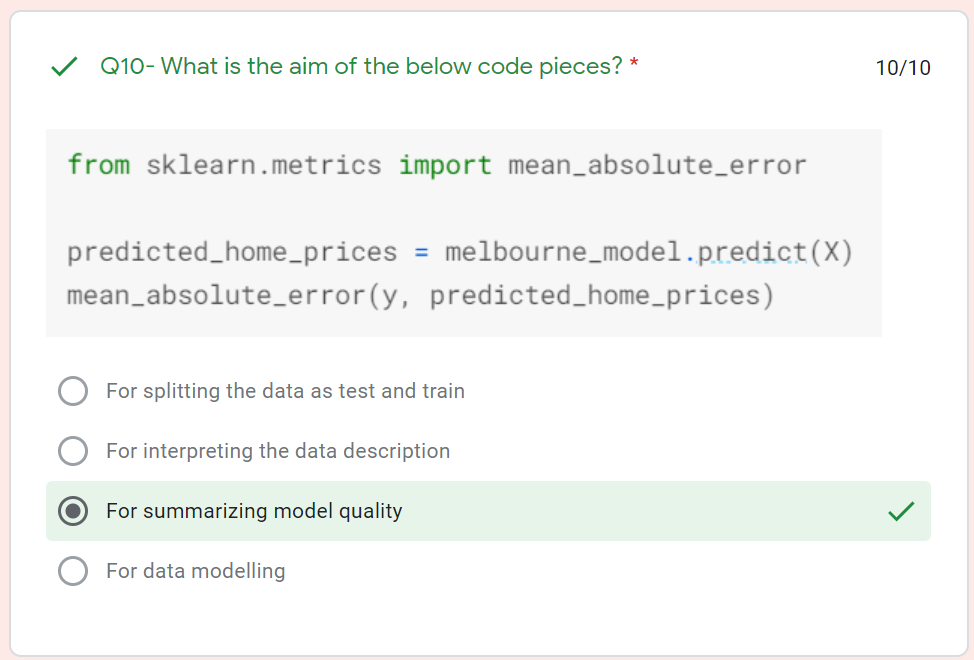












# Intermediate Machine Learning

## Introduction

Kaggle Learn'in Orta Düzey Makine Öğrenimi mikro kursuna hoş geldiniz!

Makine öğreniminde biraz geçmişiniz varsa ve modellerinizin kalitesini nasıl hızla artıracağınızı öğrenmek istiyorsanız, doğru yerdesiniz!

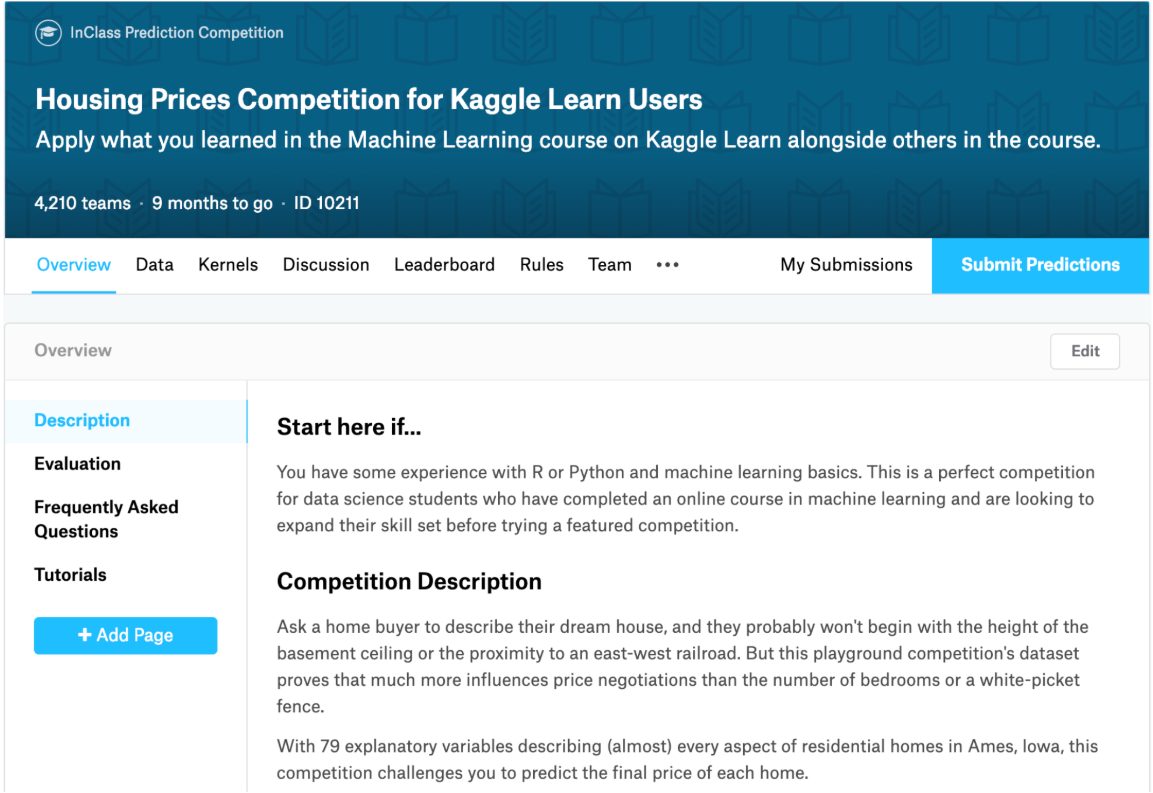
Bu mikro kursta, aşağıdakileri nasıl yapacağınızı öğrenerek makine öğrenimi uzmanlığınızı hızlandıracaksınız:

* gerçek dünya veri kümelerinde sıklıkla bulunan veri türlerini ele alır (missing values, categorical variables),
* makine öğrenme kodunuzun kalitesini artırmak için **pipeline’lar** tasarlamak,
* model doğruluğu için gelişmiş teknikler kullanabilecek (**cross validation**),
* Kaggle yarışmalarını kazanmak için yaygın olarak kullanılan son model modeller oluşturmak (**XGBoost**) ve
* yaygın ve önemli veri bilimi hatalarından (**leakege**) kaçının.

Kurs boyunca, her yeni konu için gerçek verilerle uygulamalı bir alıştırma yaparak bilginizi güçlendireceksiniz.

Uygulamalı alıştırmalar [Housing Prices Competition for Kaggle Learn Users](https://www.kaggle.com/c/home-data-for-ml-course)’dan elde edilen verileri kullanır, burada ev fiyatlarını tahmin etmek için 79 farklı açıklayıcı değişken (type of roof, number of bedrooms, and number of bathrooms gibi) kullanacaksınız.

Bu yarışmaya tahminler göndererek ve liderlik sıralamasında pozisyonunuzun yükselişini izleyerek ilerlemenizi ölçeceksiniz!

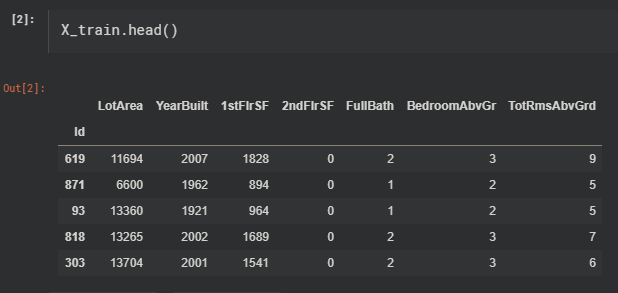


### Exercises

Bir ısınma olarak, bazı makine öğrenimi temellerini gözden geçirecek ve ilk sonuçlarınızı bir Kaggle yarışmasına sunacaksınız.

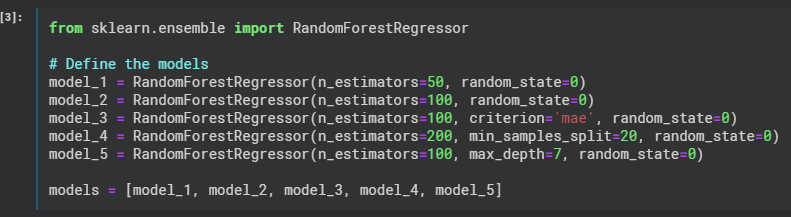
[Housing Prices Competition for Kaggle Learn Users](https://www.kaggle.com/c/home-data-for-ml-course)’dan elde edilen verilerle, evlerin her yönünü (neredeyse) tanımlayan 79 açıklayıcı değişkeni kullanarak Iowa'daki ev fiyatlarını tahmin etmek için çalışacaksınız.





### Step 1 : Eveluate Several Models (Birkaç modeli değerlendirin)

Bir sonraki kod hücresi, beş farklı Random Forest modelini tanımlar. Bu kod hücresini değişiklik yapmadan çalıştırın.



Burada kullandığımız parametrelere göz atalım;

**n\_estimators :** Random Forest içerisinde oluşturulacak ağaç sayısı. Default=10

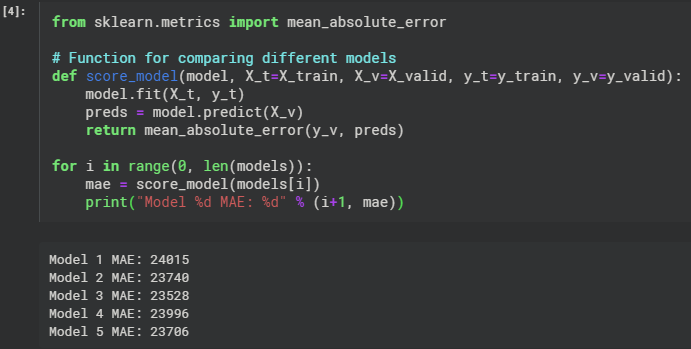
**criterion :** Bölmenin kalitesini ölçen ölçüt. Desteklenen ölçütler, ortalama kare hatası için “mse” dir; bu özellik özellik seçimi kriteri olarak varyans azaltmaya eşittir ve ortalama mutlak hata için “mae” dir.

**min\_samples\_split :** Bir bölünmenin gerçekleşmesi için verilerinizde bulunması gereken minimum örnek sayısını ayarlar. Eğer bir float ise o zaman min\_samples\_split\*n\_samples ile hesaplanır.

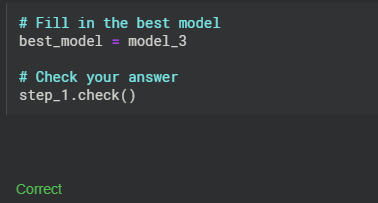
**Not:** İyi sonuçlar genellikle **max\_depth=None** ayarında **min\_samples\_split=1** ile birlikte yapılır. Bu değerleri kullanmanın belleği çok fazla işgal eden modellerle sonuçlanabileceğini unutmayın.

**max\_depth:** (integer or none) Default=None. Ağaçlarınızı ne kadar derin yapacağınızı ayarlar. max\_depth’inizi ayarlamanız, overfitting ile başa çıkabilmeniz için önerilir.

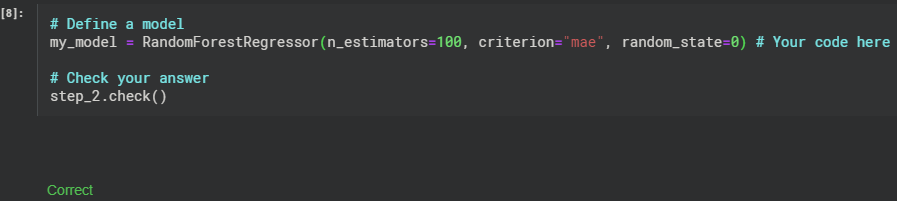
Beş model içinden en iyi modeli seçmek için, aşağıda **score\_model ()** fonksiyonunu tanımlarız. Bu işlev, doğrulama kümesinden ortalama mutlak hatayı (**MAE**) döndürür. En iyi modelin en düşük MAE'yi elde edeceğini hatırlayın.



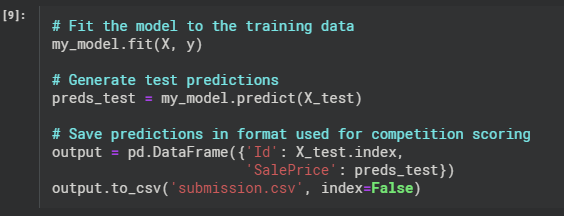
Aşağıdaki satırı doldurmak için yukarıdaki sonuçları kullanın. Hangi model en iyi modeldir? Cevabınız model\_1, model\_2, model\_3, model\_4 veya model\_5'ten biri olmalıdır.



### Step 2: Generate Test Prediction (Test tahminleri oluşturun)



Aşağıdaki kod, modeli train ve validation verilerine fit eder ve ardından bir CSV dosyasına kaydedilen test tahminleri oluşturur.



## Missing Values (Eksik Veriler)

Bu derste, eksik değerlerle başa çıkmak için üç yaklaşım öğreneceksiniz. Ardından bu yaklaşımların etkilerini gerçek dünyadaki bir veri kümesinde karşılaştıracaksınız.

Verilerin eksik değerlerle sonuçlanmasının birçok yolu vardır. Örneğin,

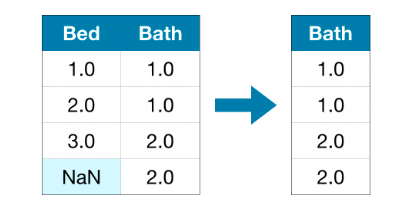
* 2 yatak odalı bir evde üçüncü bir yatak odası için bir değer bulunmayacaktır.
* Ankete katılan bir kişi gelirini paylaşmamayı tercih edebilir.

Çoğu makine öğrenme kütüphanesi (scikit-learn dahil) eksik değerlere sahip veriler kullanarak bir model oluşturmaya çalışırsanız hata verir.

### Üç Yaklaşım

#### Basit Bir Seçenek: Eksik Değerli Sütunları Düşürme

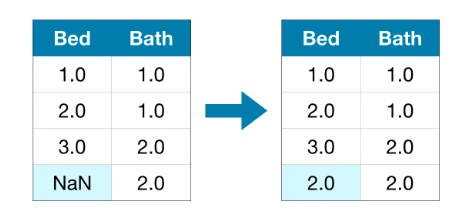
En basit seçenek, eksik değerlere sahip sütunları düşürmektir.



Düşürülen sütunlardaki değerlerin çoğu eksik değilse, model bu yaklaşımla çok sayıda bilgiye(potansiyel olarak yararlı!) erişimi kaybeder.

#### Daha İyi Bir Seçenek: Imputation

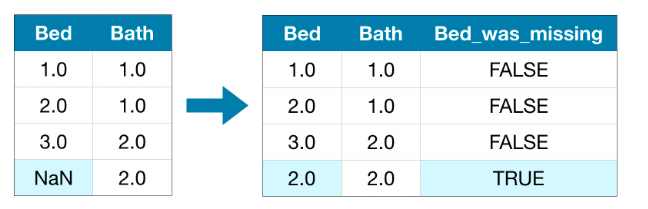
Empütasyon eksik değerleri bir sayı ile doldurur. Örneğin, her sütun boyunca ortalama değeri doldurabiliriz.



Öngörülen değer çoğu durumda tam olarak doğru olmaz, ancak genellikle sütunu tamamen bırakmanızdan daha doğru modellere yol açar.

#### An Extension To Imputation

Imputasyon standart bir yaklaşımdır ve genellikle iyi çalışır. Ancak, doldurulan değerler sistematik olarak gerçek değerlerinin (veri kümesinde toplanmayan) üstünde veya altında olabilir. Veya eksik değerleri olan satırlar başka bir şekilde benzersiz olabilir. Bu durumda, modeliniz başlangıçta hangi değerlerin eksik olduğunu göz önünde bulundurarak daha iyi tahminlerde bulunur.



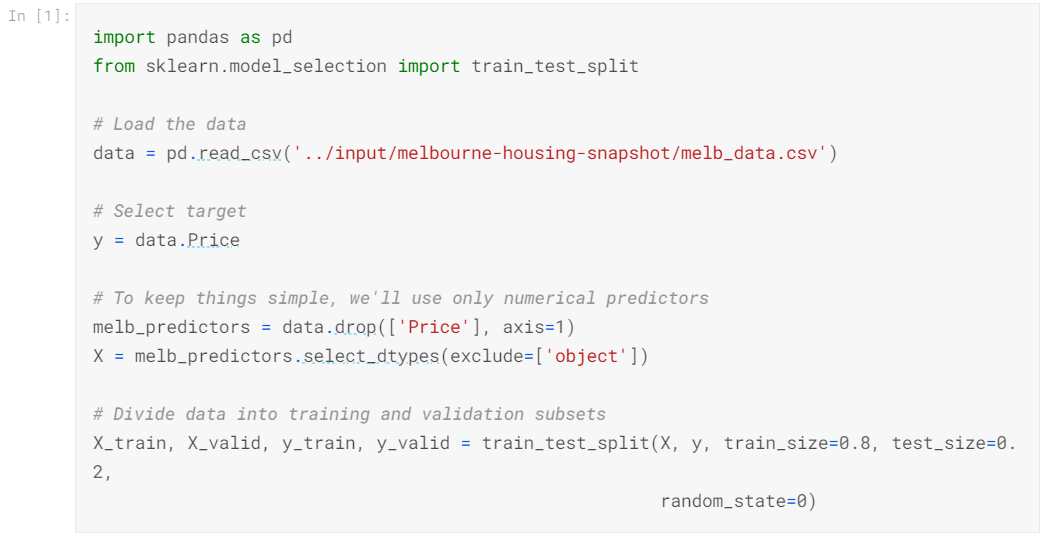
Bu yaklaşımda, eksik değerleri önceki gibi impute ediyoruz. Ayrıca, orijinal veri kümesinde eksik girişleri olan her sütun için, etkilenen girişlerin konumunu gösteren yeni bir sütun ekliyoruz.

Bazı durumlarda bu, sonuçları anlamlı şekilde iyileştirir. Diğer durumlarda, hiç yardımcı olmuyor.7

### Example

Örnekte, [Melbourne Housing dataset](https://www.kaggle.com/dansbecker/melbourne-housing-snapshot/home) ile çalışacağız. Modelimiz, ev fiyatını tahmin etmek için oda sayısı ve arazi büyüklüğü gibi bilgileri kullanacaktır.

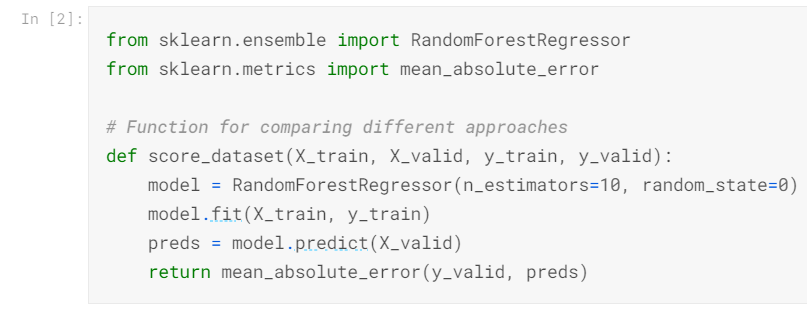
Veri yükleme adımına odaklanmayacağız. Bunun yerine, zaten X\_train, X\_valid, y\_train ve y\_valid'de train ve validation verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.



#### Define Function to Measure Quality of Each Approach (Her yaklaşımın kalitesini ölçme yaklaşımı)

Eksik değerlerle başa çıkmada farklı yaklaşımları karşılaştırmak için **score\_dataset()** işlevini tanımlarız.

Bu işlev Random Forest modelinden gelen ortalama mutlak hatayı (MAE) bildirir.



#### Score from Approach 1 (Drop Columns with Missing Values)

Hem training hem de validation setleri ile çalıştığımızdan, aynı sütunları her iki DataFrame'de de düşürmeye dikkat ediyoruz.



#### Score from Approach 2 (Imputation)

Daha sonra, eksik değerleri her sütun boyunca ortalama değerle değiştirmek için **SimpleImputer** kullanıyoruz.

Basit olmasına rağmen, ortalama değeri doldurmak genellikle oldukça iyi performans gösterir (ancak bu, veri kümesine göre değişir).

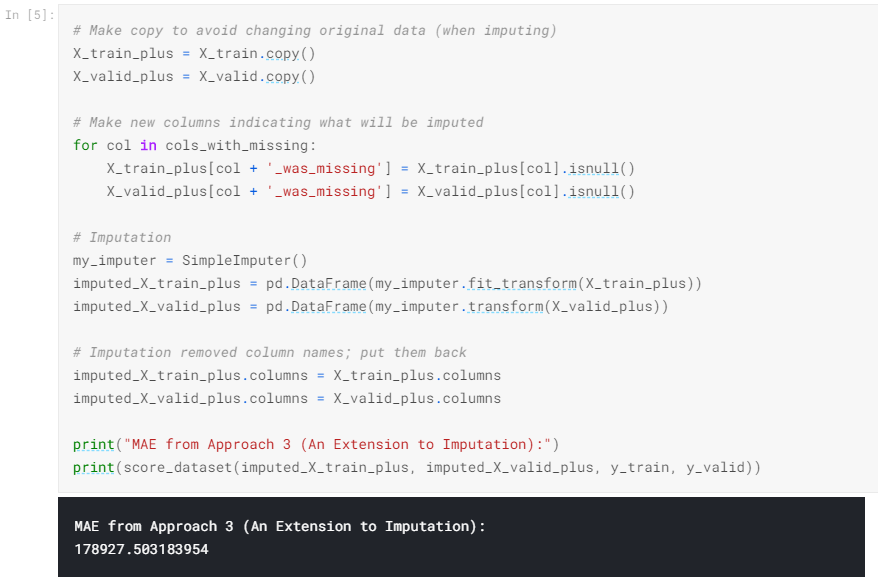
İstatistikçiler, çarpık değerleri belirlemek için daha karmaşık yollar denemiş olsa da (örneğin, **regression imputation** gibi), karmaşık stratejiler, sonuçları karmaşık makine öğrenimi modellerine bağladıktan sonra genellikle ek bir fayda sağlamaz.



Yaklaşım 2'nin, Yaklaşım 1'den daha düşük MAE'ye sahip olduğunu görüyoruz, bu nedenle Yaklaşım 2 bu veri kümesinde daha iyi performans gösterdi.

#### Score from Approach 3 (An Extension to Imputation)

Ardından, hangi değerlerin atfedildiğini takip ederken eksik değerleri de **impute**(empoze) ediyoruz.



Gördüğümüz gibi, Yaklaşım 3, Yaklaşım 2'den biraz daha kötü performans gösterdi.

**Öyleyse, neden impute edilen sütunlar drop edilenlerden daha iyi performans gösterdi?**

Training verisinde 10864 satır ve 12 sütun bulunur; burada üç sütun eksik veriler içerir. Her sütun için girişlerin yarısından azı eksik.

Bu nedenle, sütunları bırakmak çok sayıda yararlı bilgiyi kaldırır ve bu nedenle imputasyonun daha iyi performans göstermesi mantıklıdır.



### Conclusion

Genel olarak, eksik değerlerin (Yaklaşım 2 ve Yaklaşım 3'te) impute edilmesi, eksik değerlere sahip sütunları (Yaklaşım 1'de) basitçe düşürdüğümüz zamana göre daha iyi sonuçlar verdi.

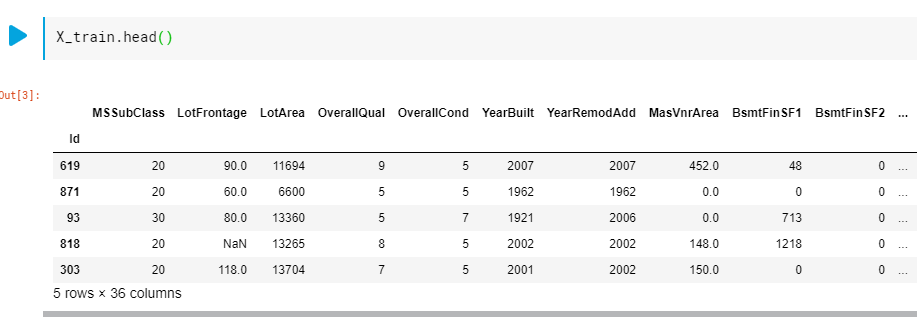
### Exercises

Şimdi, kayıp değerlerin işlenmesi hakkındaki yeni bilginizi test etme sırası sizde. Muhtemelen büyük bir fark yarattığını göreceksiniz.

Bu alıştırmada, [Housing Prices Competition for Kaggle Learn Users](https://www.kaggle.com/c/home-data-for-ml-course) verileri ile çalışacaksınız.

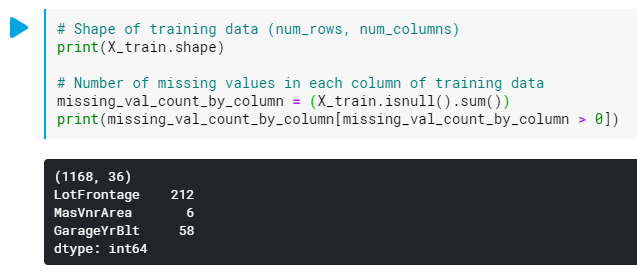




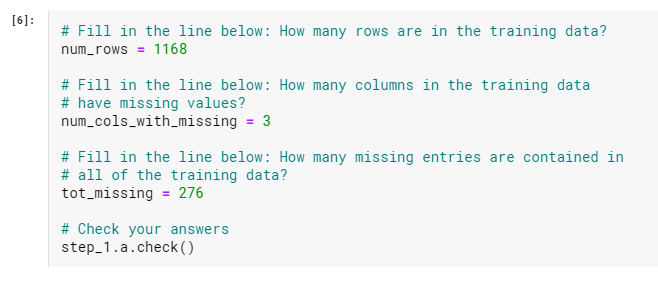


İlk birkaç satırda zaten birkaç eksik değer görebilirsiniz. Bir sonraki adımda, veri kümesindeki eksik değerleri daha kapsamlı bir şekilde anlayacaksınız.

#### Step 1: Preliminary investigation (Ön Soruşturma)



**Part A**



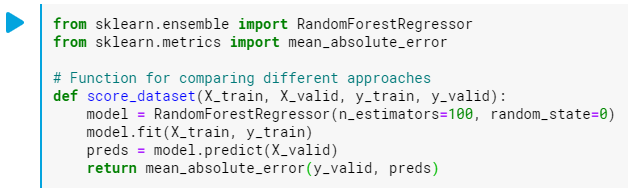
**Part B**

Yukarıdaki cevaplarınızı göz önünde bulundurarak, eksik değerlerle başa çıkmanın en iyi yaklaşımı sizce nedir?

Veri kümesinde çok fazla eksik değer var mı, yoksa sadece birkaç tane mi var? Eksik girdileri olan sütunları tamamen görmezden gelirsek çok fazla bilgi kaybeder miyiz?

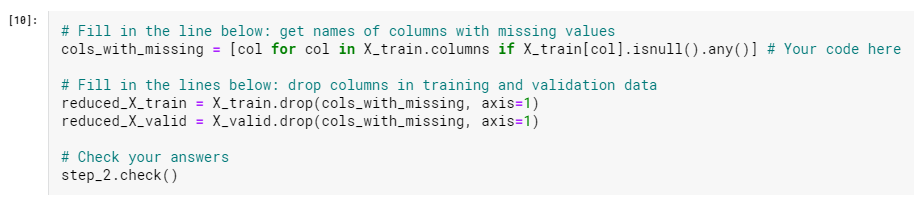
Verilerde nispeten az eksik giriş olduğundan (eksik değerlerin en büyük yüzdesine sahip sütun girişlerinin% 20'sinden daha az eksiktir), sütunları bırakmanın iyi sonuçlar vermesi beklenmez. Bunun nedeni, çok sayıda değerli veriyi atacağımızdır ve dolayısıyla imputasyon muhtemelen daha iyi performans gösterecektir.

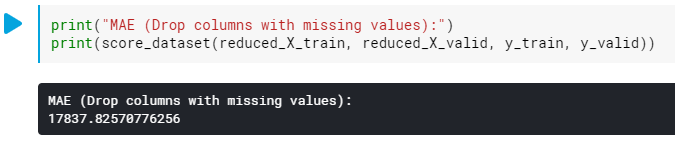
Eksik değerlerle başa çıkmak için farklı yaklaşımları karşılaştırmak için, tutorial ile aynı **score\_dataset()** işlevini kullanırsınız. Bu işlev bir Random Forest modelinden gelen ortalama mutlak hatayı (MAE) bildirir.



#### Step 2: Drop columns with missing values (Eksik değer içeren sütunları düşürün)

Bu adımda, eksik değerlere sahip sütunları kaldırmak için X\_train ve X\_valid'deki verileri önceden işlersiniz. Önceden işlenmiş DataFrames değerini sırasıyla low\_X\_train ve low\_X\_valid olarak ayarlayın.



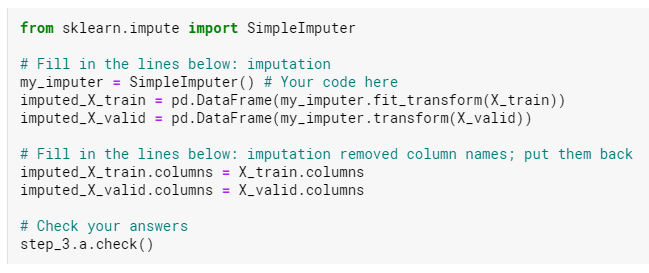


#### Step 3: Imputation

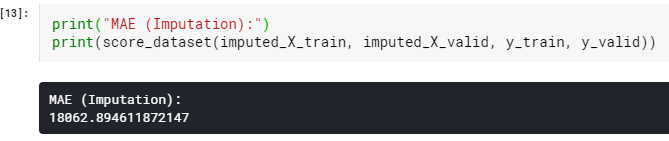
**Part A**

Her sütundaki eksik değerleri, ortalama değerler ile doldurmak için kod parçasını yazın.   
Önceden işlenmiş *DataFrames* değerini *imputed\_X\_train* ve *imputed\_X\_valid* olarak ayarlayın.

Sütun adlarının *X\_train* ve *X\_valid* ile aynı olduğundan emin olun.



Bu yaklaşım için MAE elde etmek için değişiklik olmadan sonraki kod hücresini çalıştırın.



**Part B**

Her yaklaşımdan MAE'yi karşılaştırın. Sonuçlar hakkında sizi şaşırtan bir şey var mı? Sizce neden bir yaklaşım diğerinden daha iyi performans gösteriyor?

İpucu: Kayıp değerlerin kaldırılması, impütasyondan daha büyük veya daha küçük bir MAE verdi mi? Bu, öğreticideki kodlama örneğiyle uyumlu mu?

Çözüm: Veri kümesinde çok az eksik değer olduğu düşünüldüğünde, imputasyonun sütunları tamamen düşürmekten daha iyi performans göstermesini bekleriz. Ancak bu durumda, sütunları düşürmenin biraz daha iyi performans gösterdiğini görüyoruz! Bu muhtemelen kısmen veri kümesindeki gürültüye atfedilebilirken, başka bir potansiyel açıklama, imputasyon yönteminin bu veri kümesine mükemmel bir uyumunun olmadığıdır. Yani, ortalama değer ile doldurmak yerine, her eksik değeri 0 değerine ayarlamak, en sık karşılaşılan değeri doldurmak veya başka bir yöntem kullanmak daha mantıklıdır. Örneğin, garajın inşa edildiği yılı gösteren *GarageYrBlt* sütununu düşünün. Bazı durumlarda, eksik bir değerin garajı olmayan bir evi göstermesi muhtemeldir. Bu durumda her bir sütun boyunca medyan değerini doldurmak daha anlamlı mıdır? Veya her sütun boyunca minimum değeri doldurarak daha iyi sonuçlar alabilir miyiz? Bu durumda neyin en iyisi olduğu açık değildir, ancak belki de bazı seçenekleri derhal ekarte edebiliriz - örneğin, bu sütundaki eksik değerlerin 0 olarak ayarlanması büyük olasılıkla korkunç sonuçlar verir!

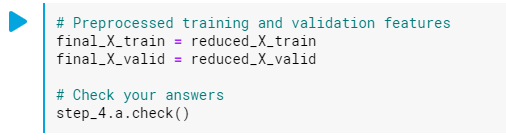
#### Step 4: Generate test predictions

Bu son adımda, eksik değerlerle başa çıkmak için seçtiğiniz herhangi bir yaklaşımı kullanacaksınız. Training ve validation özelliklerini önceden işledikten sonra, bir Random Forest modelini eğitir ve değerlendirirsiniz. Ardından, yarışmaya sunulabilecek tahminler oluşturmadan önce test verilerini önceden işlersiniz!

**Part A**

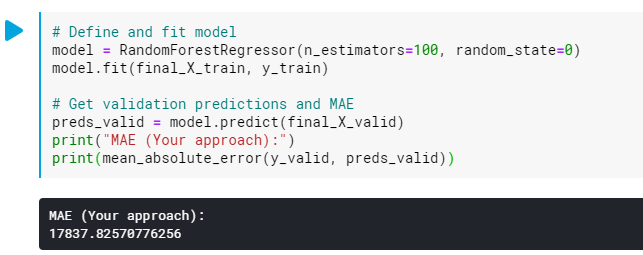
Training ve validation verilerini önceden işlemek için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames'i *final\_X\_train* ve *final\_X\_valid* olarak ayarlayın. Burada seçtiğiniz herhangi bir yaklaşımı kullanabilirsiniz! bu adımın doğru olarak işaretlenmesi için yalnızca şunlardan emin olmanız gerekir:

* önceden işlenmiş DataFrame'ler aynı sayıda sütuna sahiptir,
* önceden işlenmiş DataFrame’lerde eksik değer yoktur,
* final\_X\_train ve y\_train aynı sayıda satıra sahip olmalıdır,
* final\_X\_valid ve y\_valid aynı sayıda satıra sahip olmalıdır.



Eksik değer içeren sütunları drop işlemine tabi tuttuğumuz durumu seçtik.

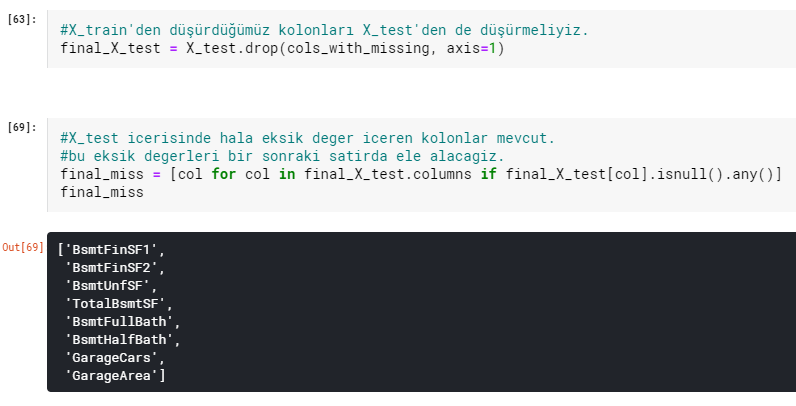
Random Forest modelini eğitmek ve değerlendirmek için bir sonraki kod hücresini çalıştırın. (Yukarıdaki score\_dataset () işlevini kullanmadığımızı unutmayın, çünkü yakında test tahminleri oluşturmak için eğitimli modeli kullanacağız!)

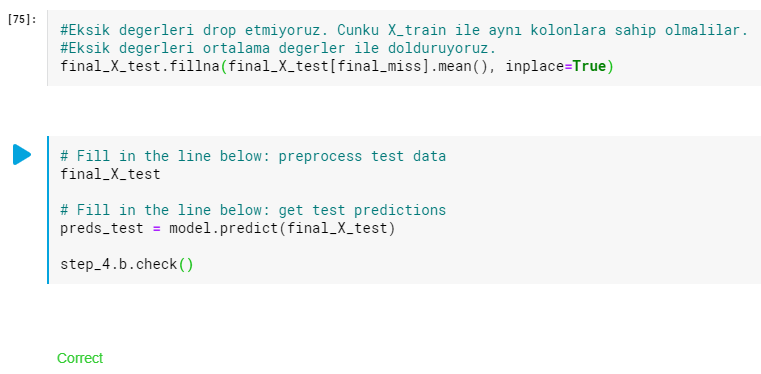


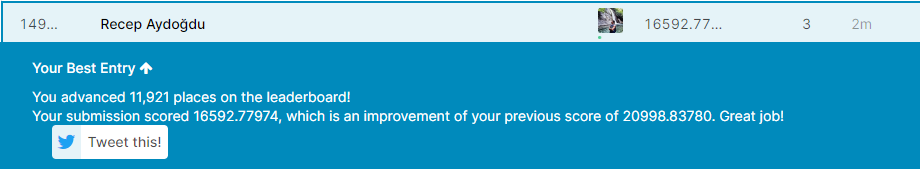
**Part B**

Test verilerinizi önceden işlemek için bir sonraki kod hücresini kullanın. Eğitim ve doğrulama verilerini nasıl önceden işleme koyduğunuzu kabul eden bir yöntem kullandığınızdan emin olun ve önceden işlenmiş test feature’larını `final\_X\_test` olarak ayarlayın.

Ardından, `preds\_test 'içinde test tahminleri oluşturmak için önceden işlenmiş test feature’larını ve eğitimli modeli kullanın.







## Categorical Variables

Bu öğreticide, bu tür verileri işlemek için üç yaklaşımla birlikte kategorik bir değişkenin ne olduğunu öğreneceksiniz.

### Introduction

Kategorik bir değişken yalnızca sınırlı sayıda değer alır.

* Ne sıklıkta kahvaltı yaptığınızı soran ve dört seçenek sunan bir anket düşünün: "Asla", "Nadiren", "Çoğu gün" veya "Her gün". Bu durumda, veriler kategoriktir, çünkü yanıtlar sabit bir kategori grubuna girer.
* İnsanlar hangi markaya sahip oldukları ile ilgili bir ankete cevap verselerdi, cevaplar "Honda", "Toyota" ve "Ford" gibi kategorilere girerdi. Bu durumda, veriler de kategoriktir.

Bu değişkenleri Python'daki çoğu makine öğrenimi modeline ilk önce ön işlem yapmadan bağlamaya çalışırsanız bir hata alırsınız.

Bu derste, kategorik verilerinizi hazırlamak için kullanabileceğiniz üç yaklaşımı karşılaştıracağız.

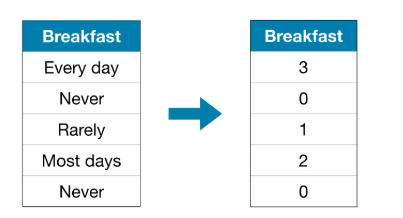
### Üç Yaklaşım

#### 1) Drop Categorical Variables

Kategorik değişkenlerle başa çıkmanın en kolay yolu, bunları veri kümesinden basitçe kaldırmaktır.   
Bu yaklaşım yalnızca sütunlar yararlı bilgiler içermiyorsa iyi sonuç verecektir.

#### 2) Label Encoding

**Label Encoding** her benzersiz değeri farklı bir tamsayıya atar.



Bu yaklaşım, kategorilerin sıralanmasını varsayar: "Asla" (0) <"Nadiren" (1) <"Çoğu gün" (2) <"Her gün" (3).

Bu varsayım bu örnekte anlamlıdır, çünkü kategorilerde tartışılmaz bir sıralama vardır.

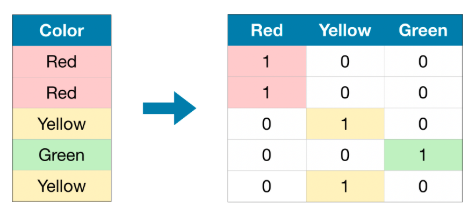
Tüm kategorik değişkenlerin değerlerde açık bir sırası yoktur, ancak **ordinal**(sıralı) değişkenler olarak adlandırılanlara atıfta bulunuruz.

Ağaç tabanlı modeller için (decision tree ve random forest gibi) label encoding’in ordinal değişkenleriyle iyi çalışmasını bekleyebilirsiniz.

#### 3) One-Hot Encoding

**One-hot encoding**, orijinal verilerdeki her olası değerin varlığını (veya yokluğunu) gösteren yeni sütunlar oluşturur.

Bunu anlamak için bir örnek üzerinde çalışacağız.



Orijinal veri kümesinde "Renk", üç kategoriden oluşan kategorik bir değişkendir: "Kırmızı", "Sarı" ve "Yeşil".   
Karşılık gelen one-hot encoding, olası her değer için bir sütun ve orijinal veri kümesindeki her satır için bir satır içerir.  
Orijinal değer "Kırmızı" olduğunda, "Kırmızı" sütununa 1 koyarız; orijinal değer "Sarı" ise, "Sarı" sütununa 1 koyarız vb.

Label encoding’in aksine, one-hot encoding kategorilerin sıralanmasını kabul etmez.  
Dolayısıyla, kategorik verilerde net bir düzen yoksa (örneğin, "Kırmızı" ne "Sarı" dan daha az veya daha az ise) bu yaklaşımın özellikle iyi çalışmasını bekleyebilirsiniz.  
İçsel sıralaması olmayan kategorik değişkenleri **nominal değişkenler** olarak adlandırırız.

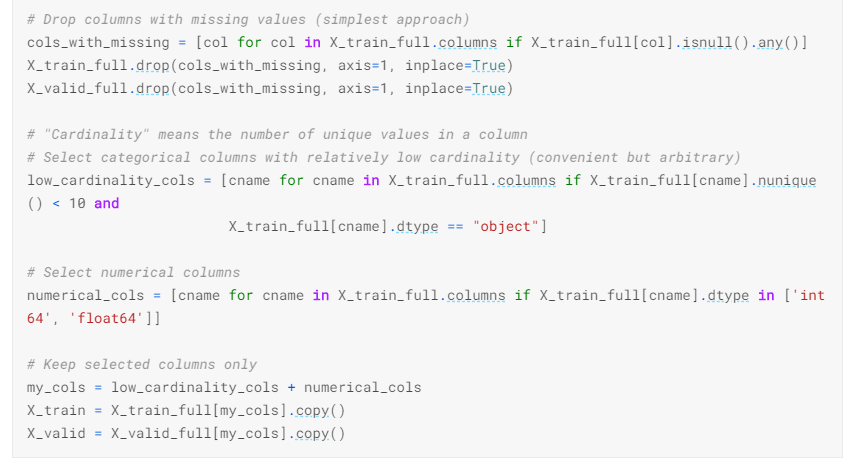
One-hot encoding, kategorik değişken çok sayıda değer alıyorsa genellikle iyi performans göstermez (yani, genellikle 15'ten fazla farklı değer alan değişkenler için kullanmazsınız).

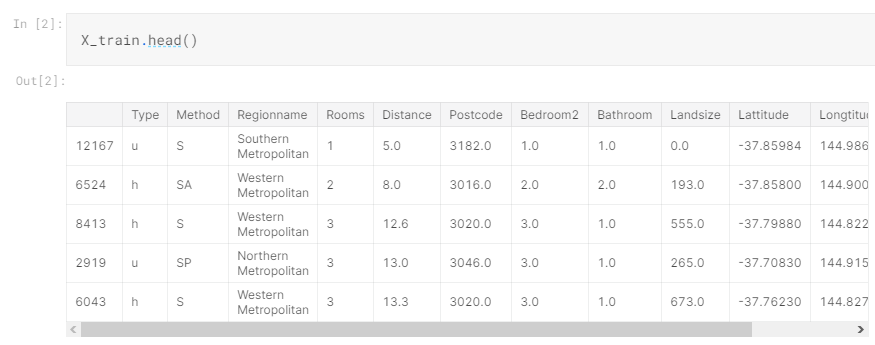
### Example

Önceki derste olduğu gibi [Melbourne Housing dataset](https://www.kaggle.com/dansbecker/melbourne-housing-snapshot/home) üzerinde çalışacağız.

Veri yükleme adımına odaklanmayacağız. Bunun yerine, zaten X\_train, X\_valid, y\_train ve y\_valid'de eğitim ve doğrulama verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.



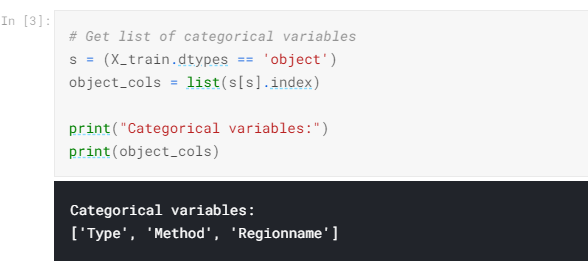






Ardından, training verilerindeki tüm kategorik değişkenlerin bir listesini elde ederiz.

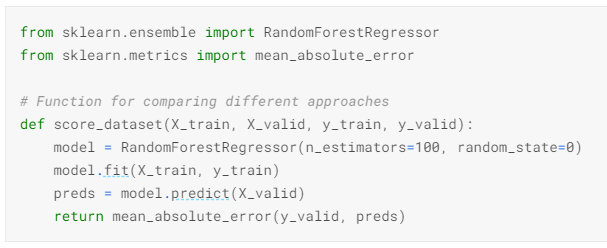
Bunu, her sütunun veri türünü (veya **dtype**) kontrol ederek yaparız. Dtype **object** bir sütunun metne sahip olduğunu gösterir (teorik olarak olabilecek başka şeyler de vardır, ancak bu bizim amaçlarımız için önemsizdir). Bu veri kümesi için, metin içeren sütunlar kategorik değişkenleri gösterir.



#### Define Function to Measure Quality of Each Approach

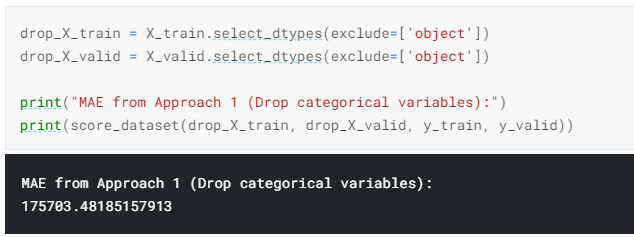
Kategorik değişkenlerle başa çıkmak için üç farklı yaklaşımı karşılaştırmak için score\_dataset () fonksiyonunu tanımlarız.

Bu işlev bir Random Forest modelinden gelen ortalama mutlak hatayı (MAE) döndürür. Genel olarak MAE'nin mümkün olduğunca düşük olmasını istiyoruz!



#### Score from Approach 1 (Drop Categorical Variables)

*Object* sütunlarını select\_dtypes () yöntemiyle düşürürüz.



#### Score from Approach 2 (Label Encoding)

Scikit-learn, etiket kodlamaları almak için kullanılabilecek bir *LabelEncoder* sınıfına sahiptir.

Kategorik değişkenler üzerinde döngü yapar ve Label Encoding’i her sütuna ayrı ayrı uygularız.



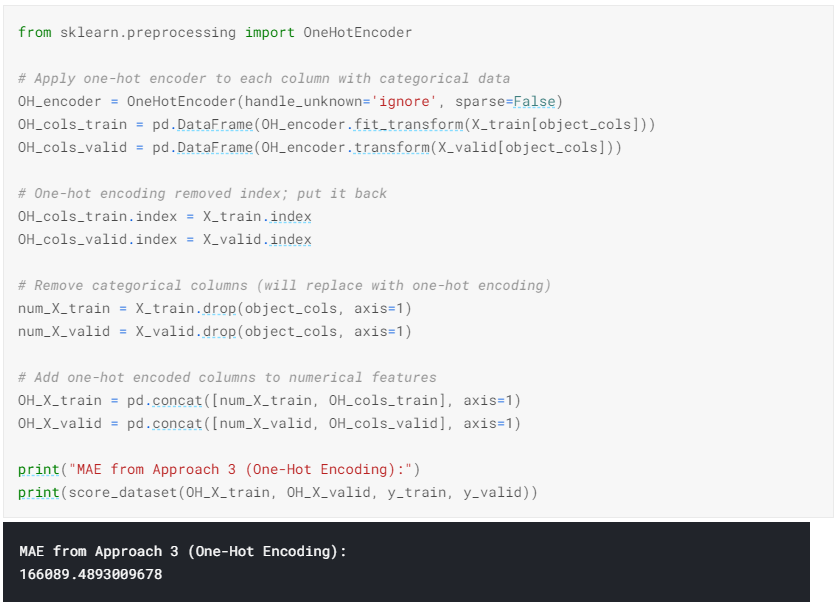
Yukarıdaki kod hücresinde, her sütun için, her benzersiz değeri rastgele farklı bir tamsayıya atarız.  
Bu, özel etiketler sağlamaktan daha basit olan yaygın bir yaklaşımdır; ancak, tüm sıralı değişkenler için daha iyi bilgilendirilmiş etiketler sağlarsak, performansta ek bir artış bekleyebiliriz.

#### Score from Approach 3 (One-Hot Encoding)

Scikit-learn'un OneHotEncoder sınıfını, one-hot encoding yapmak için kullanıyoruz. Davranışını özelleştirmek için kullanılabilecek bir dizi parametre vardır.

* Validation verileri, training verilerinde gösterilmeyen sınıflar içerdiğinde hataları önlemek için *handle\_unknown = 'ignore'* ayarını yaparız ve
* *sparse = False*, kodlanmış sütunların sayısal bir dizi olarak döndürülmesini sağlar (seyrek bir matris yerine).

Encoder’ı kullanmak için yalnızca one-hot encoded olmasını istediğimiz kategorik sütunları sağlıyoruz.  
Örneğin, training verilerini encode için **X\_train[object\_cols]** 'u sağlıyoruz.  
(aşağıdaki kod hücresindeki *object\_cols*, kategorik verileri olan sütun adlarının bir listesidir ve bu nedenle **X\_train[object\_cols]**, eğitim kümesindeki tüm kategorik verileri içerir.)



### En iyi yaklaşım hangisi?

Bu durumda, kategorik sütunları bırakmak (Yaklaşım 1) en kötü performansı gösterdi, çünkü en yüksek MAE puanına sahipti.

Diğer iki yaklaşıma gelince, geri dönen MAE puanları çok yakın olduğundan, birinin diğerine karşı anlamlı bir faydası görünmemektedir.

Genel olarak, **one-hot encoding** (Yaklaşım 3) tipik olarak en iyi performansı gösterir ve kategorik sütunları düşürmek (Yaklaşım 1) genellikle en kötü performansı gösterir, ancak duruma göre değişir.

### Sonuç

Dünya kategorik verilerle doludur. Bu ortak veri türünü nasıl kullanacağınızı biliyorsanız çok daha etkili bir veri bilimcisi olacaksınız!

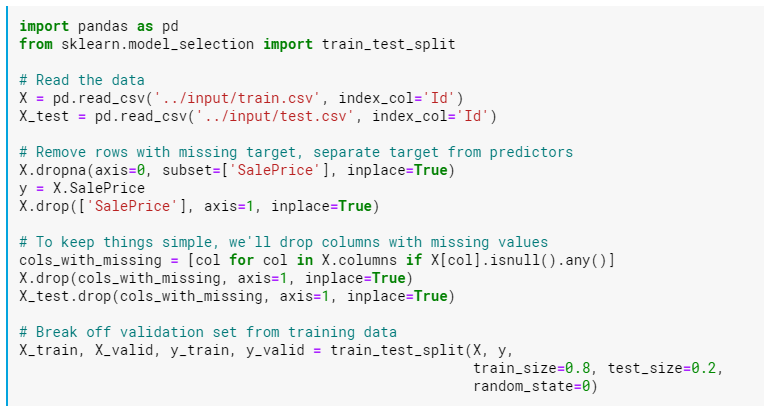
### Exercises

Kategorik değişkenleri encode ederek şimdiye kadarki en iyi sonucu elde edeceksiniz!

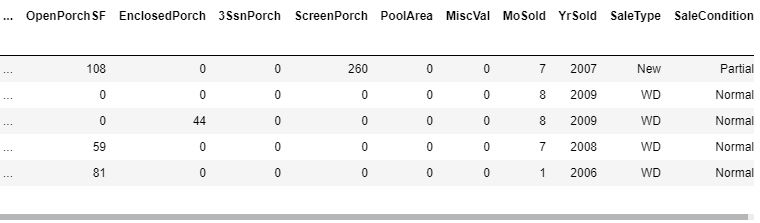
Bu alıştırmada [Housing Prices Competition for Kaggle Learn Users](https://www.kaggle.com/c/home-data-for-ml-course) ile çalışacağız.



X\_train, X\_valid, y\_train ve y\_valid'e training ve validation setlerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X\_test'e yüklenir.

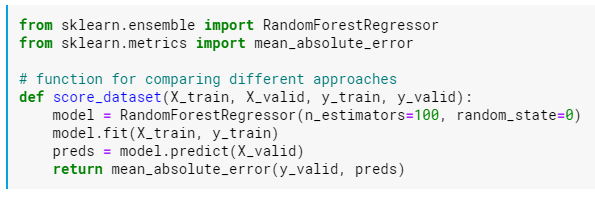






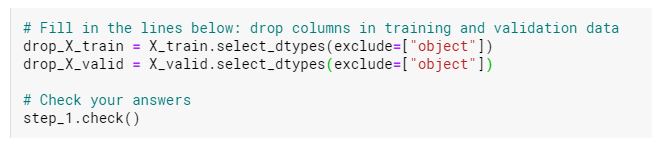
Veri kümesinin hem sayısal hem de kategorik değişkenler içerdiğine dikkat edin. Bir modeli eğitmeden önce kategorik verileri encode işlemine tabi tutmanız gerekir.

Farklı modelleri karşılaştırmak için tutorila’daki ile aynı score\_dataset () işlevini kullanırsınız. Bu işlev bir random forest modelinden gelen ortalama mutlak hatayı (MAE) bildirir.

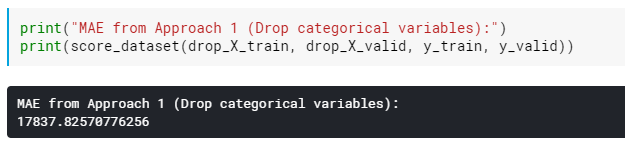


#### Step 1: Drop columns with categorical data

En basit yaklaşımla başlayacaksınız. Kategorik veriler içeren sütunları kaldırmak için X\_train ve X\_valid'deki verileri önceden işlemek için aşağıdaki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla drop\_X\_train ve drop\_X\_valid olarak ayarlayın.

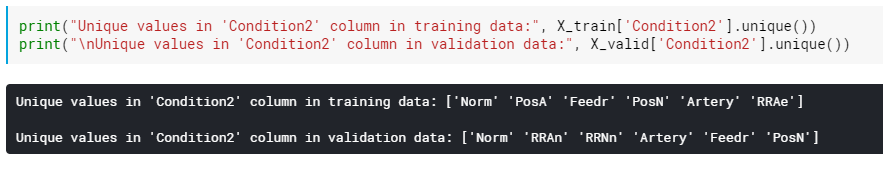


Bu yaklaşım için MAE hesaplayalım.



#### Step 2: Label Encoding

Label Encoding’e geçmeden önce veri kümesini araştıracağız. Özellikle, “Contition2” sütununa bakacağız. Aşağıdaki kod hücresi, hem eğitim hem de doğrulama kümelerindeki benzersiz girişleri yazdırır.



Şimdi buna göre kod yazarsanız:

* label encoder’i training data’ya fit ederseniz, ve sonra
* hem training hem validation verilerini transform yaparsanız,

bir hata alırsınız. Durumun neden böyle olduğunu görebiliyor musunuz? (\_Bu soruyu cevaplamak için yukarıdaki çıktıyı kullanmanız gerekir.\_)

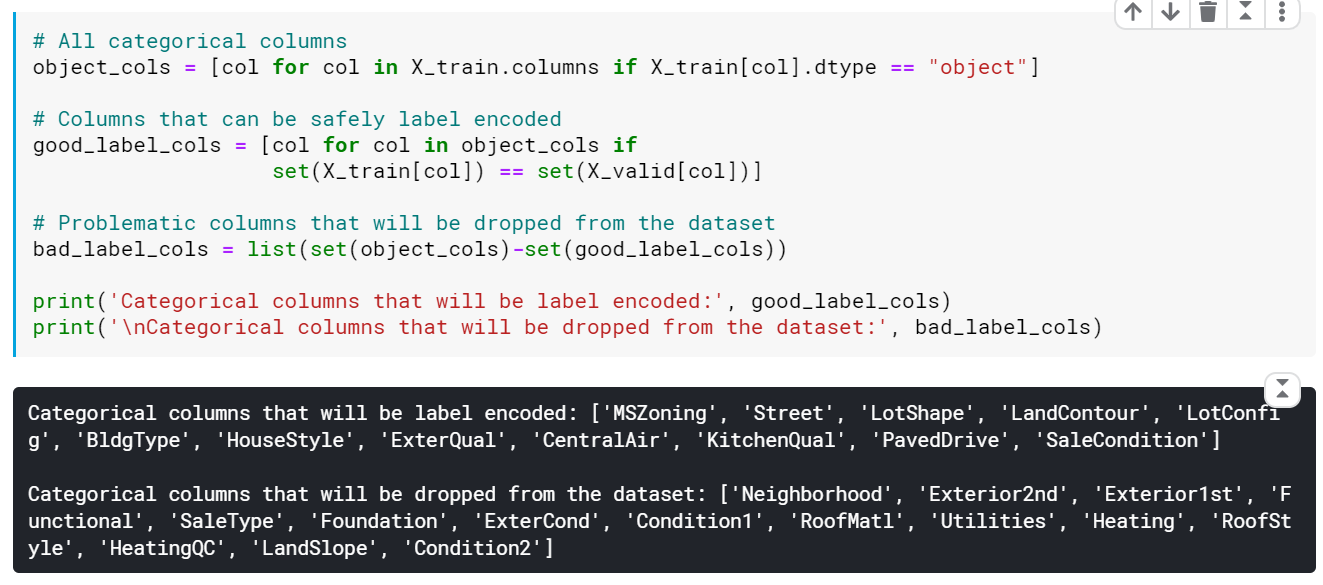
Validation verilerinde görünen ancak training verilerinde olmayan değerler var mı?

Çözüm: Training verilerindeki bir sütuna label encoding uygulanması, training verilerinde görünen her bir benzersiz değer için karşılık gelen tamsayı değerli bir etiket oluşturur. Validation verilerinin training verilerinde de görünmeyen değerler içermesi durumunda, kodlayıcı bir hata atar, çünkü bu değerlere kendilerine atanan bir tamsayı olmaz.

Doğrulama verilerindeki “Condition2” sütununun 'RRAn' ve 'RRNn' değerlerini içerdiğine dikkat edin, ancak bunlar eğitim verilerinde görünmez - bu nedenle, scikit-learn ile bir etiket kodlayıcı kullanmaya çalışırsak, kodu hata verir.

Bu gerçek dünyadaki verilerle karşılaşacağınız yaygın bir sorundur ve bu sorunu düzeltmek için birçok yaklaşım vardır. Örneğin, yeni kategorilerle ilgilenmek için özel bir Label Encoder yazabilirsiniz.  
Ancak en basit yaklaşım, sorunlu kategorik sütunları düşürmektir.

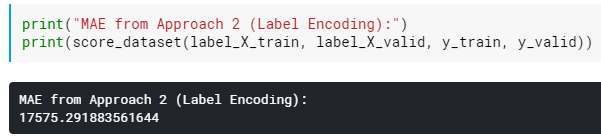
Sorunlu sütunları *bad\_label\_cols* Python listesine kaydetmek için aşağıdaki kod hücresini çalıştırın. Benzer şekilde, güvenli bir şekilde etiketlenebilen sütunlar *good\_label\_cols* içinde saklanır.



X\_train ve X\_valid içindeki verilere label encode yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla label\_X\_train ve label\_X\_valid olarak ayarlayın.

* Kategorik sütunları veri kümesinden bad\_label\_cols içine çekmek için aşağıdaki kodu sağladık.
* Kategorik sütunlar içinden good\_label\_cols’lara label encode uygulamanız gerekir.

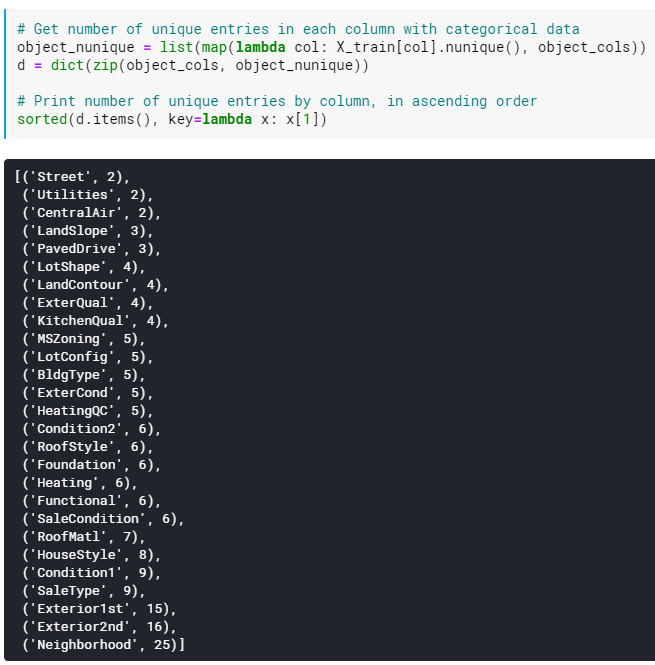




#### Step 3: Investigating Cardinality (Kardinalite Araştırması)

Şimdiye kadar, kategorik değişkenlerle başa çıkmak için iki farklı yaklaşım denediniz. Ve kategorik verileri kodlamanın, sütunları veri kümesinden kaldırmaktan daha iyi sonuçlar verdiğini gördünüz.

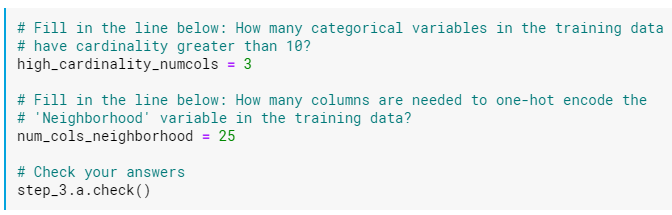
Yakında, one-hot encoding deneyeceksiniz. O zamandan önce, ele almamız gereken bir konu daha var. Bir sonraki kod hücresini değişiklik olmadan çalıştırarak başlayın.



Yukarıdaki çıktı, kategorik verilere sahip her sütun için sütundaki benzersiz değerlerin sayısını gösterir. Örneğin, training verilerindeki Street sütununun iki benzersiz değeri vardır: sırasıyla bir çakıl yol ve asfalt bir yola karşılık gelen 'Grvl' ve 'Pave'.

Kategorik bir değişkenin benzersiz girişlerinin sayısını, o kategorik değişkenin temel niteliği olarak ifade ederiz. Örneğin, 'Street' değişkeni 2 kardinaliteye sahiptir.

Aşağıdaki soruları cevaplamak için yukarıdaki çıktıyı kullanın.

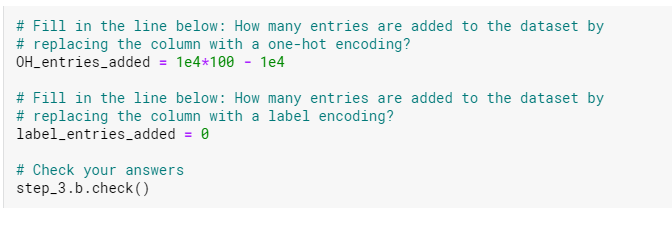


Birçok satıra sahip büyük veri kümeleri için, one-hot encoding, veri kümesinin boyutunu büyük ölçüde genişletebilir. Bu nedenle, yalnızca tipik olarak nispeten düşük kardinaliteye sahip sütunlara one-hot encoding uygulayacağız. Daha sonra, yüksek kardinalite sütunları veri kümesinden kaldırılabilir veya label encoding kullanabiliriz.

Örnek olarak, 10.000 satır içeren ve 100 benzersiz giriş içeren bir kategorik sütun içeren bir veri kümesini düşünün.

* Bu sütun karşılık gelen one-hot encoding ile değiştirilirse, veri kümesine kaç giriş eklenir?
* Bunun yerine sütunu label encoding ile değiştirirsek, kaç giriş eklenir?

Aşağıdaki satırları doldurmak için cevaplarınızı kullanın.

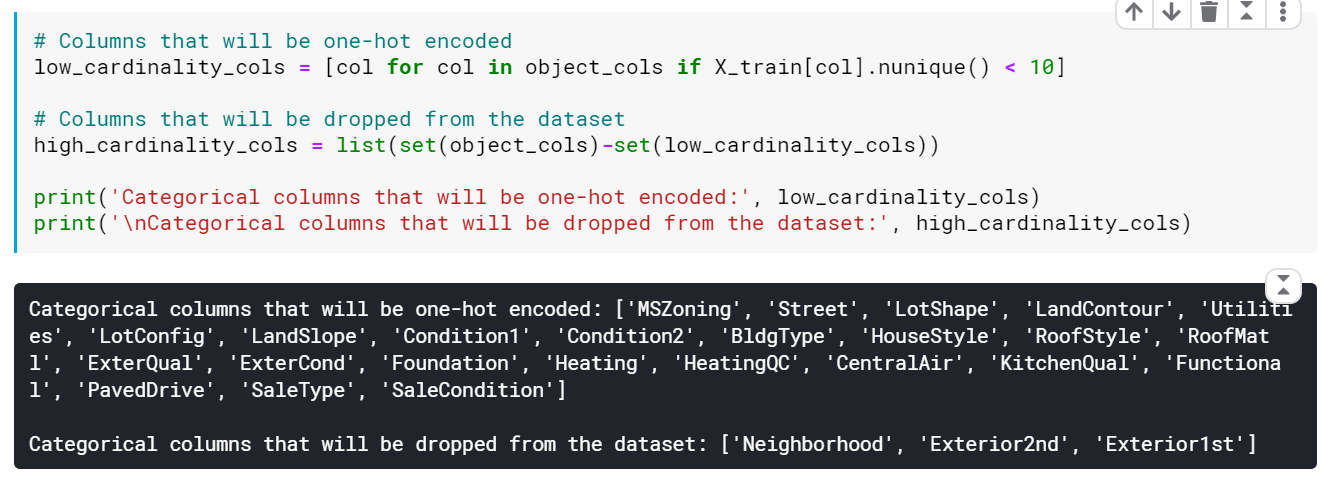


one-hot encoding yoluyla veri kümesine kaç girdi eklendiğini hesaplamak için, kategorik değişkeni kodlamak için kaç girdinin gerekli olduğunu hesaplayarak başlayın (satır sayısını one-hot encoding’deki sütun sayısıyla çarparak). Ardından, veri kümesine kaç girdi eklendiğini öğrenmek için, orijinal sütundaki girdi sayısını çıkarın.

#### Step 4: one-hot encoding

Bu adımda, one-hot encoding deneyeceksiniz. Ancak, veri kümesindeki tüm kategorik değişkenleri kodlamak yerine, kardinalitesi 10'dan az olan sütunlar için yalnızca one-hot encoding oluşturacaksınız.

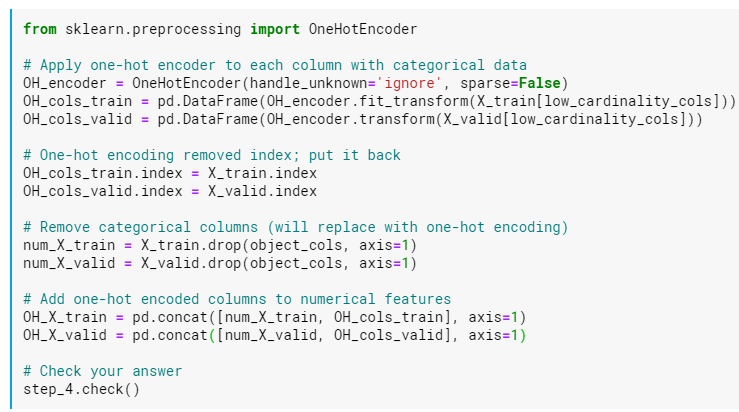
Low\_cardinality\_cols değerini one-hot encoding sütunları içeren bir Python listesine ayarlamak için aşağıdaki kod hücresini değiştirmeden çalıştırın. Benzer şekilde, high\_cardinality\_cols, veri kümesinden bırakılacak kategorik sütunların bir listesini içerir.

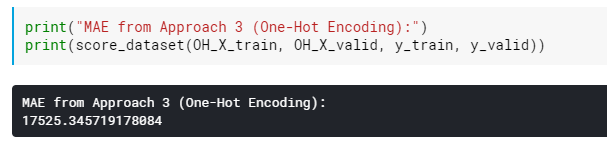


X\_train ve X\_valid içindeki verilere one-hot encoding yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla OH\_X\_train ve OH\_X\_valid olarak ayarlayın.

* Veri kümesindeki kategorik sütunların tam listesi Python listesi object\_cols içinde bulunabilir.
* yalnızca Low\_cardinality\_cols içindeki kategorik sütunlara one-hot encoding uygulanmalı. Diğer tüm kategorik sütunlar veri kümesinden çıkarılmalıdır.

One-hot encoding’i sırasıyla X\_train [low\_cardinality\_cols] ve X\_valid [low\_cardinality\_cols] içindeki eğitim ve doğrulama verilerindeki düşük kardinalite sütunlarına uygulayarak başlayın.





# Kaynaklar

* Kaggle – Intro to Machine Learning Course

<https://www.kaggle.com/learn/intro-to-machine-learning>

* Kaggle – Intermediate Machine Learning Course

<https://www.kaggle.com/learn/intermediate-machine-learning>