# İçindekiler

[İçindekiler 1](#_Toc45406473)

[(Intro to Machine Learning)Makine Öğrenimine Giriş 4](#_Toc45406474)

[How Models Work (Modeller Nasıl Çalışır): 4](#_Toc45406475)

[Giriş: 4](#_Toc45406476)

[Decision Tree’nin Geliştirilmesi 5](#_Toc45406477)

[Basic Data Exploration (Temel Veri Keşfi) 6](#_Toc45406478)

[Using Pandas to Get Familiar With Your Data ( Verilerinizi Öğrenmek için Pandas kullanma): 6](#_Toc45406479)

[Interpreting Data Description (Veri Açıklamalarını Yorumlama): 7](#_Toc45406480)

[Excercise: Explore Your Data 7](#_Toc45406481)

[Your First Machine Learning Model: 9](#_Toc45406482)

[Selecting Data for Modeling (Modelleme için Veri Seçmek): 9](#_Toc45406483)

[Selecting The Prediction Target (Tahmin Hedefini Seçme) 12](#_Toc45406484)

[Choosing "Features" (Özellik Seçimi): 12](#_Toc45406485)

[Building Your Model (Model Oluşturma): 14](#_Toc45406486)

[Exercises: Your First Machine Learning Model 16](#_Toc45406487)

[Model Validation(Model geçerliliği): 19](#_Toc45406488)

[What is Model Validation: (Model Validation Nedir) 19](#_Toc45406489)

[The Problem with "In-Sample" Scores("In-Sample(Örnek İçi)" Puanlarla İlgili Sorun) 21](#_Toc45406490)

[Coding It 21](#_Toc45406491)

[Wow! 22](#_Toc45406492)

[Exercises: Model Validation 22](#_Toc45406493)

[Underfitting and Overfitting 24](#_Toc45406494)

[Experimenting With Different Models 24](#_Toc45406495)

[Examples: 26](#_Toc45406496)

[Sonuç: 27](#_Toc45406497)

[Exercise: Underfitting and Overfitting 28](#_Toc45406498)

[Random Forests: 30](#_Toc45406499)

[Giriş: 30](#_Toc45406500)

[Example 30](#_Toc45406501)

[Sonuç: 31](#_Toc45406502)

[Exercises: Random Forest 31](#_Toc45406503)

[Exercises: Machine Learning Competitions 33](#_Toc45406504)

[Introduction 33](#_Toc45406505)

[Creating a Model For the Competition 34](#_Toc45406506)

[Make Predictions 34](#_Toc45406507)

[Quiz: Intro to Machine Learning 35](#_Toc45406508)

[Intermediate Machine Learning (Orta Düzey Makine Öğrenimi) 44](#_Toc45406509)

[Introduction (Giriş) 44](#_Toc45406510)

[Prerequisites (Önkoşullar) 45](#_Toc45406511)

[Your Turn(Sıra Sende) 45](#_Toc45406512)

[Exercise: Introduction 45](#_Toc45406513)

[Setup 45](#_Toc45406514)

[Missing Values (Eksik Değerler) 49](#_Toc45406515)

[Introduction (Giriş) 49](#_Toc45406516)

[Three Approaches 49](#_Toc45406517)

[Example(Örnek) 50](#_Toc45406518)

[Exercises 53](#_Toc45406519)

[Categorical Variables 58](#_Toc45406520)

[Introduction 58](#_Toc45406521)

[Üç Yaklaşım 58](#_Toc45406522)

[Example 59](#_Toc45406523)

[En iyi yaklaşım hangisi? 62](#_Toc45406524)

[Sonuç 63](#_Toc45406525)

[Exercises 63](#_Toc45406526)

[Pipelines 70](#_Toc45406527)

[Introduction 70](#_Toc45406528)

[Example 70](#_Toc45406529)

[Sonuç 73](#_Toc45406530)

[Exercise: Pipelines 73](#_Toc45406531)

[Cross-Validation 76](#_Toc45406532)

[Introduction 76](#_Toc45406533)

[Cross-Validation Nedir? 77](#_Toc45406534)

[Ne Zaman Cross-Validation Kullanmalıyız? 78](#_Toc45406535)

[Example 78](#_Toc45406536)

[Sonuç 79](#_Toc45406537)

[Exercise: Cross-Validation 79](#_Toc45406538)

[KAYNAKLAR 86](#_Toc45406539)

# (Intro to Machine Learning)Makine Öğrenimine Giriş

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.

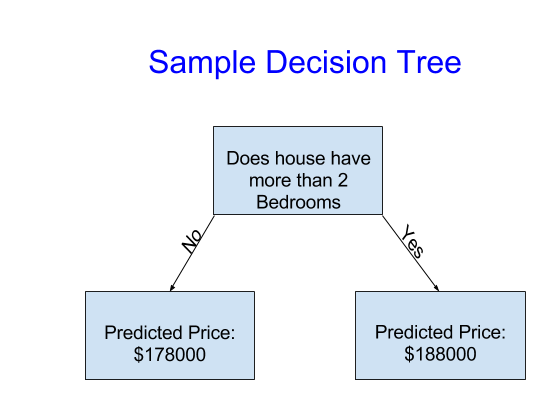
Bu mikro kurs, aşağıdaki senaryodan geçerken modeller oluşturmanızı sağlayacaktır:

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. Karar Ağacı **(Decision Tree)** adlı bir modelle başlayacağız. Daha doğru tahminler veren meraklı modeller var. Ancak karar ağaçları’nın anlaşılması kolaydır ve bunlar veri bilimindeki en iyi modellerin bazıları için temel yapı taşıdır.

Basitlik için, mümkün olan en basit karar ağacıyla başlayacağız.



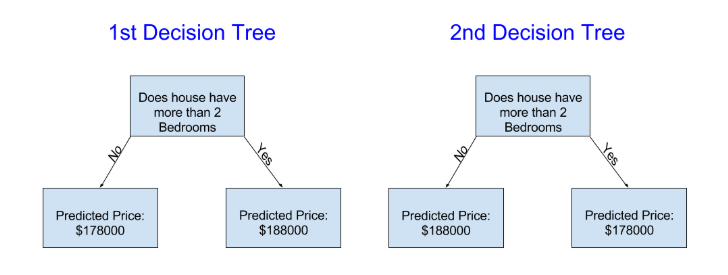
Evleri sadece iki kategoriye ayırır. Dikkate alınan herhangi bir ev için tahmini fiyat, aynı kategorideki evlerin tarihsel ortalama fiyatıdır.

Verileri, evlerin iki gruba nasıl ayrılacağına karar vermek için ve sonra her grupta öngörülen fiyatı belirlemek için kullanıyoruz. Verilerden **pattern(desen)** 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(tahmin)** edebilmek için yeni verilere uygulayabilirsiniz.

### Decision Tree’nin Geliştirilmesi

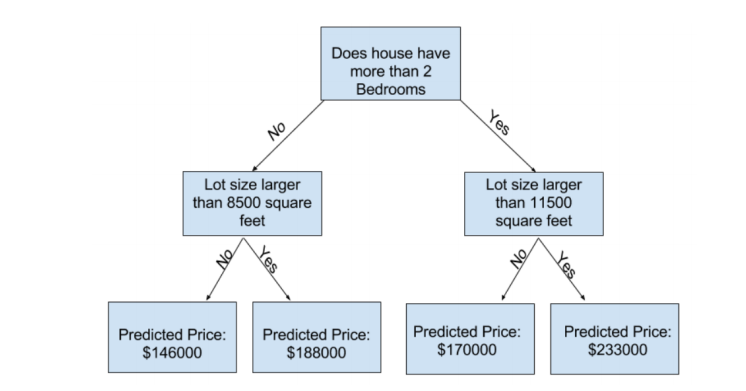
Aşağıdaki iki karardan hangisinin gayrimenkul eğitim verilerinin fit edilmesinden kaynaklanması daha olasıdır?



Soldaki karar ağacı (Karar Ağacı 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, yer 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.

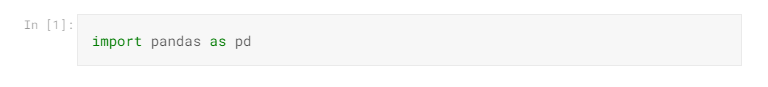
Altta tahmin yaptığımız noktaya **leaf(yaprak)** denir.

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

## Basic Data Exploration (Temel Veri Keşfi)

### Using Pandas to Get Familiar With Your Data ( Verilerinizi Öğrenmek için Pandas kullanma):

Herhangi bir makine öğrenimi projesinin ilk adımı, verileri tanımaktır. Bunun için Pandas kütüphanesini kullanacaksınız. **Pandas**, bilim insanlarının verileri keşfetmek ve işlemek için kullandığı temel araçtır. Çoğu kişi **Pandas** kodlarında **pd** olarak kısaltılır. 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 Lowa'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 inceleriz:

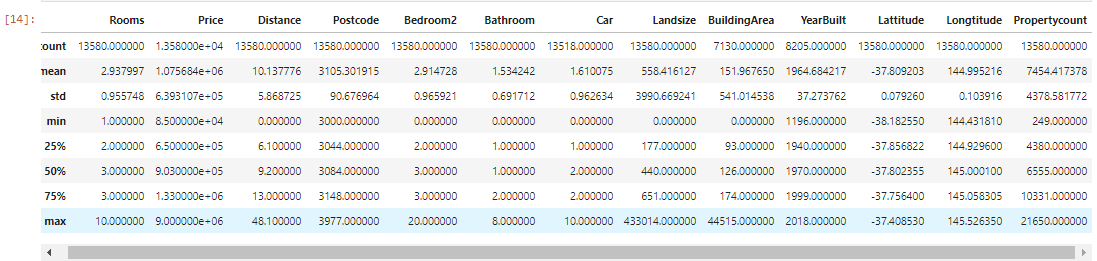
* **# kolay erişim için dosya yolunu değişkene kaydet**



* **# verileri okuyun ve DataFrame'de melbourne \_data başlıklı verileri depolayın**







### Interpreting Data Description (Veri Açıklamalarını Yorumlama):

Sonuçlar, orijinal veri kümenizdeki her 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, ortalama olan **mean'dir**.

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

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

İlk (en küçük) değer **min**. Liste'nin dörtte birini incelerseniz, değerlerin **% 25**'inden daha büyük ve değerlerin **% 75**'inden daha küçük bir sayı bulacaksınız.

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

### Excercise: Explore Your Data

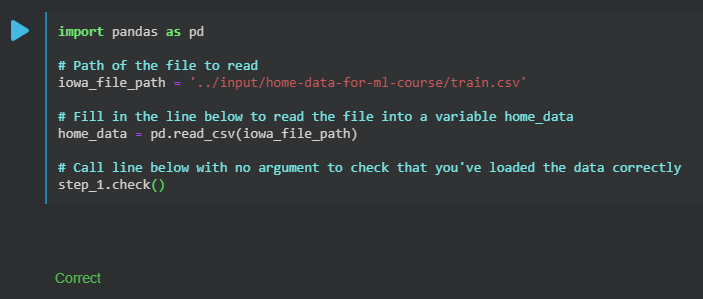
Bu alıştırma, bir veri dosyasını okuma ve verilerle ilgili istatistikleri anlama yeteneğinizi test edecektir.

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.

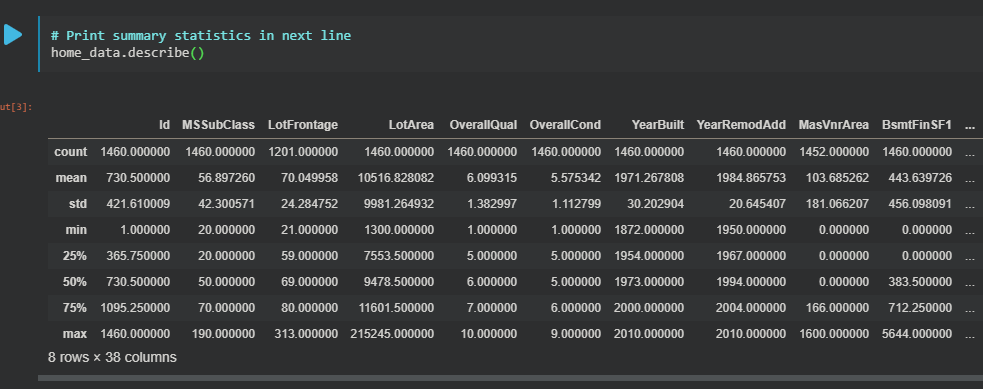
#### **Step 1: Loading Data (Veri Yükleme)**

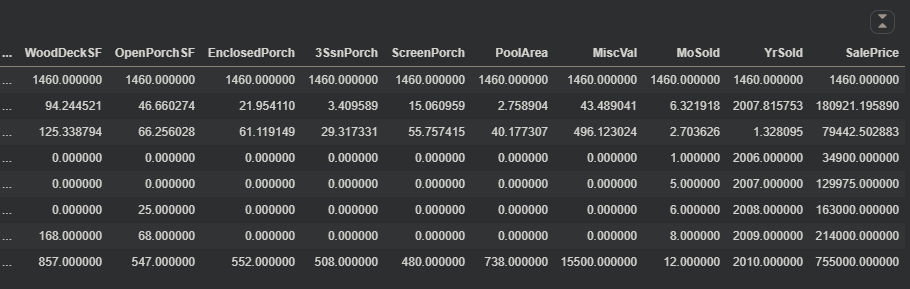
Iowa veri dosyasını home\_data adlı bir Pandas DataFrame'de okuyun.

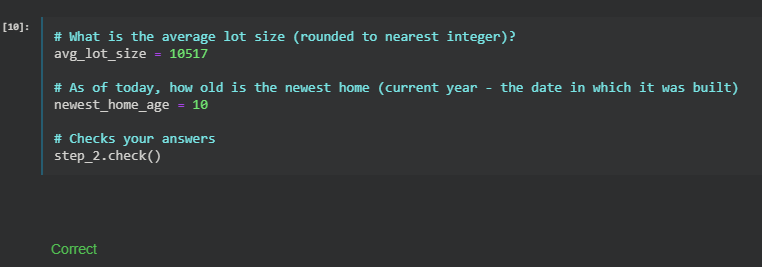


#### **Step 2: Review The Data (Verileri Gözden Geçirme)**

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?

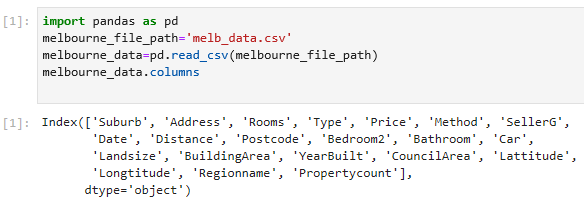
## Your First Machine Learning Model:

### Selecting Data for Modeling (Modelleme için Veri Seçmek):

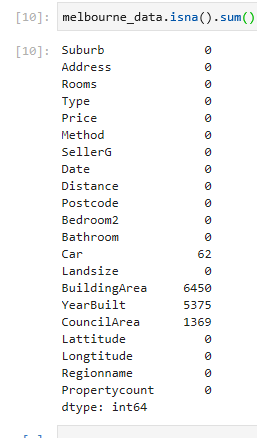
Veri kümenizin, kafanızda canlanması veya güzelce ekrana yazdırmak için çok fazla değişkeni vardı. Bu başa çıkılamaz veri miktarını anlayabileceğiniz bir şeye nasıl ayırabilirsiniz?

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.

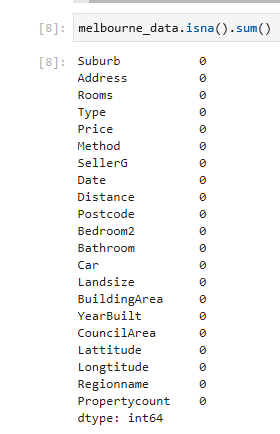
# Lowa verileriniz, kullandığınız sütunlarda eksik değerlere sahip değildi.

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

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



# Column’ların içinde kaçar tane eksik veri var ona baktık.



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

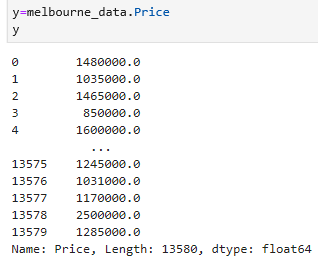
### Selecting The Prediction Target (Tahmin Hedefini Seçme)

**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.

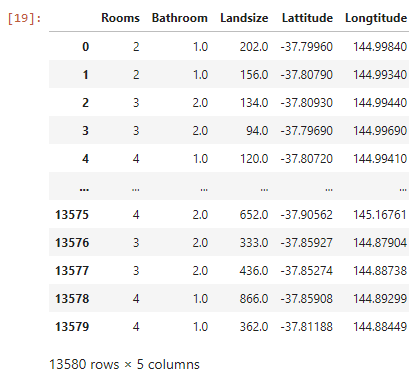
Köşeli parantez içine sütun adlarının listesini yazarak birden fazla özellik seçiyoruz. Bu listedeki her öğe bir string (tırnak işaretli) olmalıdır.

Here is an example:

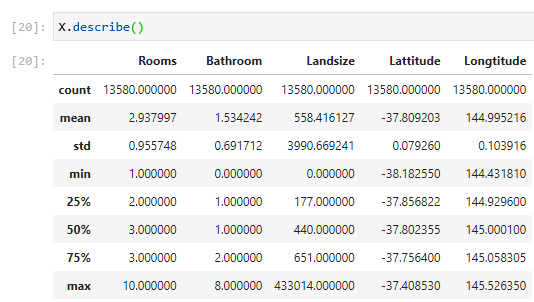


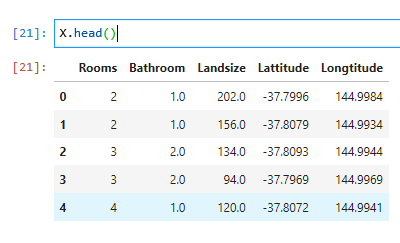
Kural olarak, bu verilere X denir.





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.

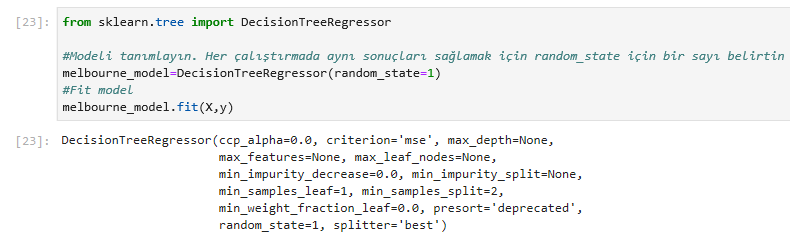
**Scikit-learn**, tipik olarak **DataFrames**'da depolanan veri türlerini modellemek için en popüler kütüphanedir.

**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 .

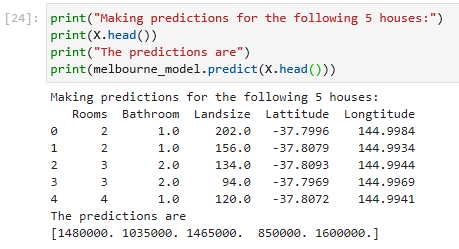
Birçok makine öğrenimi modeli, model eğitiminde bazı rasgeleliklere izin verir.

**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.

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.

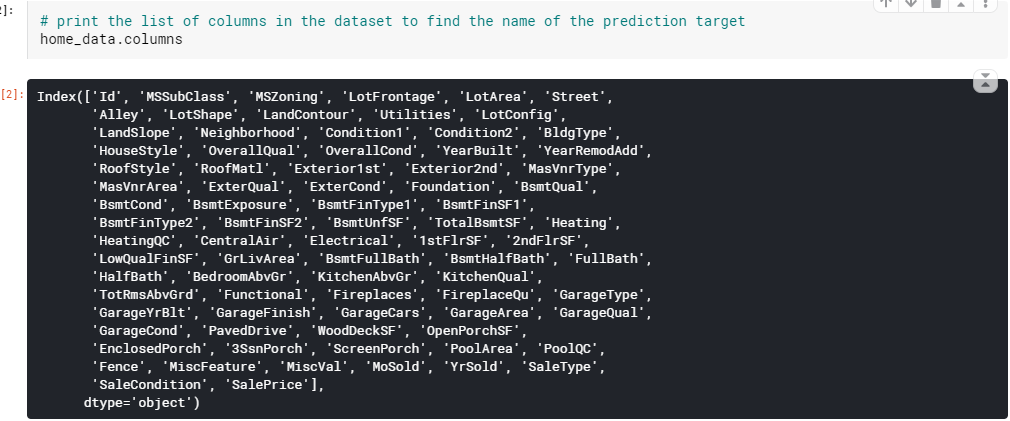


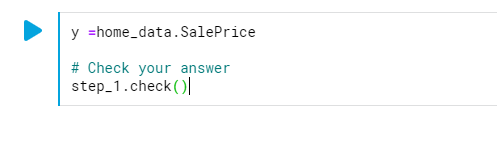
### Exercises: Your First Machine Learning Model

#### **Step 1: Specify Prediction Target: (Tahmin Hedefi Belirtme)**

"Satış fiyatı(sales price)" na karşılık gelen hedef değişkeni seçin. Bunu "y" adlı yeni bir değişkene kaydedin. İhtiyacınız olan sütunun adını bulmak için sütunların bir listesini yazdırmanız gerekir.

# tahmin hedefinin adını bulmak için veri kümesindeki sütunların listesini yazdır





# Aşağıdaki satırlar size bir ipucu veya çözüm gösterecektir.

# step\_1.hint()

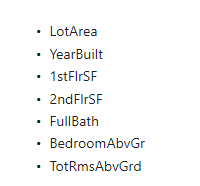
# step\_1.solution()

#### **Step 2: Create X**

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 :

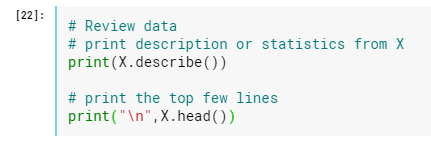


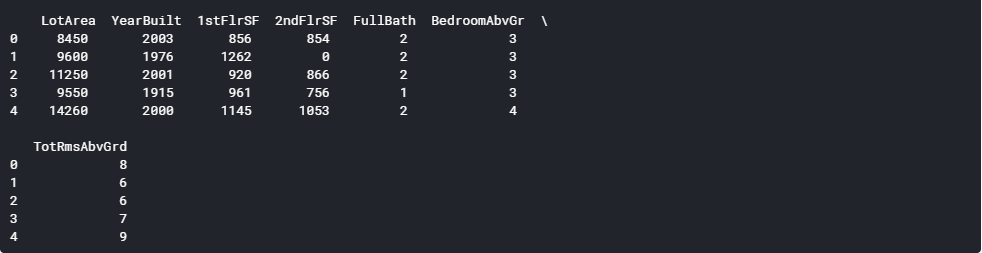
Bu özellik listesini oluşturduktan sonra, modeli fit etmek için kullanacağınız DataFrame'i oluşturmak için kullanın.



#### **Review Data**

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



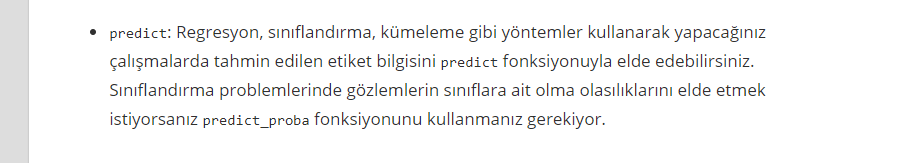
#### **Step 3: Specify and Fit Model:**

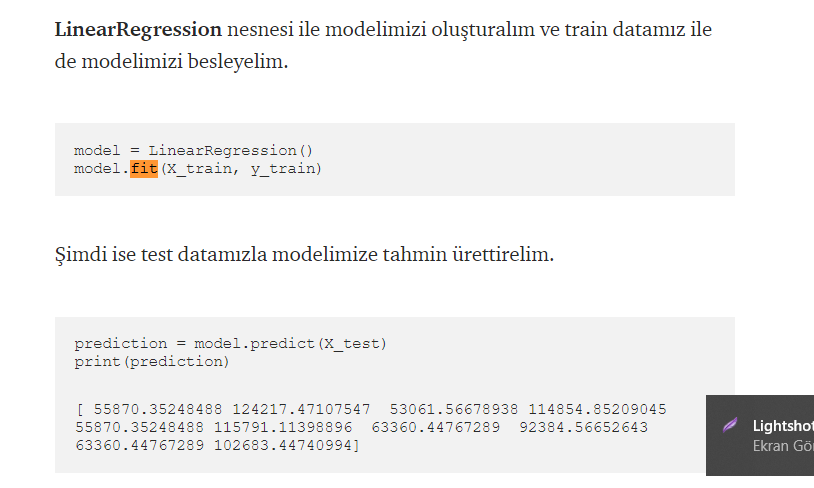
#### 

#### 

#### **Step 4: Make Predictions: (Tahmin Yapma)**

**Notlar:**





## Model Validation(Model geçerliliği):

Bir model oluşturdunuz. Ama ne kadar iyi?

Bu derste, modelinizin kalitesini ölçmek için model doğrulamayı kullanmayı öğreneceksiniz.

Model kalitesini ölçmek, modellerinizi tekrar tekrar geliştirmenin anahtarıdır.

### What is Model Validation: (Model Validation Nedir)

Oluşturduğunuz hemen hemen her modeli değerlendirmek isteyeceksiniz. Çoğu (hepsi olmasa da) uygulamada, model kalitesinin ilgili ölçüsü tahmini doğruluktur. Başka bir deyişle, modelin tahminleri gerçekte olanlara yakın olacak mı?

Birçok kişi tahmini doğruluğu ölçerken büyük bir hata yapar. "Training data" ile tahminler yaparlar ve bu tahminleri "Tranining data" daki hedef değerlerle karşılaştırırlar.

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

Önce model kalitesini anlaşılabilir 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 metrik olarak özetlemeliyiz.

Model kalitesini özetlemek için birçok metrik vardır, ancak **Mean Absolute Error**(ortalama mutlak hata) **(MAE** olarak da adlandırılır) olarak adlandırılan biriyle başlayacağız.

Bu metriği son kelimeyle başlayarak parçalayalım, hata.

Her ev için tahmin hatası:

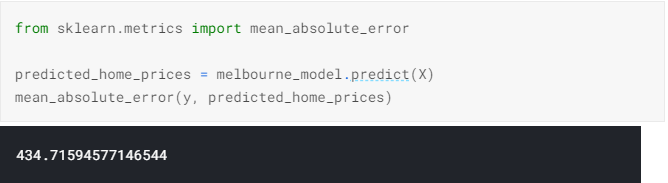


Yani, eğer bir ev 150.000 dolara mal olursa ve bunun 100.000 dolara mal olacağını tahmin ettiyseniz, hata 50.000 dolar. MAE metriği ile, her hatanın mutlak değerini alırız. Bu, her hatayı pozitif bir sayıya dönüştürür. Daha sonra bu mutlak hataların ortalamasını alırız. Bu bizim model kalitesinin ölçüsüdür. Düz İngilizce olarak, şu şekilde söylenebilir:

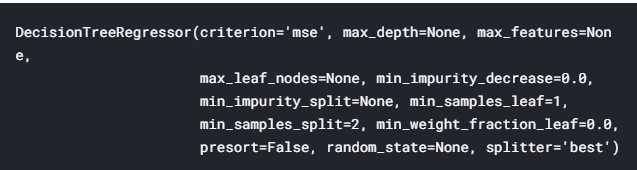
Ortalama olarak, tahminlerimiz yaklaşık X civarında.( On average, our predictions are off by about X.)

MAE'Yİ hesaplamak için önce bir modele ihtiyacımız var. Bu, code düğmesine tıklayarak inceleyebileceğiniz aşağıdaki gizli bir hücreye yerleştirilmiştir.

Bir modelimiz olduğunda, ortalama mutlak hatayı nasıl hesaplarız:







### The Problem with "In-Sample" Scores("In-Sample(Örnek İçi)" Puanlarla İlgili Sorun)

Sadece hesapladığımız ölçü "örnek" puanı olarak adlandırılabilir. Hem modeli oluşturmak hem de değerlendirmek için tek bir "örnek" ev kullandık. İşte bu yüzden kötü.

Büyük emlak piyasasında, kapı rengi ev fiyatı ilgisiz olduğunu düşünün. Ancak, modeli oluşturmak için kullandığınız veri örneğinde, yeşil Kapılı tüm evler çok pahalıydı. Modelin işi, ev fiyatlarını tahmin eden **desenleri(patterns)** bulmaktır, bu yüzden bu deseni görecektir ve her zaman yeşil Kapılı evler için yüksek fiyatları tahmin edecektir.

Bu model eğitim verilerinden türetildiğinden, model eğitim verilerinde doğru görünecektir.

Ancak, model yeni veriler gördüğünde bu örüntü tutmazsa, model pratikte kullanıldığında çok yanlış olur.

Modellerin pratik değeri yeni veriler üzerinde tahmin yapmaktan geldiğinden, modeli oluşturmak için kullanılmayan veriler üzerindeki performansı ölçüyoruz.

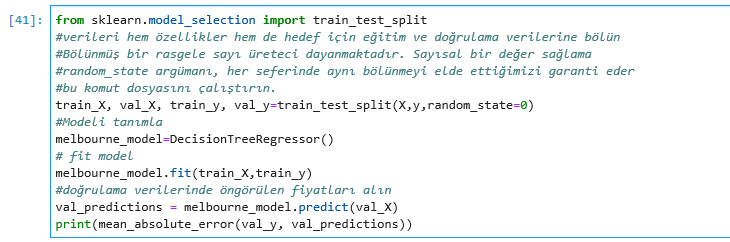
Bunu yapmanın en basit yolu, bazı verileri model oluşturma sürecinden dışlamak ve daha önce görmediği veriler üzerindeki modelin doğruluğunu test etmek için bunları kullanmaktır.Bu verilere **validation data** denir.

### Coding It

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

Bu verilerin bir kısmını modele uyacak şekilde eğitim verileri olarak kullanacağız ve diğer verileri **mean\_absolute\_error** hesaplamak için doğrulama verileri olarak kullanacağız.

Here is the code:





### Wow!

Örnek içi veriler için ortalama mutlak hatanız yaklaşık 500 dolardı. Örnek dışı 250.000 dolardan fazla.

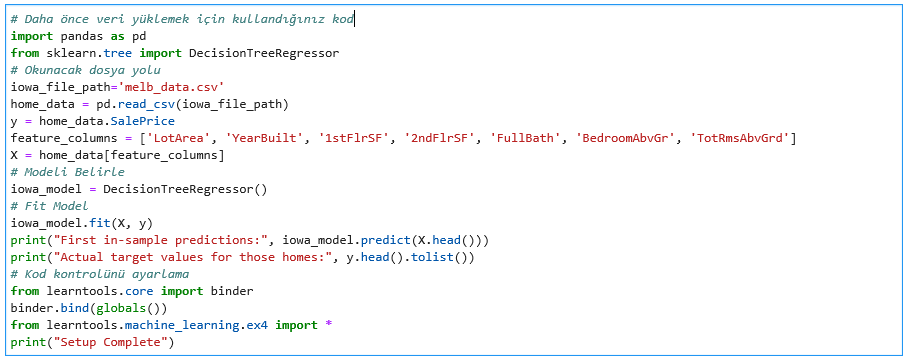
Bu, neredeyse tamamen doğru olan bir model ile en pratik amaçlar için kullanılamayan bir model arasındaki farktır. Bir referans noktası olarak, doğrulama verilerindeki ortalama ev değeri 1,1 milyon dolar. Yani yeni verilerdeki hata ortalama ev değerinin dörtte biri kadardır.

Daha iyi özellikler veya farklı model türleri bulmak için deneme yapmak gibi bu modeli geliştirmenin birçok yolu vardır.

### Exercises: Model Validation

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

Önceki alıştırmanın kaldığı kodlama ortamınızı ayarlamak için aşağıdaki hücreyi çalıştırın.



#### **Step 1: Split Your Data(Verilerinizi Bölün)**

Verilerinizi bölmek için train\_test\_split işlevini kullanın.

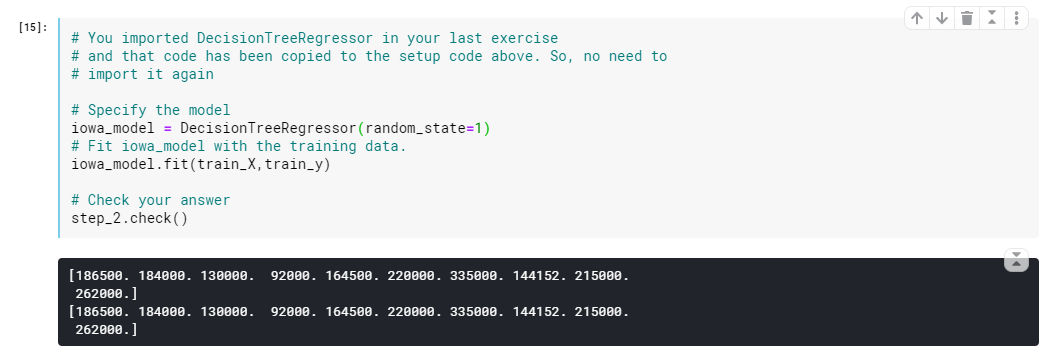
Random\_state = 1 argümanını verin, böylece kontrol fonksiyonları kodunuzu doğrularken ne bekleyeceğini bilir.

Geri çağırma, Özellikleri Veri Çerçevesi x yüklenir ve hedef yüklenir senin.

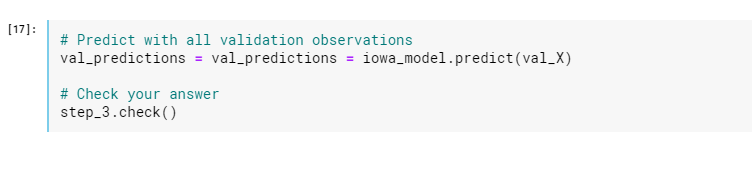


#### **Step 2: Specify and Fit the Model**

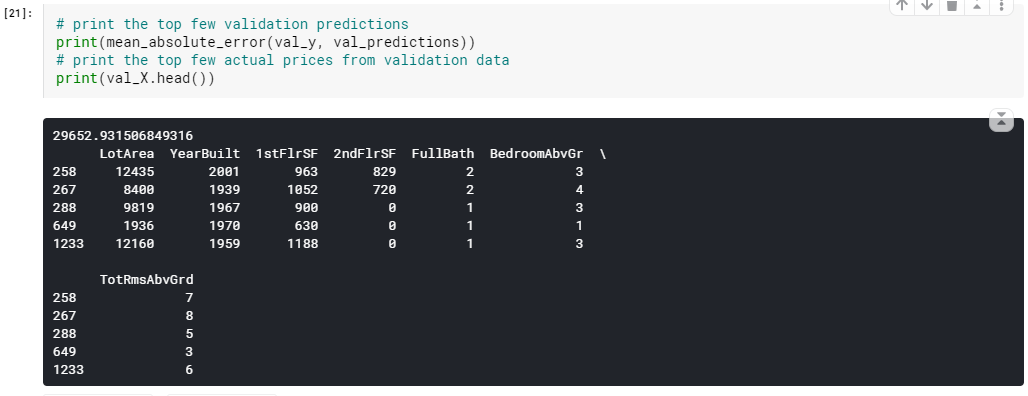
Bir DecisionTreeRegressor modeli oluşturun ve ilgili verilere uydurun. Modeli oluştururken random\_state öğesini tekrar 1 olarak ayarlayın.



#### **Step 3: Make Predictions with Validation data(Doğrulama verileriyle tahminler yapın)**



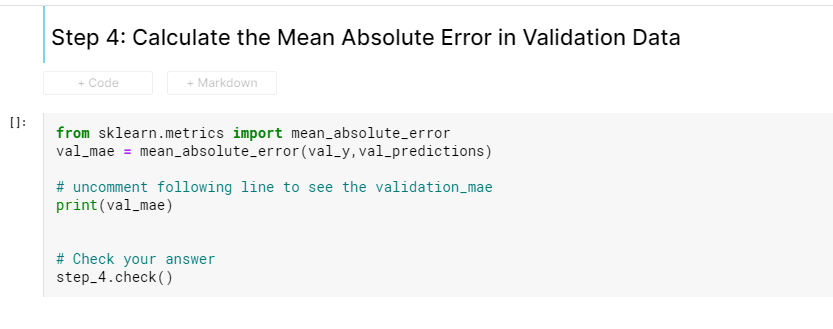
Doğrulama verilerinden tahminlerinizi ve gerçek değerlerinizi inceleyin.



İçinde gördüğünüzden farklı olan ne fark edersiniz-örnek tahminler (bu sayfadaki en üst kod hücresinden sonra yazdırılır).

Doğrulama tahminlerinin neden örnek içi (veya eğitim) tahminlerinden farklı olduğunu hatırlıyor musunuz? Bu son dersten önemli bir fikir.

#### **Step 4: Calculate the Mean Absolute Error in Validation Data(doğrulama verilerinde ortalama mutlak hatayı hesaplayın)**



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

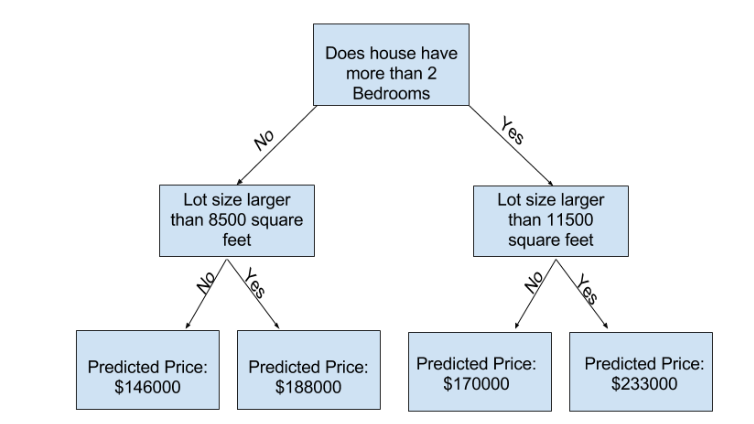
Bu adımın sonunda, uygun olmayan ve fazla uygunluk kavramlarını anlayacak ve modellerinizi daha doğru hale getirmek için bu fikirleri uygulayabileceksiniz.

### Experimenting With Different Models

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 documentation'da(<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>)  
karar ağacı modelinin birçok seçeneğe sahip olduğunu görebilirsiniz(uzun süre isteyeceğinizden veya ihtiyaç duyacağınızdan daha fazla).

En önemli seçenekler ağacın derinliğini belirler. Bu mikro kurstaki ilk dersten, bir ağacın derinliğinin, bir tahmine gelmeden önce kaç bölmenin yaptığının bir ölçüsü olduğunu hatırlayın. Bu nispeten derin olmayan bir ağaçtır.



Uygulamada, bir ağacın üst seviye (tüm evler) ve bir yaprak arasında 10 bölünmesi nadir değildir.

Ağaç derinleştikçe, veri kümesi daha az ev ile yapraklara dilimlenir. Bir ağacın yalnızca 1 bölünmesi varsa, verileri 2 gruba böler. Her grup tekrar bölünürse, 4 grup ev alırız. Bunların her birini tekrar bölmek 8 grup oluşturacaktır. Her seviyede daha fazla bölme ekleyerek grup sayısını ikiye katlamaya devam edersek, 10. seviyeye geldiğimizde 210 grup evimiz olacak. Bu 1024 yaprak yapar.

Evleri birçok yaprak arasında böldüğümüzde, her yaprakta daha az evimiz var. Çok az ev bulunan yapraklar, bu evlerin gerçek değerlerine oldukça yakın tahminler yapacaktır, ancak yeni veriler için çok güvenilmez tahminler yapabilirler(çünkü her tahmin sadece birkaç eve dayanmaktadır).

Bu, bir modelin eğitim verileriyle neredeyse mükemmel bir şekilde eşleştiği, ancak doğrulama ve diğer yeni verilerde zayıf olduğu **overfitting** adlı bir olgudur.

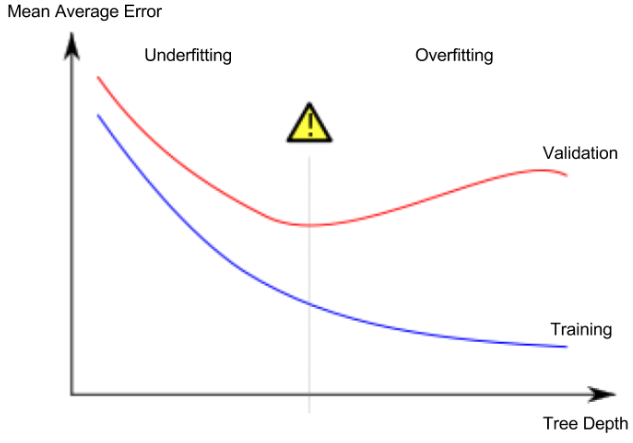
Ters tarafta, eğer ağacımızı çok yüzeysel yaparsak, evleri çok farklı gruplara ayırmaz.

Aşırı derecede, eğer bir ağaç evleri sadece 2 veya 4'e bölerse, her grup hala çok çeşitli evlere sahiptir.

Ortaya çıkan tahminler, eğitim verilerinde bile çoğu ev için çok uzak olabilir (ve aynı nedenden dolayı doğrulamada da kötü olacaktır). Bir model verilerde önemli ayrımlar(import distinctions) ve desenler(patterns) yakalamak için başarısız olduğunda, bu yüzden bile eğitim verilerinde(training data) kötü performans, bu underfitting denir.

Doğrulama verilerimizden tahmin ettiğimiz yeni veriler üzerindeki doğruluğu önemsediğimizden, **underfitting** ve **overfitting** arasındaki en etkili noktayı(sweet spot) bulmak istiyoruz.

Görsel olarak, (kırmızı) doğrulama eğrisinin düşük noktasını istiyoruz



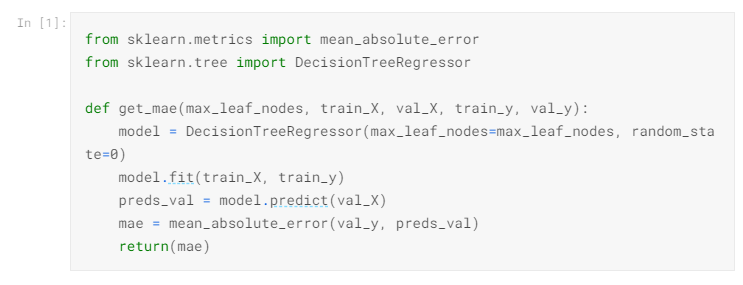
### Examples:

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

Ancak **max\_leaf\_nodes** argümanı overfitting vs underfitting kontrol etmek için çok mantıklı bir yol sağlar.

Modelin yapmasına izin verdiğimiz daha fazla yaprak, yukarıdaki grafikteki underfitting alanından overfitting alanına daha fazla hareket ederiz.

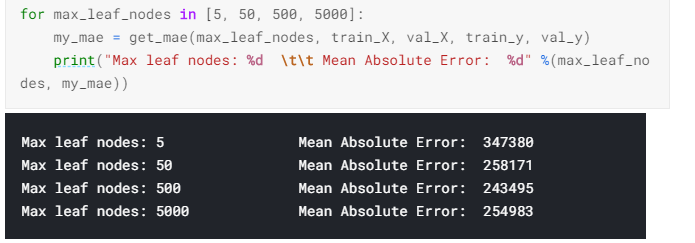
**Max\_leaf\_nodes** için farklı değerlerden Mae puanlarını karşılaştırmaya yardımcı olmak için bir yardımcı program işlevi kullanabiliriz:



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 oluşturulmuş modellerin doğruluğunu karşılaştırmak için bir for-loop kullanabiliriz.

# mae'yi farklı max\_leaf\_nodes değerleriyle karşılaştırın



Listelenen seçeneklerden 500, en uygun yaprak sayısıdır.

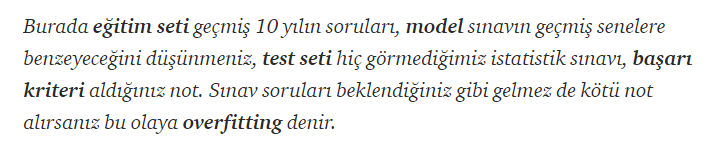
### Sonuç:

Modeller şunlardan herhangi birine sahip olabilir.

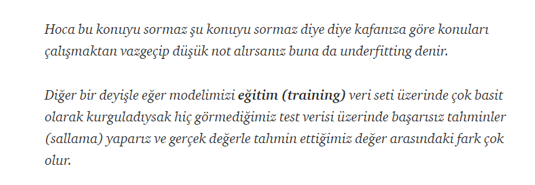
* Overfitting: gelecekte tekrarlamayacak sahte pattern(desen)leri yakalamak , daha az doğru tahminlere yol açmak veya
* Underfittin: alakalı pattern’leri yakalayamama, yine daha az doğru tahminlere yol açma.

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

**Overfitting:**



**Underfitting:**



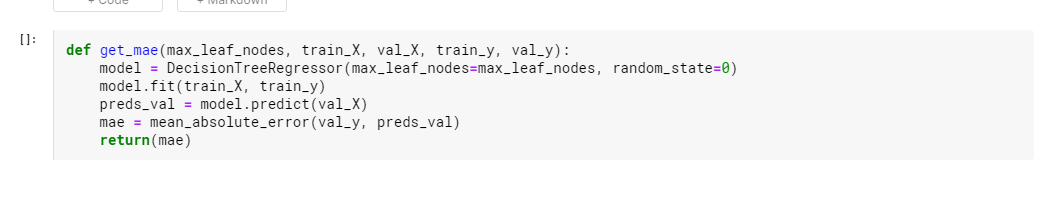
### Exercise: Underfitting and Overfitting

#### **Tekrarlamak:**

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.



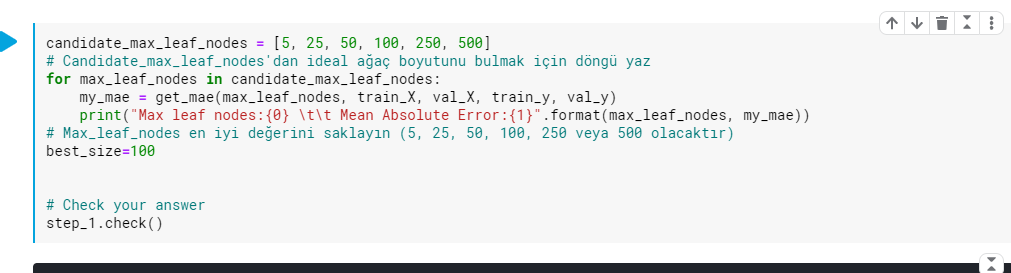
Get\_mae fonksiyonunu kendiniz yazabilirsiniz. Şimdilik tedarik edeceğiz. Bu, bir önceki derste okuduğunuz işlevle aynıdır. Aşağıdaki hücreyi çalıştırmanız yeterlidir.

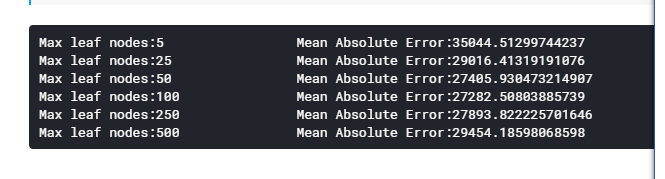


#### **Step 1: Compare Different Tree Sizes**

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.

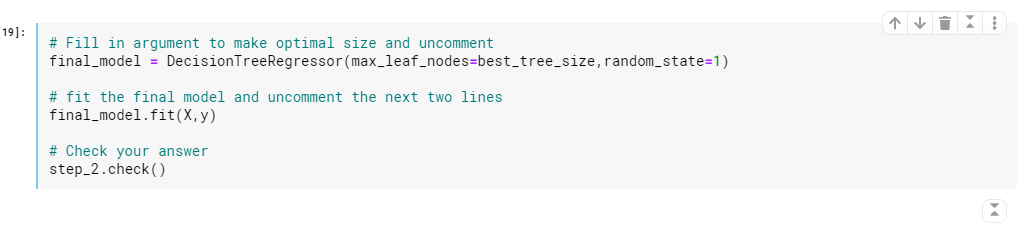




#### **Step 2: Fit Model Using All Data(Modeli tüm verileri kullanarak sığdır)**

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:

### Giriş:

**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

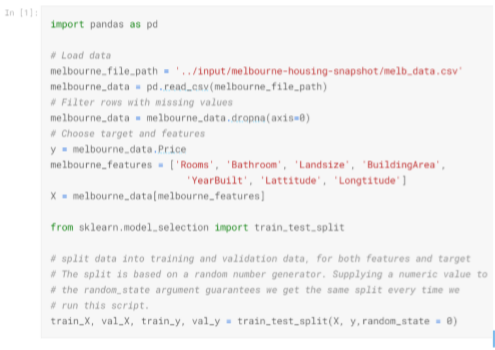
Verileri yüklemek için gereken kodu zaten birkaç kez gördünüz. Veri yüklemenin sonunda aşağıdaki değişkenler bulunur:

• train\_X

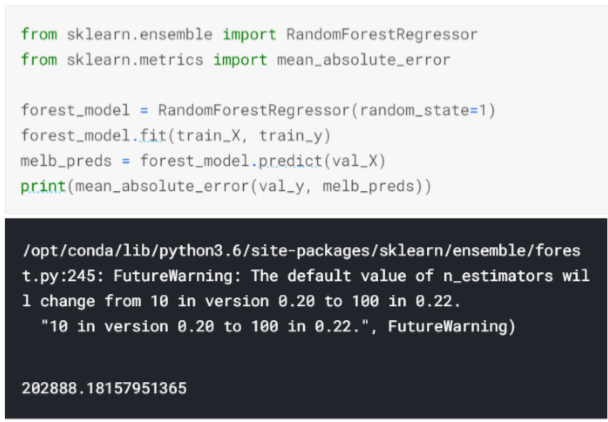
• val\_X

• 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.

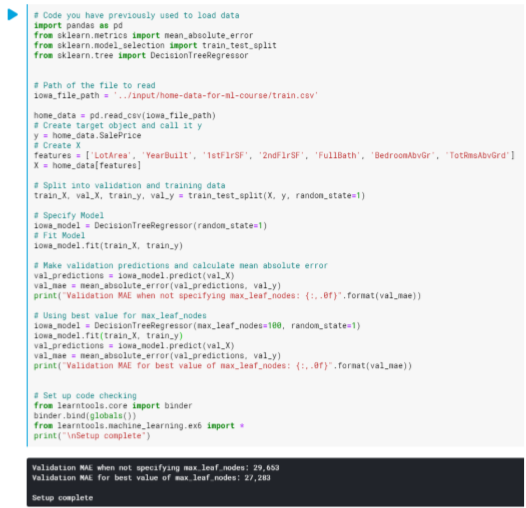
Single decision tree’nin maksimum derinliğini değiştirdiğimiz gibi Random Forest'ın da performansını değiştirmenize izin veren parametreler var.

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

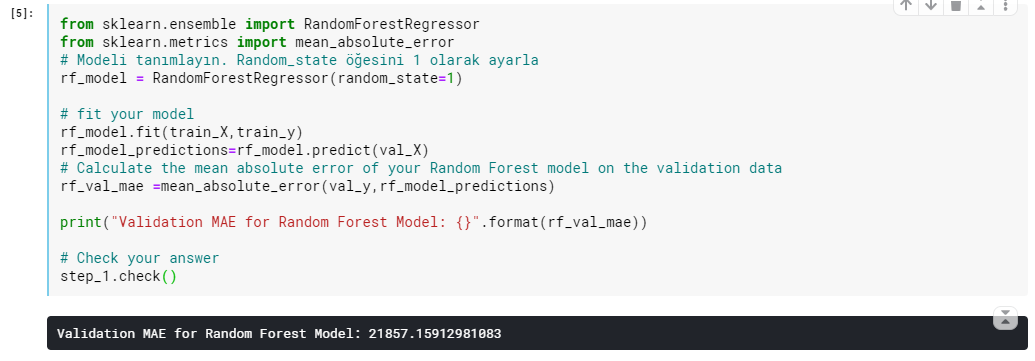
Şimdiye kadar yazdığımız kod:



#### **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.



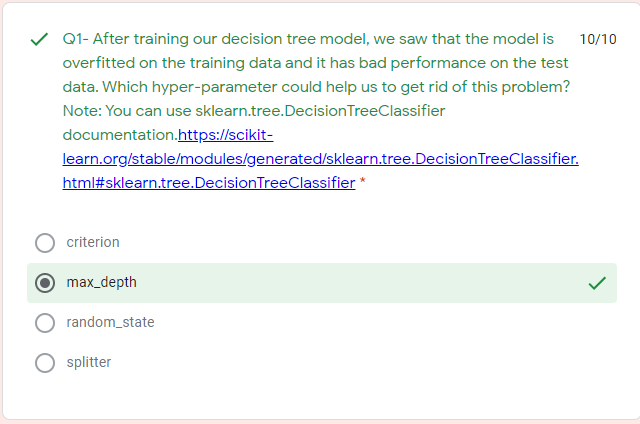
### Make Predictions

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



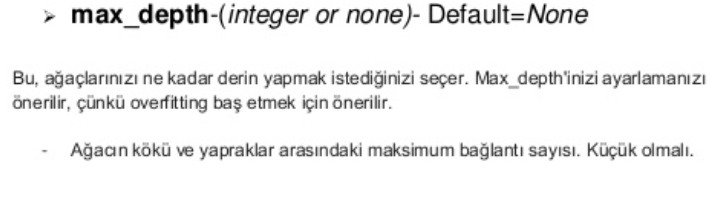
Modelinizi geliştirmenin birçok yolu vardır ve deneme yapmak bu noktada öğrenmenin harika bir yoludur. Modelinizi geliştirmenin en iyi yolu özellikler eklemektir. Sütun listesine bakın ve konut fiyatlarını nelerin etkileyebileceğini düşünün. Bazı özellikler, eksik değerler veya sayısal olmayan veri türleri gibi sorunlar nedeniyle hatalara neden olur.

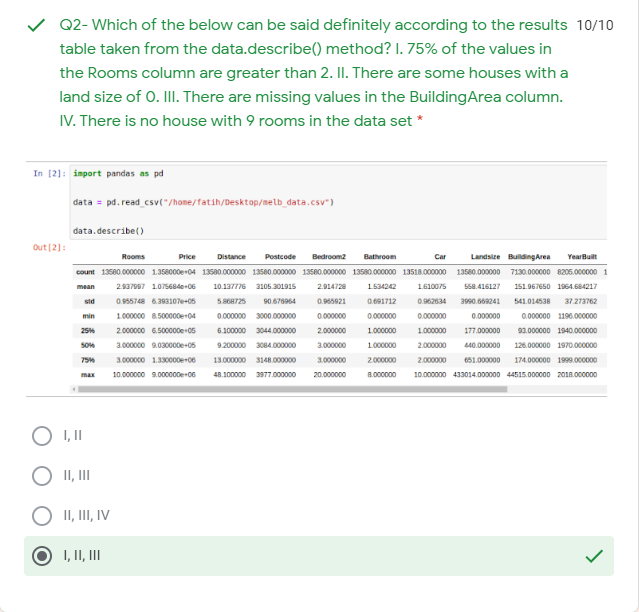
### Quiz: Intro to Machine Learning



#### AÇIKLAMA:

Karar ağacı modelimizi eğittikten sonra, modelin eğitim verilerine fazla uyduğunu ve test verilerinde kötü performans gösterdiğini gördük. Hangi hiper parametre bu problemden kurtulmamıza yardımcı olabilir? Not: sklearn.tree.DecisionTreeClassifier belgelerini kullanabilirsiniz. Http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier \*

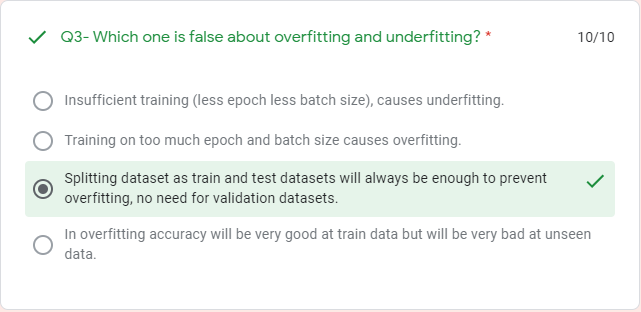




#### AÇIKLAMA

Aşağıdaki data.describe() method’undan alınan sonuç tablosuna göre kesinlikle söylenebilir?

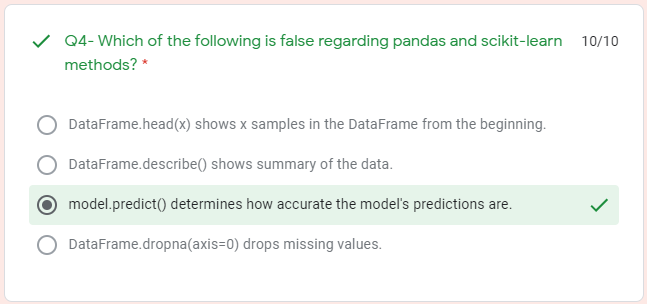
1. Rooms sütunundaki değerlerin %75'i 2'den büyüktür.
2. II. land size'ı 0 olan bazı evler vardır.
3. III. BuildingArea sütununda eksik değerler vardır.
4. IV. veri kümesinde 9 odalı bir ev yoktur



#### AÇIKLAMA

Overfitting ve underfitting için hangisi yanlıştır?

* Yetersiz eğitim (daha az epoch daha az küme boyutu), underfitting'e neden olur.
* Çok fazla epoch ve küme boyutu üzerinde eğitim overfitting neden olur.
* Veri kümesini train ve test veri kümeleri olarak bölmek, validation veri kümelerine gerek kalmadan overfitting önlemek için her zaman yeterli olacaktır.
* Overfitting doğruluğu train verilerinde çok iyi olacak, ancak unseen(görülmeyen) verilerde çok kötü olacak.

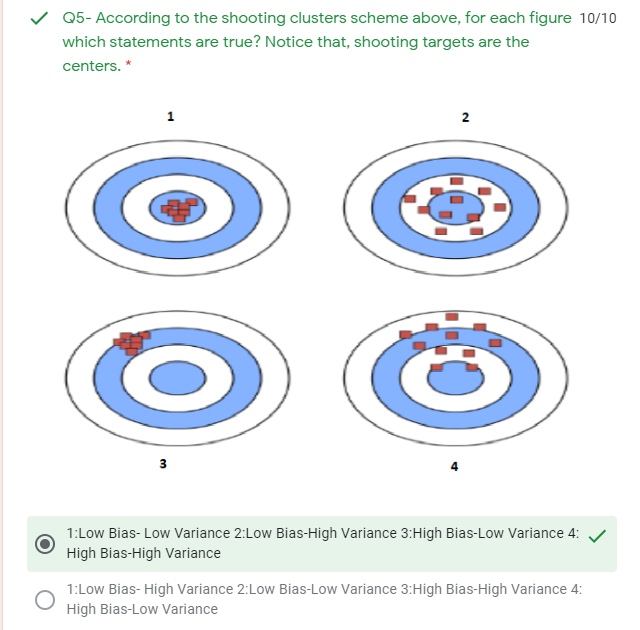


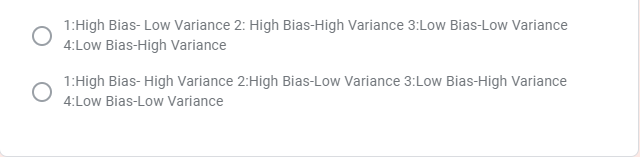
#### AÇIKLAMA

Pandas ve scikit-learn yöntemleri ile ilgili aşağıdakilerden hangisi yanlıştır?

* DataFrame.head (x), DataFrame x örneklerini baştan gösterir.
* DataFrame.describe () verilerin özetini gösterir.
* model.predict (), modelin tahminlerinin ne kadar doğru olduğunu belirler.
* DataFrame.dropna (axis=0) eksik değerleri düşürür.



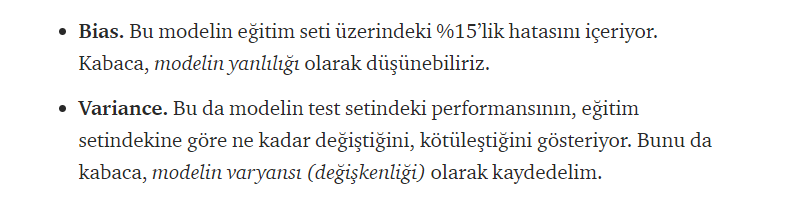


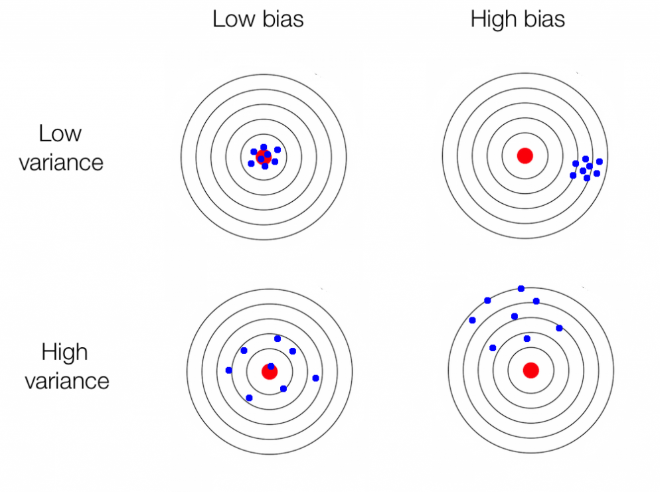


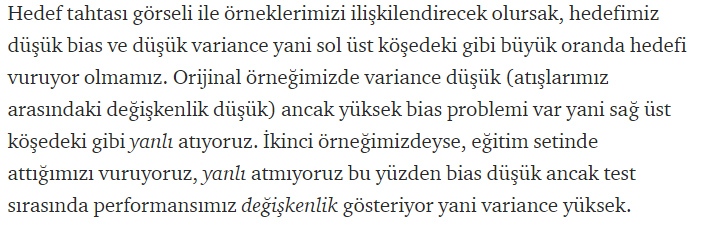
#### AÇIKLAMA

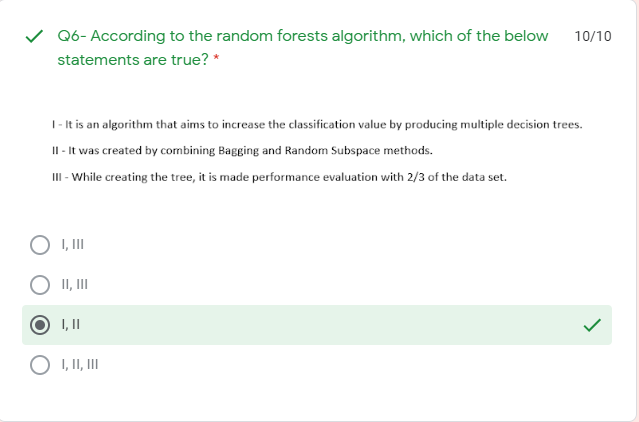
Yukarıdaki shooting clusters şemasına göre, her şekil için hangi ifadeler doğrudur? Dikkat edin, shooting targets merkezlerdir.

* 1: Düşük Bias-Düşük Varyans 2: Bias-Yüksek Varyans 3: Yüksek Bias-Düşük Varyans 4: Yüksek Bias-Yüksek Varyans
* 1: Düşük Bias-Yüksek Varyans 2: Düşük Bias-Düşük Varyans 3: Yüksek Bias-Yüksek Varyans 4: Yüksek Bias-Düşük Varyans
* 1: Yüksek Bias-Düşük Varyans 2: Yüksek Bias-Yüksek Varyans 3: Düşük Bias-Düşük Varyans 4: Düşük Bias-Yüksek Varyans
* 1: Yüksek Bias-Yüksek Varyans 2: Yüksek Bias-Düşük Varyans 3: Düşük Bias-Yüksek Varyans 4: Düşük Bias-Düşük Varyans





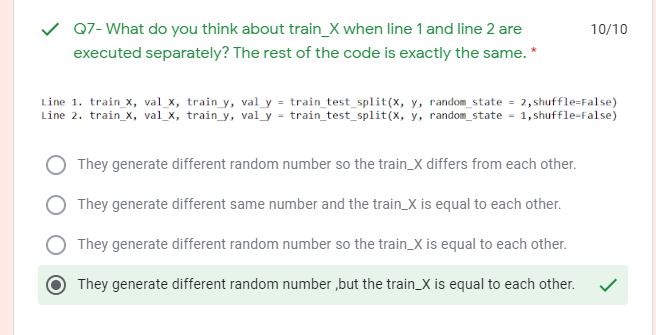




#### AÇIKLAMA

Random forests algoritmasına göre, aşağıdaki ifadelerden hangisi doğrudur?

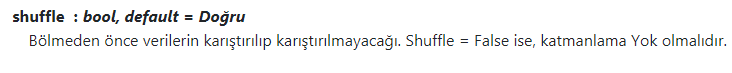
* Birden fazla karar ağacı üreterek sınıflandırma değerini arttırmayı amaçlayan bir algoritmadır
* Bagging ve Rastgele Subspace yöntemlerini birleştirerek oluşturuldu
* Ağacı oluştururken veri kümesinin 2 / 3'ü ile değerlendirme yapılır

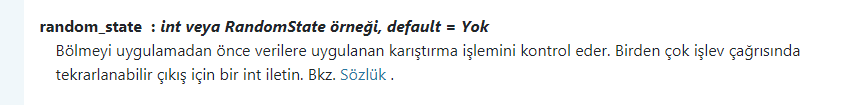


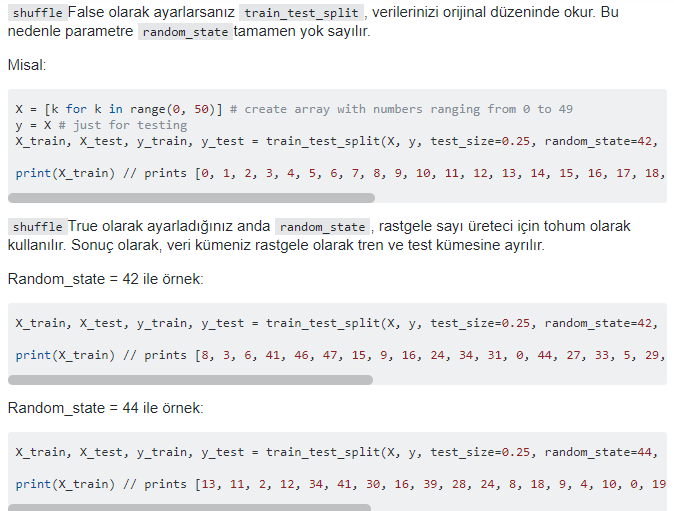
#### AÇIKLAMA

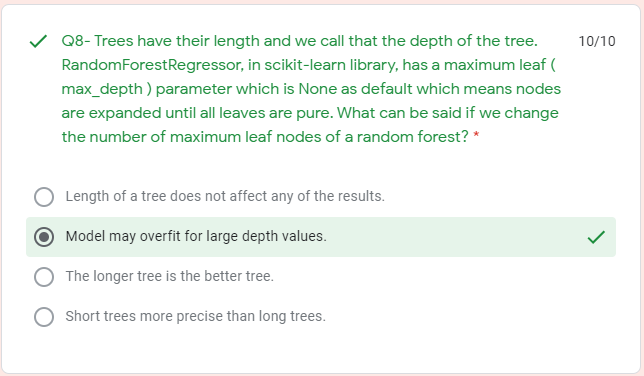
Satır 1 ve satır 2 ayrı ayrı yürütüldüğünde train\_X hakkında ne düşünüyorsunuz? Kodun geri kalanı tamamen aynıdır.

* Farklı rasgele sayı üretirler, böylece train\_X birbirinden farklıdır.
* Farklı aynı sayı üretirler ve train\_X birbirine eşittir.
* Farklı rasgele sayı üretirler, böylece train\_X birbirine eşittir.
* Farklı rasgele sayı üretirler, ancak train\_X birbirine eşittir.





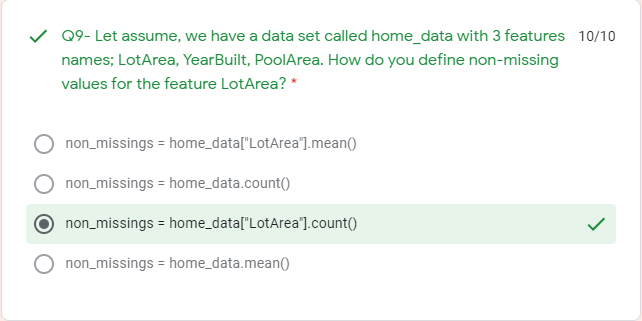




#### AÇIKLAMA

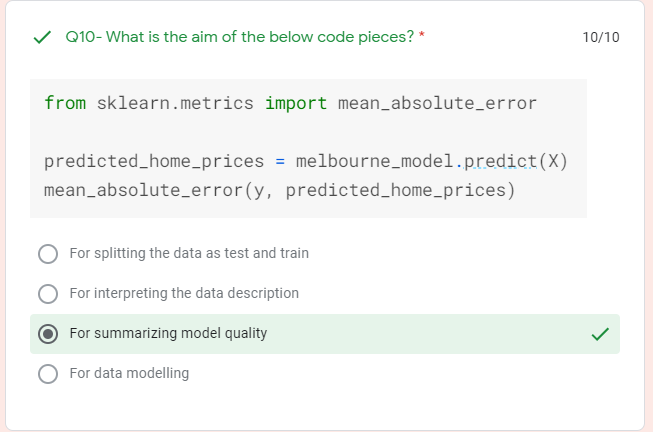
Ağaçların uzunluğu var ve buna ağacın derinliği diyoruz. RandomForestRegressor, scikit-learn kütüphanesinde, varsayılan olarak hiçbiri olmayan bir maksimum yaprak ( max\_depth ) parametresine sahiptir, bu da tüm yapraklar saf olana kadar düğümlerin genişletildiği anlamına gelir. Rastgele bir ormanın maksimum yaprak düğümlerinin sayısını değiştirirsek ne söylenebilir?

* Bir ağacın uzunluğu sonuçların hiçbirini etkilemez.
* Model büyük derinlik değerleri için overfit olabilir.
* Uzun ağaç daha iyi ağaçtır.
* Kısa ağaçlar uzun ağaçlardan daha hassastır.



#### AÇIKLAMA

Varsayalım, home\_data adlı bir veri kümemiz var 3 özellik isimleri; LotArea, YearBuilt, PoolArea. Özellik Lot alanı için eksik olmayan değerleri nasıl tanımlarsınız?



#### AÇIKLAMA

Aşağıdaki kod parçalarının amacı nedir?

* Verileri test ve train olarak bölmek için
* Veri açıklamasının yorumlanması için
* Model kalitesini özetlemek için
* Veri modelleme için

# Intermediate Machine Learning (Orta Düzey Makine Öğrenimi)

Eksik değerleri, sayısal olmayan değerleri, veri sızıntısını ve daha fazlasını ele almayı öğrenin. Modelleriniz daha doğru ve kullanışlı olacaktır.

## Introduction (Giriş)

Bu mikro kurs için neye ihtiyacınız olduğunu inceleyin.

**Kaggle Learn'ın Intermediate Machine Learning mikro kursuna hoş geldiniz!**

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

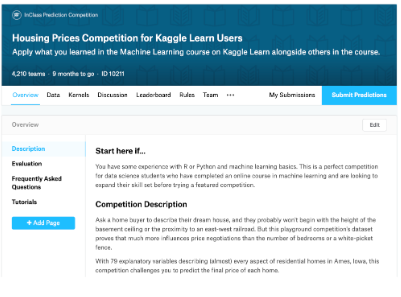
Bu mikro derste, nasıl yapılacağını öğrenerek makine öğrenme uzmanlığınızı hızlandıracaksınız:

* gerçek dünyadaki veri kümelerinde sıklıkla bulunan veri türlerini ele alın (**missing values(eksik değerler)**, **categorical variables(kategorik değişkenler)**),
* makine öğrenme kodunuzun kalitesini artırmak için **design pipelines** tasarlayın,
* model doğrulama için gelişmiş teknikleri kullanın (**cross-validation(çapraz doğrulama**)),
* kaggle yarışmalarını (XGBoost) kazanmak için yaygın olarak kullanılan son teknoloji modelleri oluşturun ve
* yaygın ve önemli veri bilimi hatalarından kaçının (leakage(sızıntı).

Yol boyunca, her yeni konu için gerçek dünya verileri ile bir uygulamalı egzersiz tamamlayarak bilginizi pekiştireceğiz.

Uygulamalı egzersizler, ev fiyatlarını tahmin etmek için 79 farklı açıklayıcı değişken (çatı tipi, yatak odası sayısı ve banyo sayısı gibi) kullanacağınız Kaggle Learn kullanıcıları için konut fiyatları Yarışmasından elde edilen verileri kullanır.

Bu yarışmaya tahminler göndererek ve afiş üzerinde durum yükselişi izleyerek ilerlemeyi ölçmek gerekir!



## Prerequisites (Önkoşullar)

Daha önce bir machine learning model oluşturduysanız, bu mikro kursa hazırsınız ve model validation(doğrulama), underfitting ve overfitting ve random forestes gibi konulara aşinasınız.

Makine öğreniminde tamamen yeniyseniz, lütfen bu ara mikro kurs için hazırlamanız gereken her şeyi kapsayan tanıtım mikro kursumuza göz atın.

## Your Turn(Sıra Sende)

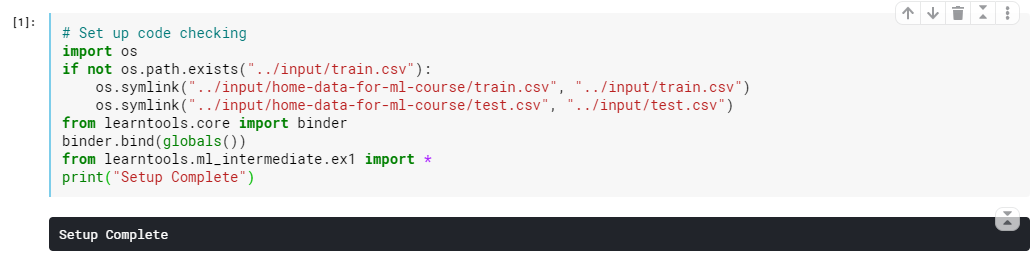
Bir Kaggle yarışmasına tahminlerin nasıl gönderileceğini öğrenmek ve başlamadan önce gözden geçirmeniz gerekenleri belirlemek için ilk alıştırmaya devam edin.

## Exercise: Introduction

As a warm-up, you'll review some machine learning fundamentals and submit your initial results to a Kaggle competition.

### Setup

Aşağıdaki sorular size işinizle ilgili geri bildirim verecektir. Geri bildirim sistemini ayarlamak için aşağıdaki hücreyi çalıştırın.



Kaggle learn kullanıcıları, evlerin her yönünü (hemen hemen) açıklayan 79 açıklayıcı değişken kullanarak Iowa'daki ev fiyatlarını tahmin etmek için konut fiyatları Yarışmasından elde edilen verilerle çalışacaksınız.

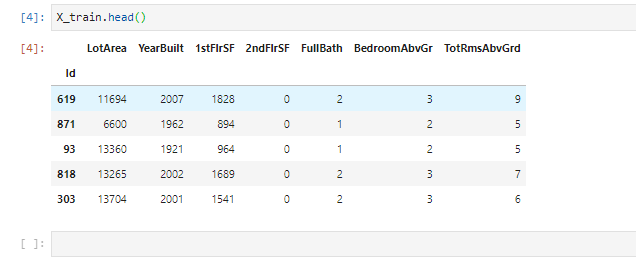


Y\_train ve y\_valid'deki tahmin hedefleriyle(prediction targets) birlikte x\_train ve X\_valid'deki training (eğitim) ve validation(doğrulama) features yüklemek için değişiklik yapmadan bir sonraki kod hücresini çalıştırın.

Test features x\_test yüklenir. (Özellikleri ve tahmin hedeflerini gözden geçirmeniz gerekiyorsa, lütfen bu kısa öğreticiye göz atın . Model doğrulama hakkında okumak için buraya bakın. Alternatif olarak, tüm bu konuları gözden geçirmek için tam bir kursa bakmayı tercih ederseniz, buradan başlayın.)



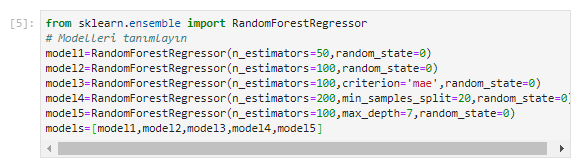
Verilerin ilk birkaç satırını yazdırmak için sonraki hücreyi kullanın. Fiyat tahmin modelinizde kullanacağınız verilere genel bir bakış elde etmenin güzel bir yoludur.



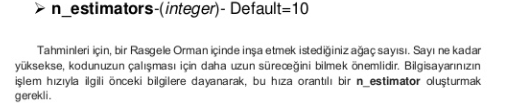
#### **Step 1: birkaç modeli değerlendirin**

Bir sonraki kod hücresi beş farklı random forest modelini tanımlar. Bu kod hücresini değişiklik yapmadan çalıştırın. (\_To inceleme random forests , buraya bak.\_)

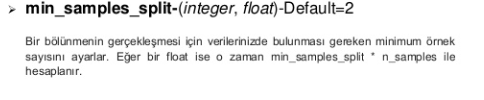
Kod hücresini değişiklik yapmadan çalıştırın.

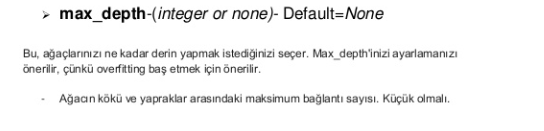


**Buradaki parametrelere göz atalım;**







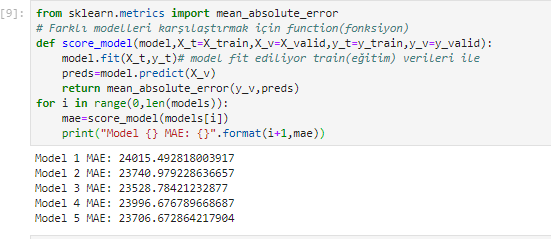


Beşten en iyi modeli seçmek için, aşağıda bir score\_model () işlevi tanımlarız.

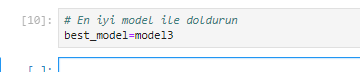
Bu işlev, doğrulama kümesinden ortalama mutlak hata (MAE) döndürür.

En iyi modelin en düşük MAE'Yİ elde edeceğini hatırlayın. (Ortalama mutlak hatayı gözden geçirmek için buraya bakın .) <https://www.kaggle.com/dansbecker/model-validation>

Kod hücresini değişiklik yapmadan çalıştırın.



Aşağıdaki satırı doldurmak için yukarıdaki sonuçları kullanın. Hangi model en iyi model nedir? Cevabınız model1, model2, model3, model4 veya model5 olmalıdır.



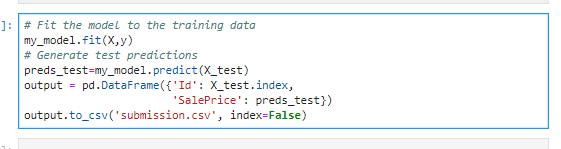
#### **Step 2: Generate test predictions (Test tahminleri oluştur)**

Harika. Doğru bir modeli neyin yaptığını nasıl değerlendireceğinizi biliyorsunuz.

Şimdi modelleme sürecinden geçme ve tahmin yapma zamanı. Aşağıdaki satırda, my\_model değişken adıyla random forest model oluşturun.



Bir sonraki kod hücresini değişiklik yapmadan çalıştırın. Kod, eğitim ve doğrulama verilerine modele uyar ve daha sonra bir CSV dosyasına kaydedilen test tahminleri oluşturur. Bu test tahminleri doğrudan yarışmaya sunulabilir!



## Missing Values (Eksik Değerler)

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

### Introduction (Giriş)

Verilerin eksik değerlerle sonuçlanmasının birçok yolu vardır.

* 2 yatak odalı bir ev, üçüncü bir yatak odasının büyüklüğü için bir değer içermez.
* ankete katılan bir kişi gelirini paylaşmamayı tercih edebilir

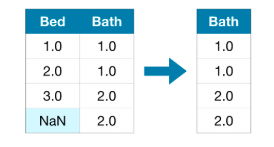
Çoğu makine öğrenme Kütüphanesi (scikit-learn dahil), eksik değerlere sahip verileri kullanarak bir model oluşturmaya çalışırsanız bir hata verir.

Yani aşağıdaki stratejilerden birini seçmeniz gerekir.

### Three Approaches

#### **A Simple Option: Drop Columns with Missing Values (basit bir seçenek: eksik değerlere sahip sütunları drop etmek(düşürmek))**

En basit seçenek, eksik değerlere sahip sütunları drop etmek(düşürmek).



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

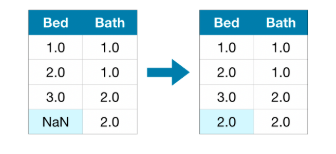
Çok büyük bir örnek olarak, önemli bir sütunun tek bir girdinin eksik olduğu 10.000 satır içeren bir veri kümesini düşünün.

#### Bu yaklaşım sütunu tamamen drop edecektir.

#### **A Better Option: Imputation (Daha iyi bir seçenek)**

İmputation, eksik değerleri bazı sayılarla doldurur.

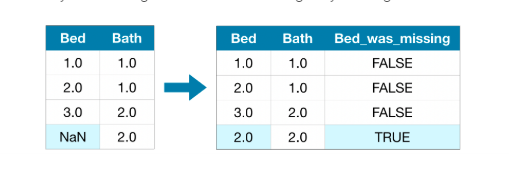
Örneğin, her sütun boyunca ortalama değeri doldurabiliriz.



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

#### **An Extension To Imputation**

İmputation standart yaklaşımdır ve genellikle iyi çalışır. Bununla birlikte, imputed edilen değerler sistematik olarak gerçek değerlerinin (veri kümesinde toplanmamış) üstünde veya altında olabilir.Veya eksik değerlere sahip 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.

### Example**(Örnek)**

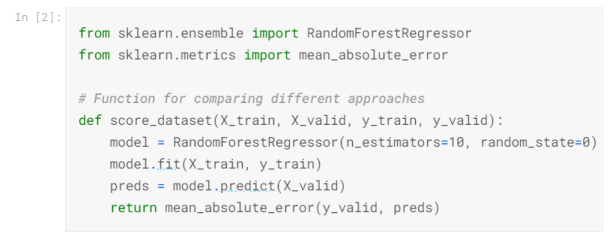
Örnekte,  [Melbourne Housing dataset](https://www.kaggle.com/dansbecker/melbourne-housing-snapshot/home) ile çalışacağız.

Modelimiz, ev fiyatını tahmin etmek için oda sayısı ve arazi büyüklüğü gibi bilgileri kullanacaktır.

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

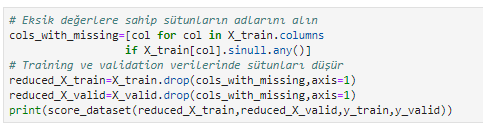
#### **Define Function to Measure Quality of Each Approach((Her yaklaşımın kalitesini ölçme yaklaşımı)**

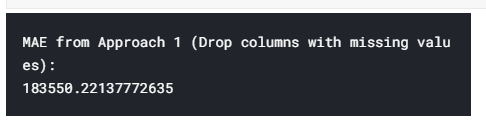
Eksik değerlerle başa çıkmak için farklı yaklaşımları karşılaştırmak için bir işlev score\_dataset() tanımlarız. Bu işlev, bir random forest modelinden ortalama mutlak hata (MAE) bildirir.



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

Hem training hem de validation setleri ile çalıştığımızdan, her iki veri karesinde de aynı sütunları drop etmeye dikkat ediyoruz.



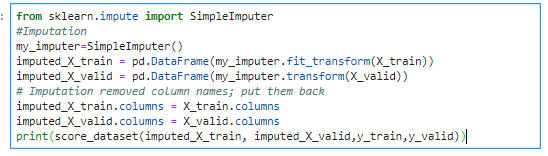


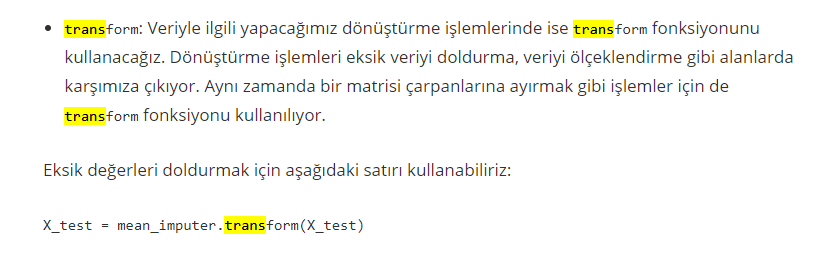
#### **Score from Approach 2 (Imputation)**

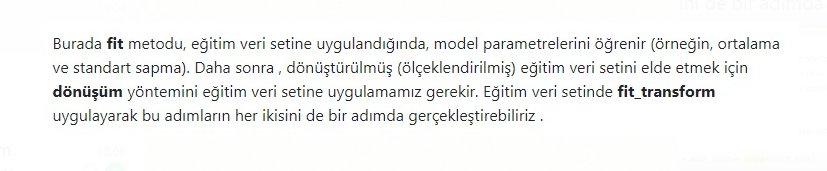
Ardından, eksik değerleri her sütun boyunca ortalama değerle değiştirmek için **SimpleImputer** kullanıyoruz.

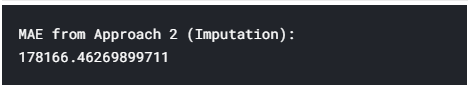
Basit olmasına rağmen, ortalama değeri doldurmak genellikle oldukça iyi performans gösterir (ancak bu veri kümesine göre değişir).

İstatistikçiler, imputed değerleri belirlemek için daha karmaşık yollar denemiş olsa da (örneğin regresyon varsayımı gibi), karmaşık stratejiler genellikle sonuçları sofistike makine öğrenme modellerine taktığınızda ek bir fayda sağlamaz





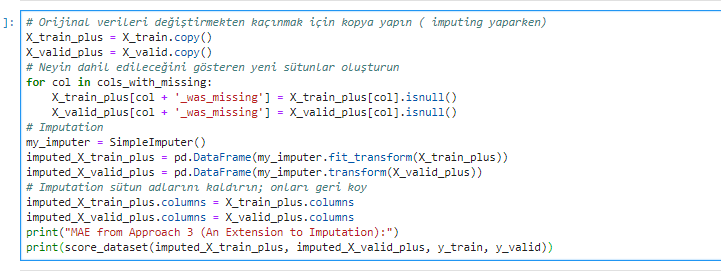


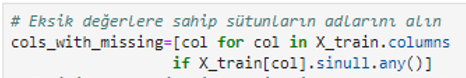


Yaklaşım 2'nin yaklaşım 1'den daha düşük MAE'YE sahip olduğunu görüyoruz, bu nedenle yaklaşım 2 Bu veri kümesinde daha iyi performans gösterdi.

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

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







**Not:** Imputation yöntemi eksik(Nan) olan veri setindeki yerleri, o niteliğin ortalama değerini yazıyor. Yani önceden boş olan yerler o sütunun ortlama değerine sahip.

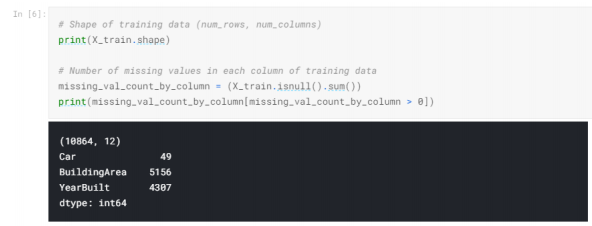
Extension to Imputation yöntemi ise yine buraya kadar olan işlemleri aynı şekilde yapıyoruz. Buna ek olarak o ortalama değeri yazdığım sütunun eleman numarasına da erişiyoruz; Bunu lojik operatörler yardımıyla yeni bir sütun ekleyip, False olanlar zaten veri olan yerler ve True olan yer ise benim ortalama değerini yazdığım yerler. Bu şekilde o sütunda kaçıncı satırdaki elemanlara ortalama değerini atadığını görebiliriz.

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.



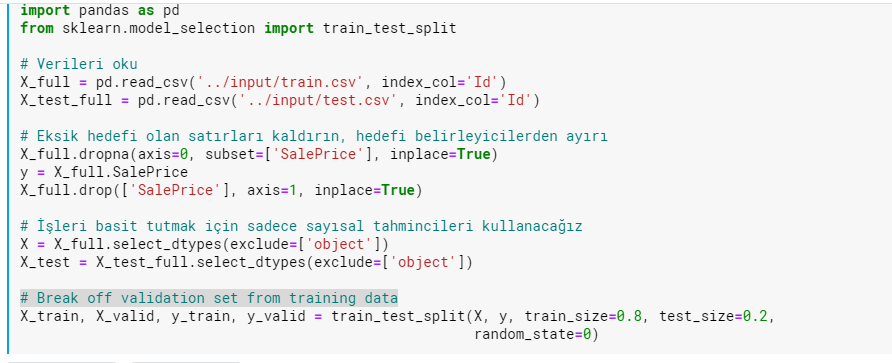
#### **Conclusion**

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.

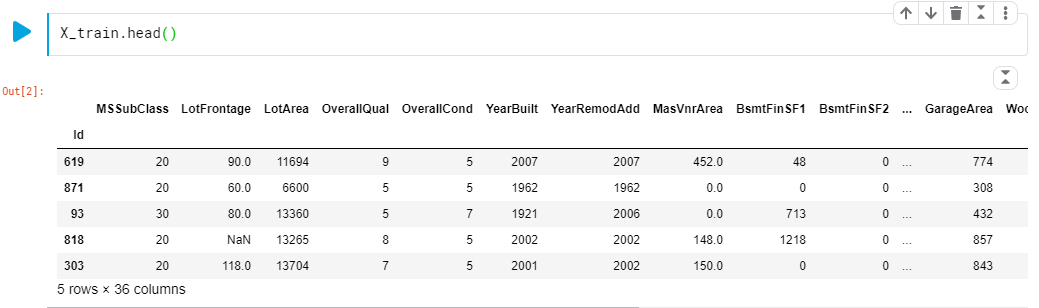


**Notlar:**

**Subset=** Alt küme parametresi, dropna'nın eksik değerleri arayacağı sütunların alt kümesini belirtmenizi sağlar. Belki de 10 değişkenli bir veri kümeniz var, ancak sadece dropna'nın 2'ye bakmasını istiyorsunuz. Dropna'yı bu iki sütunla sınırlamak için alt küme parametresini kullanabilirsiniz.

**Exclude =** Hariç tutmak anlamında kullanılır, object hariç float ve int verilerini alır.

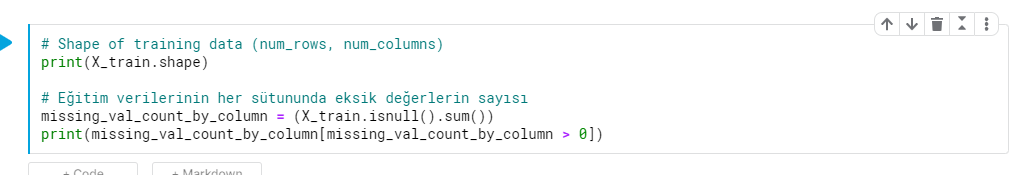
Verilerin ilk beş satırını yazdırmak için bir sonraki kod hücresini kullanın.



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

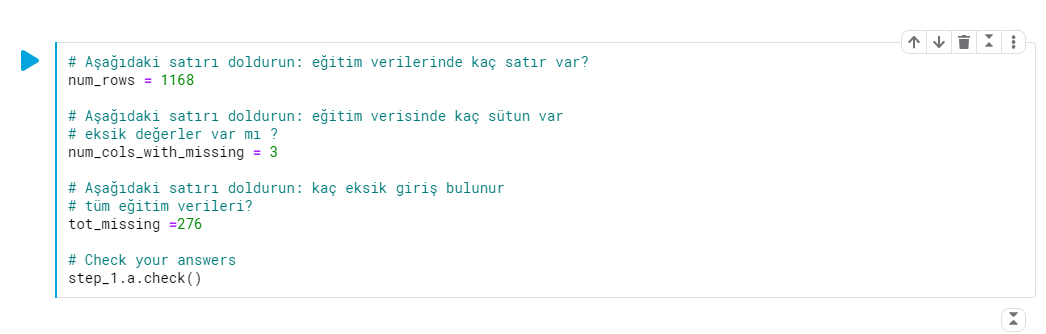
#### **Step 1: Preliminary investigation (Ön Soruşturma)**

Aşağıdaki kod hücresini herhangi bir değişiklik yapmadan çalıştırın.





##### Part A



##### Part B

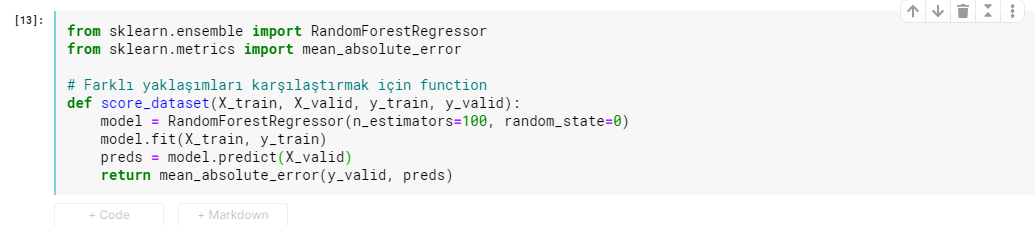
Yukarıdaki cevaplarınızı göz önünde bulundurarak, eksik değerlerle başa çıkmanın en iyi yaklaşımı sizce nedir?

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?

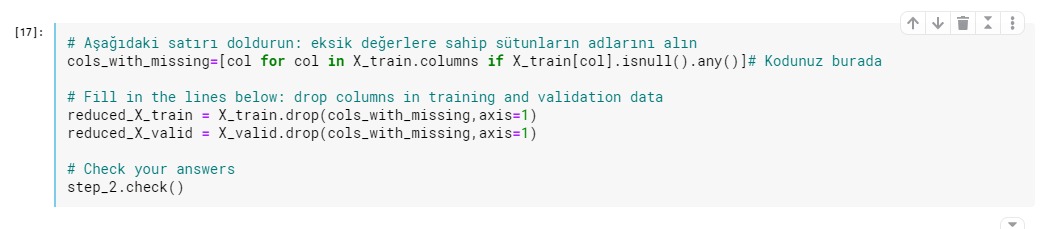
Verilerde nispeten az eksik giriş olduğundan (eksik değerlerin en büyük yüzdesine sahip sütun girişlerinin% 20'sinden daha az eksiktir), sütunları bırakmanın iyi sonuçlar vermesi beklenmez.

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.

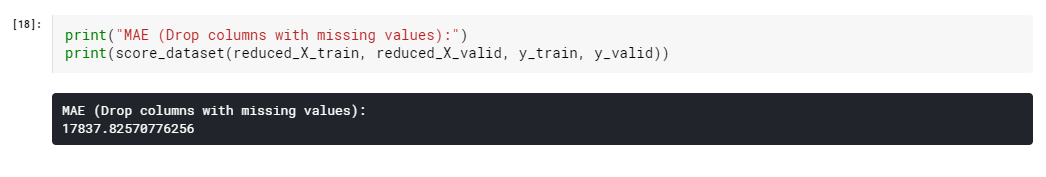


#### **Step 2: Drop columns with missing values**

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.



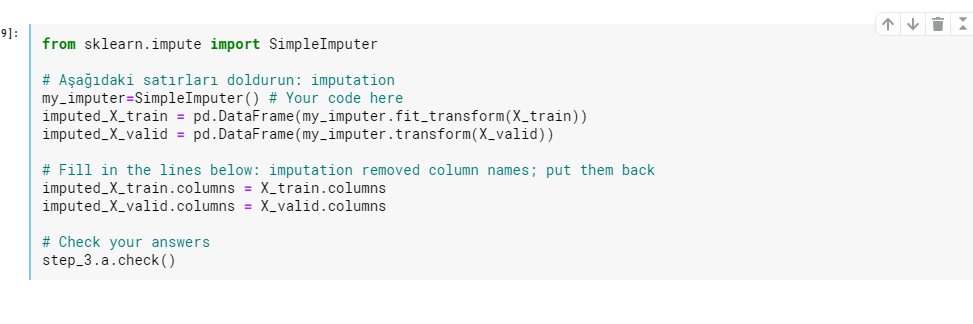
Bu yaklaşım için MAE elde etmek için değişiklik yapmadan bir sonraki kod hücresini çalıştırın.



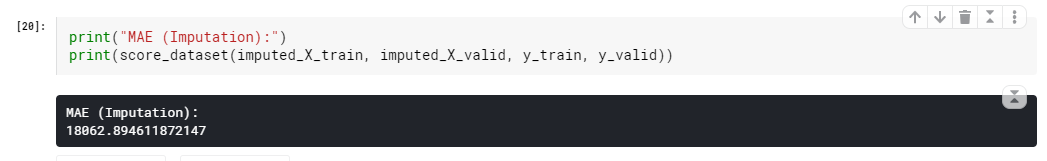
#### **Step 3: Imputation**

##### 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 yapmadan bir sonraki kod hücresini çalıştırın.



##### Part B

Her yaklaşımdan MAE'Yİ karşılaştırın. Sonuçlar hakkında sizi şaşırtan bir şey var mı? Neden bir yaklaşımın diğerinden daha iyi performans gösterdiğini düşünüyorsunuz?

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

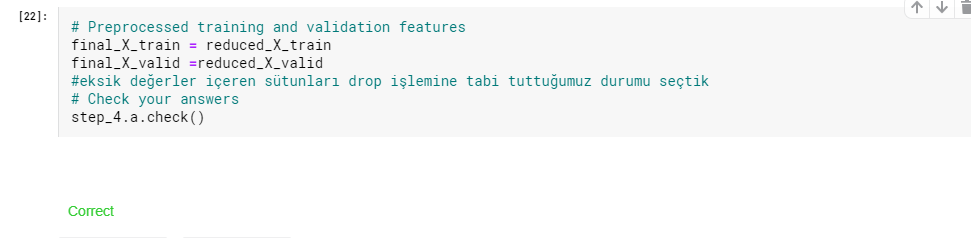
Training ve validation verilerini önceden işlemek için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames'i final\_X\_train ve final\_X\_valid olarak ayarlayın. Burada seçtiğiniz herhangi bir yaklaşımı kullanabilirsiniz! bu adımın doğru olarak işaretlenmesi için yalnızca şunlardan emin olmanız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

• önceden işlenmiş DataFrame'ler aynı sayıda sütuna sahiptir,

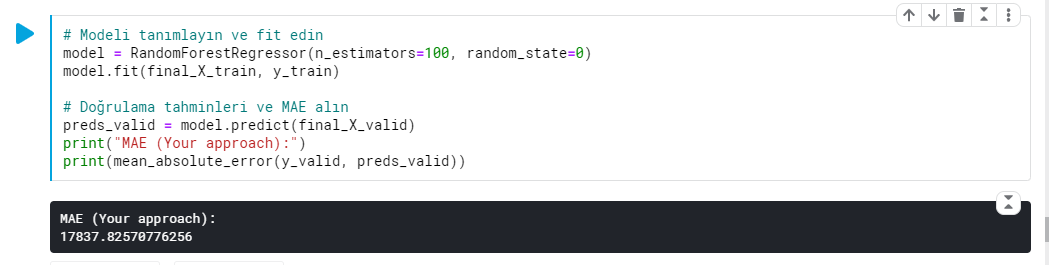
• önceden işlenmiş DataFrame’lerde eksik değer yoktur,

• final\_X\_train ve y\_train aynı sayıda satıra sahip olmalıdır,

• final\_X\_valid ve y\_valid aynı sayıda satıra sahip olmalıdır.



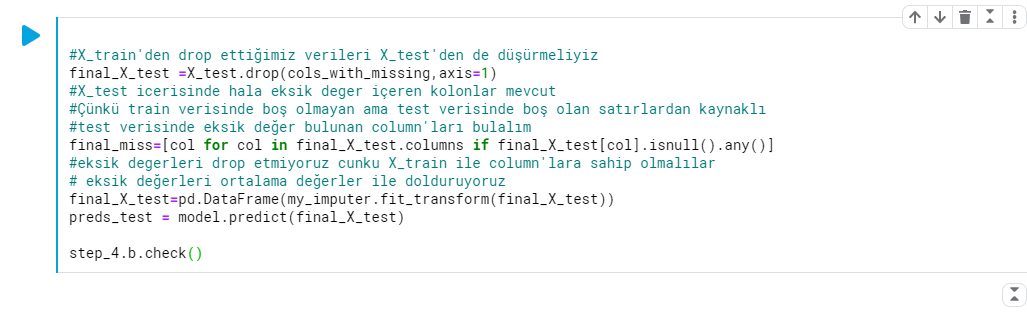
Random Forest modelini eğitmek ve değerlendirmek için bir sonraki kod hücresini çalıştırın. (Yukarıdaki score\_dataset () işlevini kullanmadığımızı unutmayın, çünkü yakında test tahminleri oluşturmak için eğitimli modeli kullanacağız!)



##### Part B

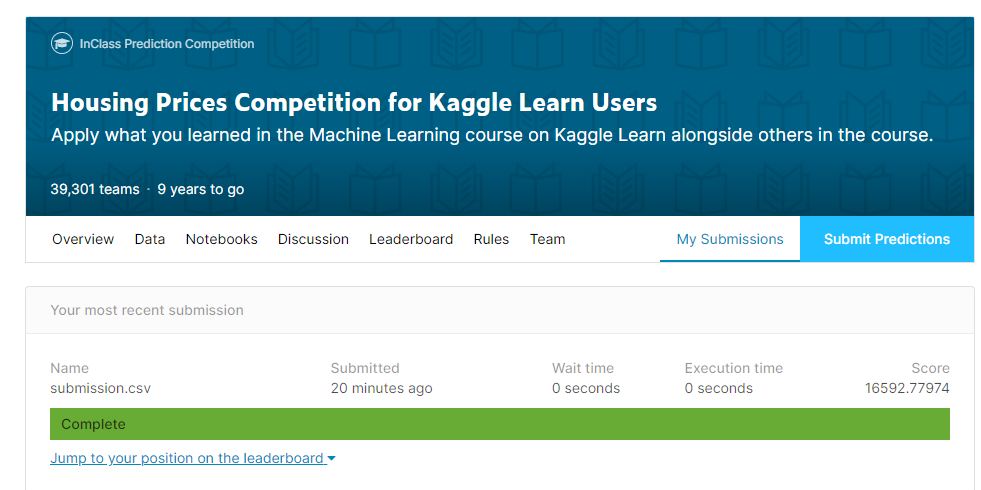
Test verilerinizi önceden işlemek için bir sonraki kod hücresini kullanın. Eğitim ve doğrulama verilerini nasıl önceden işleme koyduğunuzu kabul eden bir yöntem kullandığınızdan emin olun ve önceden işlenmiş test feature’larını `final\_X\_test` olarak ayarlayın. Ardından, `preds\_test 'içinde test tahminleri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

* önceden işlenmiş test veri çerçevesinin eksik değerleri yoktur ve
* final\_X\_test, x\_test ile aynı sayıda satıra sahiptir.



Sonuçlarınızı doğrudan yarışmaya gönderilebilecek bir CSV dosyasına kaydetmek için bir sonraki kod hücresini değişiklik yapmadan çalıştırın.





## 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.

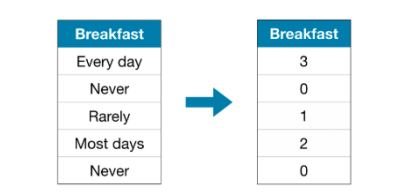
### Üç Yaklaşım

#### **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.

#### **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.

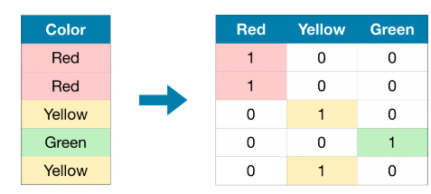
Tüm kategorik değişkenlerin değerlerde açık bir sırası yoktur, ancak **ordinal**(sıralı) değişkenler olarak adlandırılanlara atıfta bulunuruz.

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

#### **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

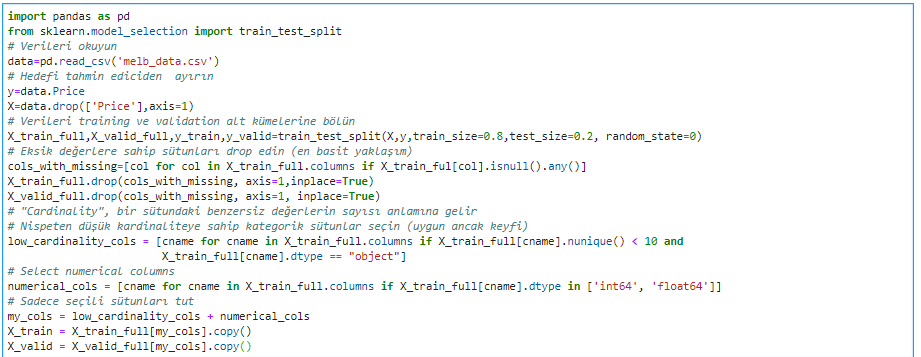
İçsel sıralaması olmayan kategorik değişkenleri **nominal** değişkenler olarak adlandırırız.

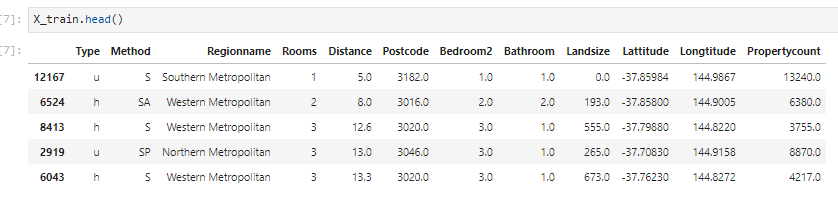
One-hot encoding, kategorik değişken çok sayıda değer alıyorsa genellikle iyi performans göstermez (yani, genellikle 15'ten fazla farklı değer alan değişkenler için kullanmazsınız).

### Example

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

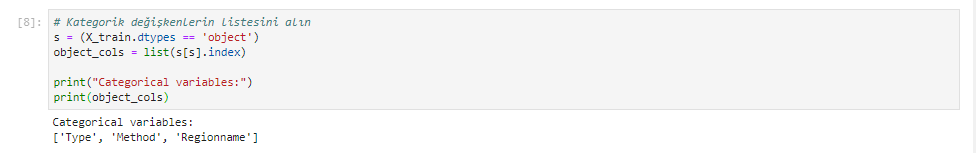
Veri yükleme adımına odaklanmayacağız. Bunun yerine, zaten X\_train, X\_valid, y\_train ve y\_valid'de eğitim ve doğrulama verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.





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**

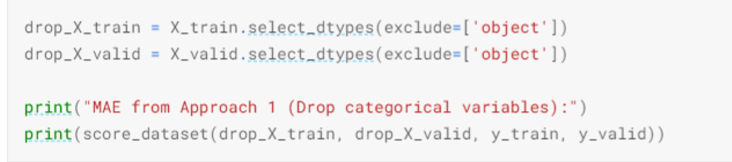
Kategorik değişkenlerle başa çıkmak için üç farklı yaklaşımı karşılaştırmak için score\_dataset () fonksiyonunu tanımlarız.

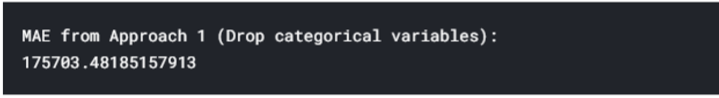
Bu işlev bir Random Forest modelinden gelen ortalama mutlak hatayı (MAE) döndürür. Genel olarak MAE'nin mümkün olduğunca düşük olmasını istiyoruz!



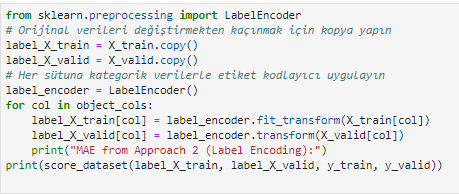
#### **Score from Approach 1 (Drop Categorical Variables)**

Object sütunlarını select\_dtypes () yöntemiyle düşürürüz.





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





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

İgnore= yoksaymak anlamında kullanılır.

• **sparse = False**, kodlanmış sütunların sayısal bir dizi olarak döndürülmesini sağlar (seyrek bir matris yerine).

Seyrek matrisler sadece daha verimlidir, büyük veri kümelerini saklamanın bir yoludur özellikle çok fazla sıfır içeriyorsa yararlı olan bir işlevdir.

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!

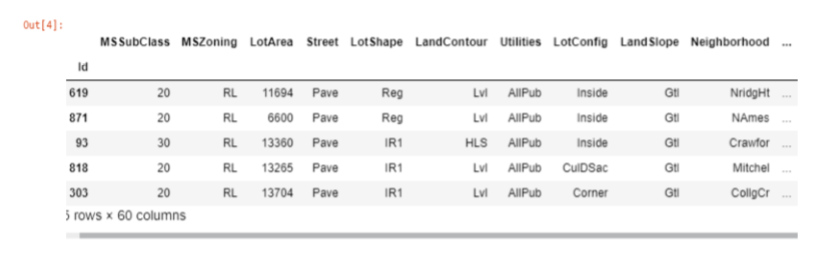
Bu alıştırmada  [Housing Prices Competition for Kaggle Learn Users](https://www.kaggle.com/c/home-data-for-ml-course) ile çalışacağız.

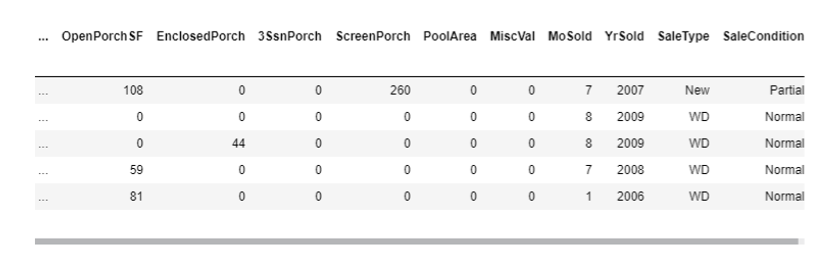


X\_train, X\_valid, y\_train ve y\_valid'e training ve validation setlerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X\_test'e yüklenir.









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 tutorial’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.

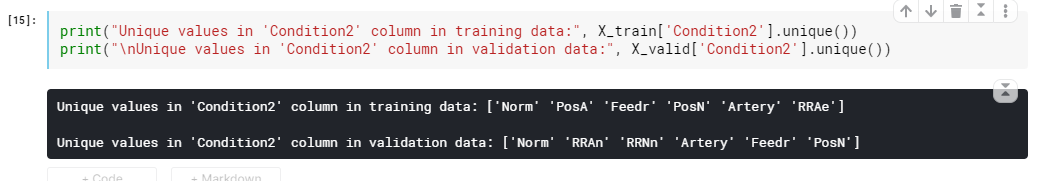


Bu yaklaşım için MAE hesaplayalım.



#### **Step 2: Label Encoding**

Label Encoding’e geçmeden önce veri kümesini araştıracağız. Özellikle, “Condition2” 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.\_)

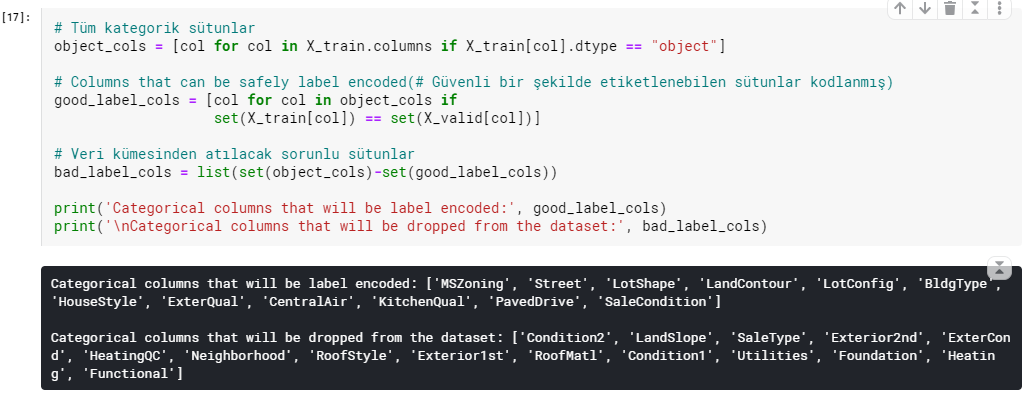
Validation verilerinde görünen ancak training verilerinde olmayan değerler var mı?

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 değerlerde kendilerine atanan bir tamsayı olmaz.

Validation 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 verilerde 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.

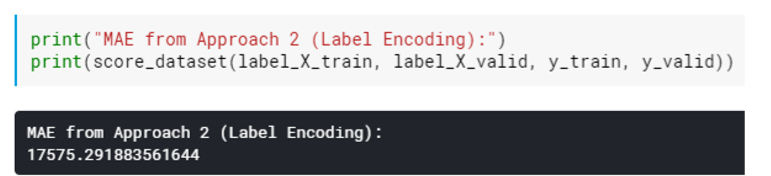


X\_train ve X\_valid içindeki verilere label encode yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla label\_X\_train ve label\_X\_valid olarak ayarlayın.

• Kategorik sütunları veri kümesinden bad\_label\_cols içine çekmek için aşağıdaki kodu sağladık.

• Kategorik sütunlar içinden good\_label\_cols’lara label encode uygulamanız gerekir.

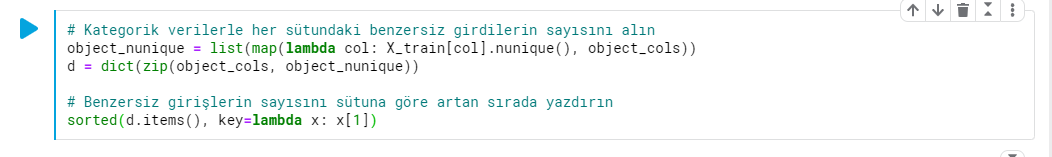




#### **Step 3: Investigating Cardinality (Kardinalite Araştırması)**

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.





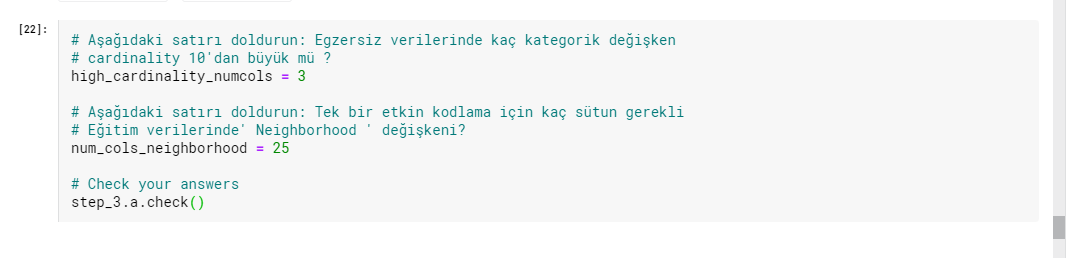
**Zip():**



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.



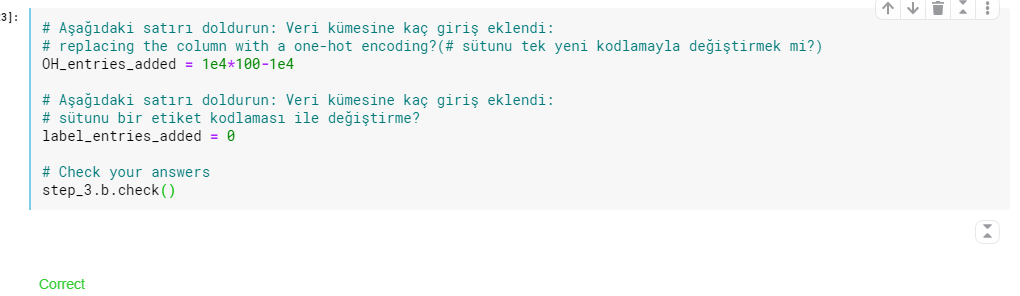
Birçok satıra sahip büyük veri kümeleri için, one-hot encoding, veri kümesinin boyutunu büyük ölçüde genişletebilir. Bu nedenle, yalnızca tipik olarak nispeten düşük kardinaliteye sahip sütunlara one-hot encoding uygulayacağız. Daha sonra, yüksek kardinalite sütunları veri kümesinden kaldırılabilir veya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

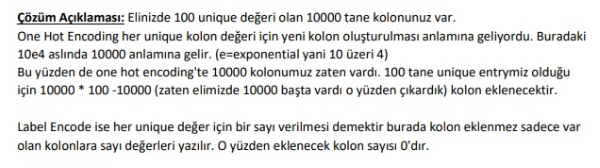
Örnek olarak, 10.000 satır içeren ve 100 benzersiz giriş içeren bir kategorik sütun içeren bir veri kümesini düşünün.

• 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.



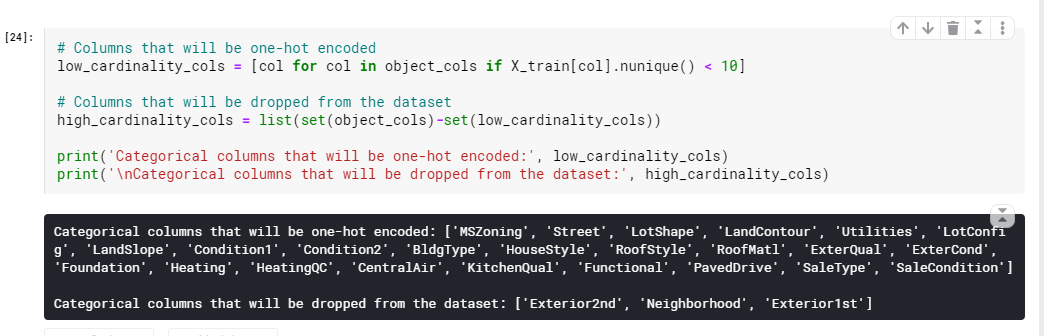


one-hot encoding yoluyla veri kümesine kaç girdi eklendiğini hesaplamak için, kategorik değişkeni kodlamak için kaç girdinin gerekli olduğunu hesaplayarak başlayın (satır sayısını one-hot encoding’deki sütun sayısıyla çarparak). Ardından, veri kümesine kaç girdi eklendiğini öğrenmek için, orijinal sütundaki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

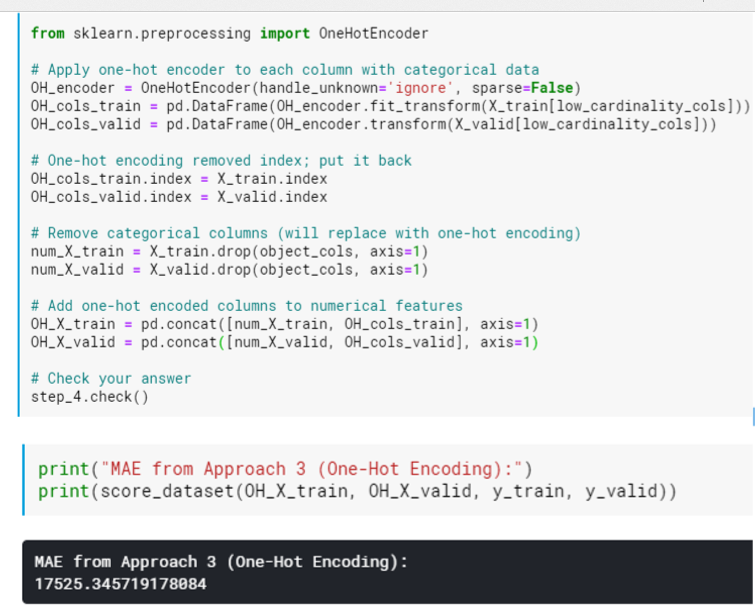


X\_train ve X\_valid içindeki verilere one-hot encoding yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla OH\_X\_train ve OH\_X\_valid olarak ayarlayın.

• 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.



**Not**

Concat fonksiyonu iki seriyi birleştirmek için kullanılır.

#### **Step 5: Generate test predictions and submit your results**

4. Adım'ı tamamladıktan sonra, sonuçlarınızı skor tablosuna göndermek için öğrendiklerinizi kullanmak isterseniz, tahminler oluşturmadan önce test verilerini önceden işlemeniz gerekir.

## Pipelines

Bu öğreticide, modelleme kodunuzu temizlemek için **pipeline**’ı nasıl kullanacağınızı öğreneceksiniz.

### Introduction

Pipeline’lar, veri önişleme ve modelleme kodunuzu düzenli tutmanın basit bir yoludur. Özellikle, bir ardışık düzen ön işleme ve modelleme adımlarını bir araya getirir, böylece tüm paketi tek bir adımmış gibi kullanabilirsiniz.

Birçok veri bilimcisi modelleri pipeline kullanmadan bir araya getirmektedir, ancak pipeline’nın bazı önemli faydaları vardır. Bunlar arasında:

* **Temiz Kod:** Ön işlemenin her adımındaki verilerin muhasebeleştirilmesi dağınık olabilir. Bir ardışık düzen ile, her adımda egzersiz ve doğrulama verilerinizi manuel olarak takip etmeniz gerekmez.
* **Daha Az Hata:** Bir adımı yanlış uygulama veya bir önişleme adımını unutmak için daha az fırsat vardır.
* **Üretim için Kolaylık:** Bir modeli bir prototipten ölçekte konuşlandırılabilir bir şeye geçirmek şaşırtıcı derecede zor olabilir. Burada birçok ilgili kaygıya girmeyeceğiz, ancak pipeline yardımcı olabilir.
* **Model Validation için Daha Fazla Seçenek:** Bir sonraki öğreticide cross validation’u kapsayan bir örnek göreceksiniz.

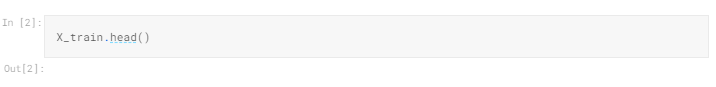
### Example

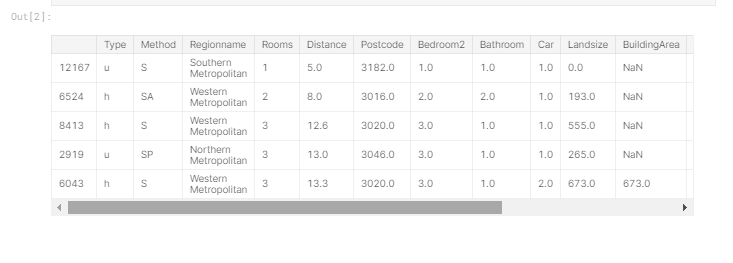
Önceki derste olduğu gibi,  [Melbourne Housing dataset](https://www.kaggle.com/dansbecker/melbourne-housing-snapshot/home) ile çalışacağız.

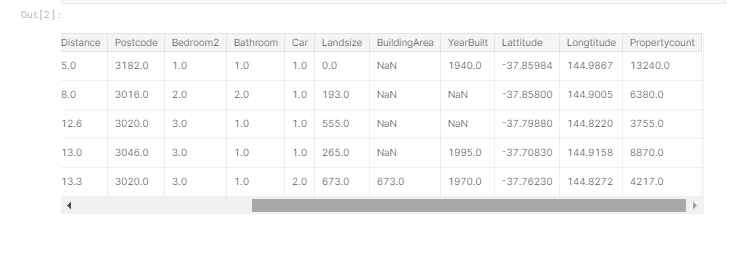
Veri yükleme adımına odaklanmayacağız. Bunun yerine, X\_train, X\_valid, y\_train ve y\_valid'de zaten eğitim ve doğrulama verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.



Aşağıdaki head () yöntemiyle eğitim verilerine bir göz atın. Verilerin hem kategorik veriler hem de eksik değerleri olan sütunlar içerdiğine dikkat edin. Bir pipeline ile her ikisiyle de başa çıkmak kolay!







**Pipeline’nın tamamını üç adımda inşa ediyoruz.**

#### **Step 1: Önişleme Adımlarını Tanımlayın**

Bir pipeline’nın ön işleme ve modelleme adımlarını nasıl bir araya getirdiğine benzer şekilde, farklı önişleme adımlarını bir araya getirmek için ColumnTransformer sınıfını kullanırız.

Aşağıdaki kod:

• sayısal verilerdeki eksik değerleri ifade eder ve

• eksik değerleri ifade eder ve kategorik verilere one-hot encoding uygular.



Constant: “constant” , eksik değerleri fill\_value ile değiştirin. Dizeler veya sayısal verilerle kullanılabilir.



#### **Step 2: Modeli tanımlayın**

Ardından, tanıdık RandomForestRegressor sınıfıyla bir Random Forest modeli tanımlarız.



#### **Step 3: Pipeline Oluşturun ve Değerlendirin**

Son olarak, ön işleme ve modelleme adımlarını bir araya getiren bir pipeline tanımlamak için Pipeline sınıfını kullanırız. Dikkat edilmesi gereken birkaç önemli nokta vardır:

* Pipeline ile, eğitim verilerini önceden işler ve modeli tek bir kod satırına sığdırırız. (Aksine, bir pipeline olmadan, ayrı adımlarla imputing, one-hot encoding ve model eğitimi yapmak zorundayız. Hem sayısal hem de kategorik değişkenlerle uğraşmak zorunda kalırsak bu özellikle dağınık hale gelir!)
* Pipeline ile, işlenmemiş özellikleri X\_valid'te predict () komutuna sağlarız ve boru hattı, tahminler oluşturmadan önce özellikleri otomatik olarak ön işleme tabi tutar. (Ancak, bir ardışık düzen olmadan, tahminlerde bulunmadan önce doğrulama verilerini önceden işlemeyi hatırlamamız gerekir.)



### Sonuç

Pipeline’lar, makine öğrenmesi kodunu temizlemek ve hatalardan kaçınmak için değerlidir ve özellikle sofistike veri önişlemeli iş akışları için yararlıdır.

### Exercise: Pipelines

Bu alıştırmada, makine öğrenme kodunuzun verimliliğini artırmak için pipeline kullanacaksınız.

Çalışmamızda [Housing Prices Competition for Kaggle Learn Users](https://www.kaggle.com/c/home-data-for-ml-course) datasetini kullanacağız.



X\_train, X\_valid, y\_train ve y\_valid'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X\_test'e yüklenir.







Bir sonraki kod hücresi, verileri önceden işlemek ve bir modeli eğitmek için tutorial’ın kodunu kullanır. Bu kodu değişiklik yapmadan çalıştırın.



Kod, ortalama mutlak hata (MAE) için 17862 civarında bir değer verir. Bir sonraki adımda, daha iyisini yapmak için kodu değiştireceksiniz.

#### **Step 1: Performansı Arttırın**

##### **Part A**

Şimdi senin sıran! Aşağıdaki kod hücresinde, kendi önişleme adımlarınızı ve Random Forest modelinizi tanımlayın. Aşağıdaki değişkenler için değerleri girin:

• numerical\_transformer

• categorical\_transformer

• model

Egzersizin bu kısmını geçmek için, sadece geçerli önişleme adımlarını ve Random Forest modelini tanımlamanız gerekir.



**İpucu:** Bu soruna birçok farklı potansiyel çözüm olsa da, yalnızca column\_transformer'ı varsayılan değerden değiştirerek tatmin edici sonuçlar elde ettik - özellikle, eksik değerlerin nasıl uygulanacağına karar veren strategy parametresini değiştirdik.

##### **Part B**

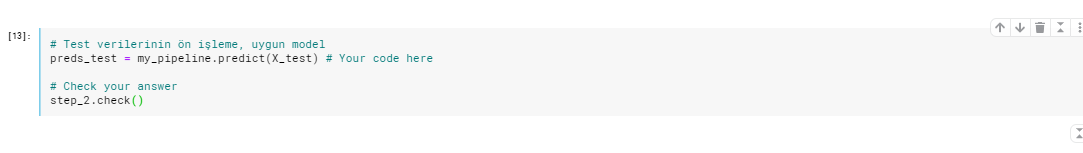
Bu adımı geçmek için, Part A'da, yukarıdaki koddan daha düşük MAE elde eden bir pipeline tanımlamanız gerekir. Burada zaman ayırıp MAE'yi ne kadar düşük alabileceğinizi görmek için birçok farklı yaklaşımı denemeniz önerilir! (Kodunuz geçmezse, lütfen ön işleme adımlarını ve modelini Part A'da değiştirin.)



**İpucu:** Daha iyi performans elde etmek için önişleme adımlarının ve modelinin nasıl değiştirileceği hakkında bazı fikirler almak için lütfen Part A'nın ipucuna bakın.

#### **Step 2: Test Tahminleri Oluşturun**

Şimdi, test verileriyle tahminler oluşturmak için eğitimli modelinizi kullanacaksınız.





## Cross-Validation

Bu tutorial’da, daha iyi model performansı ölçümleri için **cross-validation’un** nasıl kullanılacağını öğreneceksiniz.

### Introduction

Makine öğrenmesi **yinelemeli(iterative)** bir süreçtir.

Hangi öngörücü değişkenlerin kullanılacağı, hangi tür modellerin kullanılacağı, bu modellere hangi argümanların sağlanacağı vb. ile ilgili seçeneklerle karşılaşacaksınız.

Şimdiye kadar, bir validation (veya holdout(kısıtlama)) seti ile model kalitesini ölçerek bu seçimleri veriye dayalı bir şekilde yaptınız.

Ancak bu yaklaşımın bazı dezavantajları vardır.

Bunu görmek için 5000 sıralı bir veri kümeniz olduğunu hayal edin.

Tipik olarak verilerin yaklaşık % 20'sini veya 1000 satırını validation veri kümesi olarak tutacaktır.

Ancak bu, model puanlarının belirlenmesini rastgele bir şekilde şansa bırakır.

Yani, bir model farklı bir 1000 satırda yanlış olsa bile başka 1000 satırlık bir sette iyi olabilir.

Uç bir nokta olarak, validation kümesinde yalnızca 1 veri satırı olduğunu hayal edebilirsiniz. Alternatif modelleri karşılaştırırsanız, tek bir veri noktasında en iyi tahminleri yapan, çoğunlukla şans meselesi olacaktır!

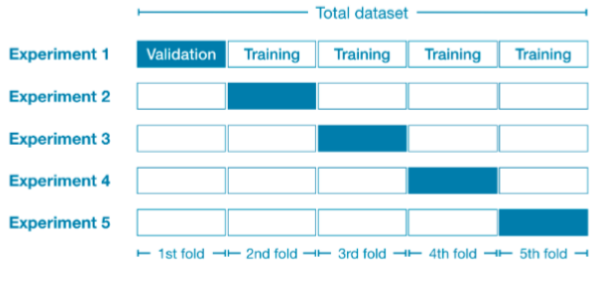
Genel olarak, validation seti ne kadar büyük olursa, model kalitesi ölçümüzde o kadar az rastgelelik ("gürültü") olur ve o kadar güvenilir olur.

Ne yazık ki, yalnızca training verilerimizdeki satırları kaldırarak büyük bir validation kümesi alabiliriz ve daha küçük training veri setleri daha kötü modeller anlamına gelir!

### Cross-Validation Nedir?

**Cross-Validation’da**, model kalitesinin birden fazla ölçüsünü almak için modelleme sürecimizi verilerin farklı alt kümelerinde çalıştırıyoruz.

Örneğin, verileri her biri tam veri kümesinin % 20'si olan 5 parçaya bölerek başlayabiliriz. Bu durumda, verileri 5 "fold"’a ayırdığımızı söylüyoruz.



Ardından, her fold için bir deneme gerçekleştiriyoruz:

* Deney 1'de, ilk foldu bir validation (veya holdout) kümesi ve diğer hepsini training verileri olarak kullanıyoruz. Bu bize % 20'lik bir holdout(dağıtım) setine dayanan bir model kalitesi ölçüsü verir.
* Deney 2'de, ikinci fold’daki verileri tutarız (ve ikinci fold dışındaki her şeyi modeli eğitmek için kullanırız). Daha sonra holdout(dağıtım) seti, model kalitesinin ikinci bir tahminini almak için kullanılır.
* Her fold’u holdout(dağıtım) seti olarak bir kez kullanarak bu işlemi tekrarlıyoruz. Bunları bir araya getirerek, verilerin % 100'ü bir noktada holdout olarak kullanılır ve veri kümesindeki tüm satırlara dayanan bir model kalitesi ölçüsü elde ederiz (tüm satırları aynı anda kullanmasak bile) .

### Ne Zaman Cross-Validation Kullanmalıyız?

Cross-Validation, model kalitesinin daha doğru bir ölçümünü verir, bu da çok fazla modelleme kararı verirseniz özellikle önemlidir.

Bununla birlikte, birden fazla modeli tahmin ettiğinden (her fold için bir tane) tahmin edilmesi daha uzun sürebilir.

Peki, bu ödünleşmeler göz önüne alındığında, her bir yaklaşımı ne zaman kullanmalısınız

* Fazladan hesaplama yükünün çok önemli olmadığı küçük veri kümeleri için cross-validation yapmalısınız.
* Daha büyük veri kümeleri için tek bir validation kümesi yeterlidir. Kodunuz daha hızlı çalışacaktır.

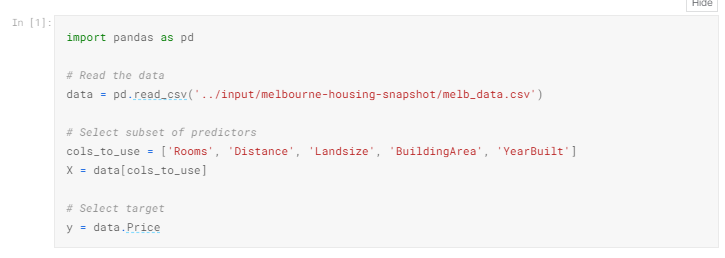
Büyük ve küçük veri kümesini oluşturan şey için basit bir eşik yoktur. Ancak modelinizin çalışması birkaç dakika veya daha az sürüyorsa, muhtemelen cross-validation’a geçmeye değer.

Alternatif olarak, cross-validation’ı çalıştırabilir ve her deney için puanların yakın olup olmadığını gözlemleyebilirsiniz.

Her deney aynı sonuçları verirse, tek bir validation seti muhtemelen yeterlidir.

### Example

Önceki derslerdeki verilerle çalışacağız. Input verilerini X'e, Output verilerini y'ye yüklüyoruz.



Ardından, eksik değerleri doldurmak için bir imputer ve tahminler yapmak için bir Random Forest modeli kullanan Pipeline tanımlarız.

Pipeline olmadan cross-validation yapmak mümkün olsa da, oldukça zor! Bir pipeline kullanmak, kodu oldukça basit hale getirecektir.



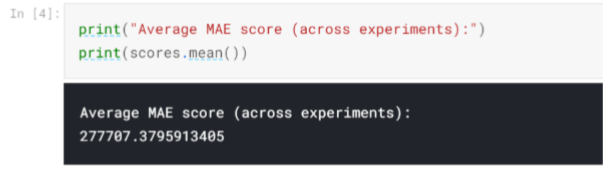
Scikit-learn'dan **cross\_val\_score()** işleviyle **cross-validation** skorlarını elde ederiz. **Fold** sayısını cv parametresi ile ayarladık.



**scoring** parametresi, raporlama için bir model kalitesi ölçüsü seçer: bu durumda negatif ortalama mutlak hata (MAE) seçtik. (<https://scikit-learn.org/stable/modules/model_evaluation.html>)

Negatif MAE'yi belirtmemiz biraz şaşırtıcı. Scikit-learn, tüm metriklerin tanımlandığı bir kurala sahiptir, bu nedenle yüksek bir sayı daha iyidir. Negatif MAE neredeyse başka bir yerde duyulmamış olsa da, negatifleri burada kullanmak bu kuralla tutarlı olmalarını sağlar.

Alternatif modelleri karşılaştırmak için genellikle tek bir model kalitesi ölçüsü istiyoruz. Bu yüzden deneyler boyunca ortalamayı alıyoruz.



### Sonuç

Cross-validation kullanılması, kodumuzu temizlemenin sağladığı ek avantajla birlikte model kalitesinin çok daha iyi bir ölçüsünü verir: artık ayrı eğitim ve doğrulama setlerini takip etmemize gerek olmadığını unutmayın. Bu nedenle, özellikle küçük veri kümeleri için bu iyi bir gelişme!

### Exercise: Cross-Validation

Bu alıştırmada, bir makine öğrenme modelini cross-validation ile ayarlamak için öğrendiklerinizden yararlanacaksınız.

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

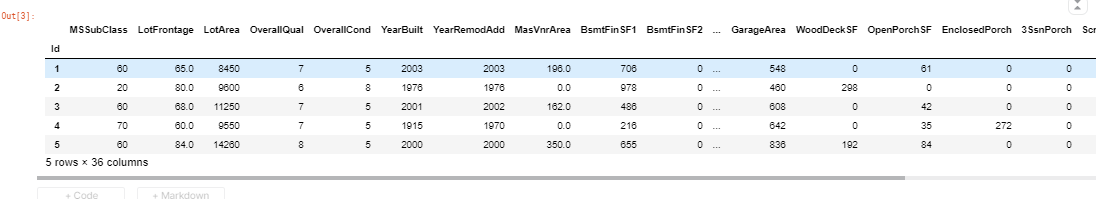


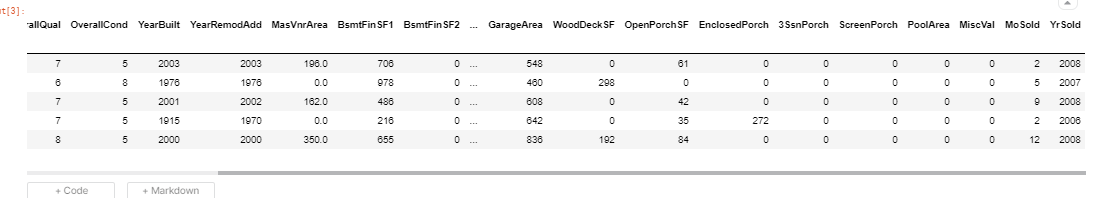
\_train, X\_valid, y\_train ve y\_valid'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X\_test'e yüklenir.

Basit olması için kategorik değişkenleri düşürüyoruz.









Şimdiye kadar, scikit-learn ile pipeline’ların nasıl kurulacağını öğrendiniz.

Örneğin, aşağıdaki pipeline, tahminler yapmak üzere bir Random Forest modeli eğitmek için RandomForestRegressor () kullanmadan önce verilerdeki eksik değerleri değiştirmek için SimpleImputer() kullanır.

Random Forest modelindeki ağaç sayısını n\_estimators parametresi ile ayarladık ve random\_state ayarı tekrarlanabilirliği sağlıyor.



Cross-validation’da pipeline’ların nasıl kullanılacağını da öğrendiniz. Aşağıdaki kod, beş farklı fold arasında ortalaması alınmış ortalama mutlak hatayı (MAE) elde etmek için cross\_val\_score() işlevini kullanır.

Fold sayısını cv parametresi ile ayarladığımızı hatırlayın.



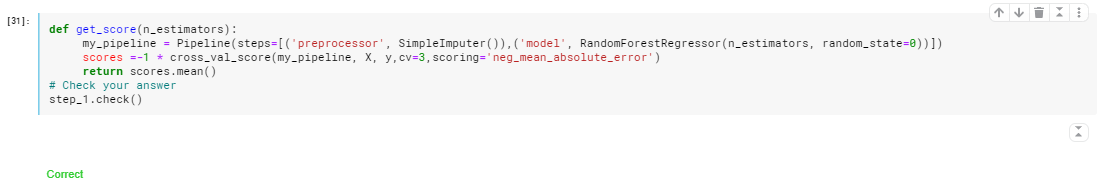
#### **Step 1: Write a Usefull Function**

Bu alıştırmada, bir makine öğrenimi modeli için parametreleri seçmek üzere cross validation kullanacaksınız.

Aşağıdakileri kullanan bir makine öğrenimi pipeline’nın MAE ortalamalarını bildiren (3 fold olacak) bir get\_score () işlevi yazarak başlayın:

* kıvrımlar oluşturmak için X ve y'deki veriler,
* Eksik değerleri değiştirmek için SimpleImputer() (tüm parametreler varsayılan olarak bırakılmıştır) ve
* Random Forest modelin fit etmek için RandomForestRegressor () (random\_state = 0 ile).

Get\_score() öğesine sağlanan **n\_estimators** parametresi, Random Forest modelindeki ağaç sayısı ayarlanırken kullanılır.

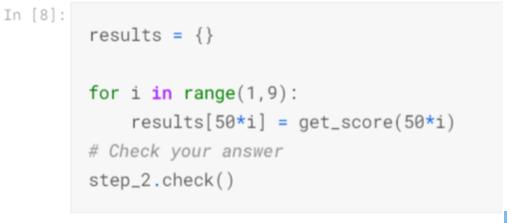


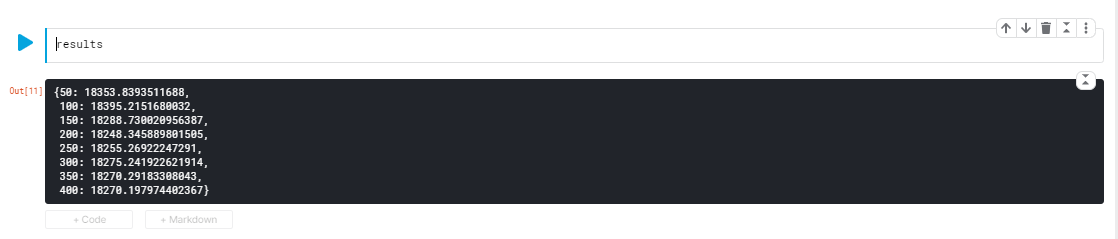
**İpucu:** Pipeline sınıfıyla bir pipeline yaparak başlayın. RandomForestRegressor () içindeki n\_estimators değerini get\_score işlevine sağlanan bağımsız değişkene ayarladığınızdan emin olun. Ardından, her fold için MAE'yi almak için cross\_val\_score() kullanın ve ortalamayı alın. Cv parametresi üzerinden fold sayısını üçe ayarladığınızdan emin olun.

#### **Step 2: Test Different Parameter Values**

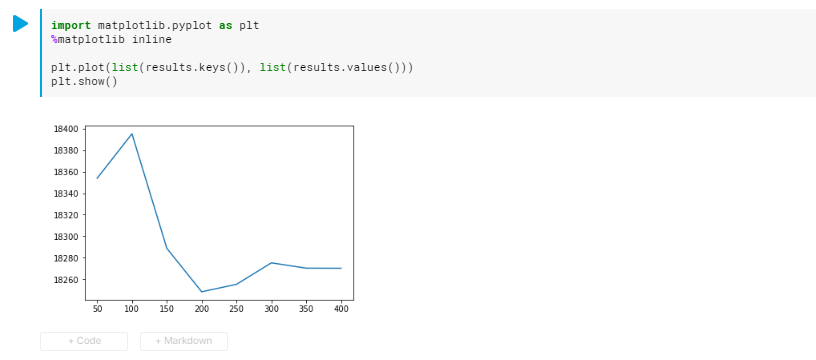
Şimdi Random Forest’daki ağaç sayısı için sekiz farklı değere karşılık gelen model performansını değerlendirmek için, Adım 1'de tanımladığınız işlevi kullanacaksınız: 50, 100, 150, ..., 300, 350, 400.

Sonuçlarınızı bir Python dictionary olan results’da saklayın; burada results[i], get\_score(i) tarafından döndürülen ortalama MAE'dir.

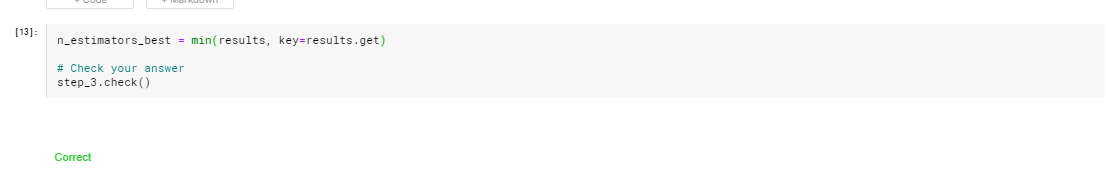




#### **Step 3: Find the Best Parameter Value**



Sonuçlar göz önüne alındığında, n\_estimators için hangi değer Random Forest modeli için en iyisi olarak görünüyor? Cevabınızı n\_estimators\_best değerini ayarlamak için kullanın.

Bu alıştırmada, bir makine öğrenme modelinde uygun parametreleri seçmek için bir yöntem araştırdınız.

[hyperparameter optimization](https://en.wikipedia.org/wiki/Hyperparameter_optimization) hakkında daha fazla bilgi edinmek isterseniz, bir makine öğrenimi modeli için en iyi parametre kombinasyonunu belirlemek için basit bir yöntem olan Grid Search ile başlamanız önerilir. Neyse ki, scikit-learn, Grid Search kodunuzu çok verimli hale getirebilen yerleşik bir işlev olan [GridSearchCV()](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)içerir!

Çeşitli veri kümelerinde son teknoloji sonuçlar elde eden güçlü bir teknik olan  [gradient boosting](https://www.kaggle.com/alexisbcook/xgboost)hakkında bilgi edinmeye devam edin.

# 

# KAYNAKLAR

* Kaggle – Intro to Machine Learnin Course

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

* <https://medium.com/data-science-tr/overfitting-underfitting-cross-validation-b47dfda0cf4e>
* <http://www.veridefteri.com/2017/11/23/scikit-learn-ile-veri-analitigine-giris/>
* <https://www.slideshare.net/VolkanOBANMsc/python-rastgele-ormanrandom-forest-parametreleri>
* <https://medium.com/@ahmetkuzubasli/modeliniz-neden-hala-hatal%C4%B1-bias-ve-variance-6368f36de751>
* <https://stackoverflow.com/questions/53249603/random-state-and-shuffle-together>
* <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>
* <https://www.slideshare.net/VolkanOBANMsc/python-rastgele-ormanrandom-forest-parametreleri>
* <https://thepythonguru.com/python-builtin-functions/zip/>
* <https://www.sharpsightlabs.com/blog/pandas-dropna/>

#### 