

# İçindekiler

|   |    |
|---|----|
| İçindekiler .....   | 1  |
| (Intro to Machine Learning)Makine Öğrenimine Giriş .....  | 3  |
| How Models Work (Modeller Nasıl Çalışır): .....   | 3  |
| Giriş:.....   | 3  |
| Decision Tree'nin Geliştirilmesi .....  | 4  |
| Basic Data Exploration (Temel Veri Keşfi) .....   | 5  |
| Using Pandas to Get Familiar With Your Data ( Verilerinizi Öğrenmek için Pandas kullanma):..... | 5  |
| Interpreting Data Description (Veri Açıklamalarını Yorumlama): .....                            | 6  |
| Excercise: Explore Your Data .....  | 6  |
| Your First Machine Learning Model: .....  | 8  |
| Selecting Data for Modeling (Modelleme için Veri Seçmek):.....                                  | 8  |
| Selecting The Prediction Target (Tahmin Hedefini Seçme) .....                                   | 11 |
| Choosing "Features" (Özellik Seçimi): .....   | 11 |
| Building Your Model (Model Oluşturma): .....  | 13 |
| Exercises: Your First Machine Learning Model.....   | 15 |
| Model Validation(Model geçerliliği):.....   | 18 |
| What is Model Validation: (Model Validation Nedir) .....  | 18 |
| The Problem with "In-Sample" Scores("In-Sample(Örnek İçi)" Puanlarla İlgili Sorun).....         | 20 |
| Coding It .....   | 20 |
| Wow! .....  | 21 |
| Exercises: Model Validation .....   | 21 |
| Underfitting and Overfitting.....   | 23 |
| Experimenting With Different Models .....   | 23 |
| Examples:.....  | 25 |
| Sonuç: .....  | 26 |
| Exercise: Underfitting and Overfitting.....   | 27 |
| Random Forests:.....  | 29 |
| Giriş:.....   | 29 |
| Example .....   | 29 |
| Sonuç: .....  | 30 |
| Exercises: Random Forest.....   | 30 |
| Exercises: Machine Learning Competitions.....   | 32 |
| Introduction.....   | 32 |
| Creating a Model For the Competition .....  | 33 |

|   |    |
|---|----|
| Make Predictions.....   | 33 |
| Quiz: Intro to Machine Learning.....                            | 34 |
| Intermediate Machine Learning (Orta Düzey Makine Öğrenimi)..... | 43 |
| Introduction (Giriş) .....                                      | 43 |
| Prerequisites (Önkoşullar) .....                                | 44 |
| Your Turn(Sıra Sende).....                                      | 44 |
| Exercise: Introduction .....                                    | 44 |
| Setup.....  | 44 |
| Missing Values (Eksik Değerler).....                            | 48 |
| Introduction (Giriş) .....                                      | 48 |
| Three Approaches .....  | 48 |
| Example(Örnek).....   | 49 |
| Exercises .....   | 52 |
| KAYNAKLAR .....   | 71 |

## (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:

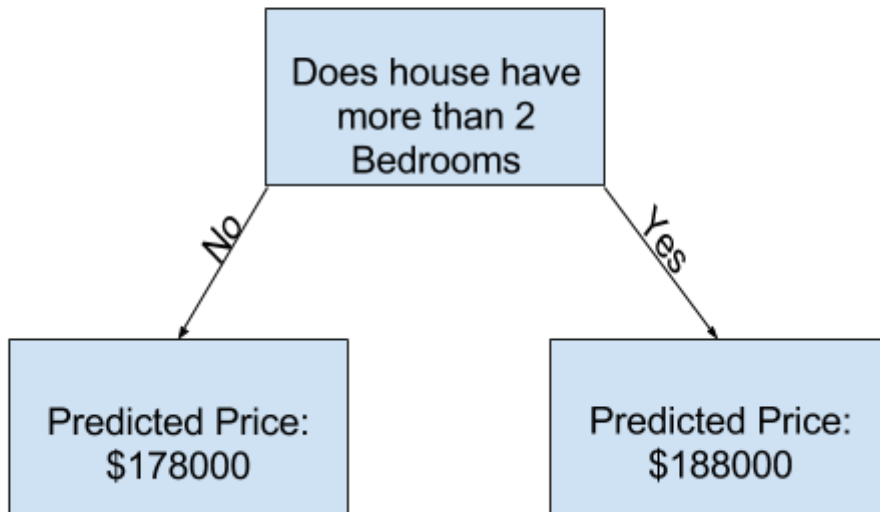
Kuzeniniz gayrimenkul konusunda spekülasyonlarla milyonlarca dolar kazandı. Veri bilimine gösterdiğiniz ilgi nedeniyle sizinle iş ortağı olmayı teklif etti. Parayı tedarik edecek ve çeşitli evlerin ne kadar değerli olduğunu tahmin eden modeller sunacaksınız.

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.

## Sample Decision Tree



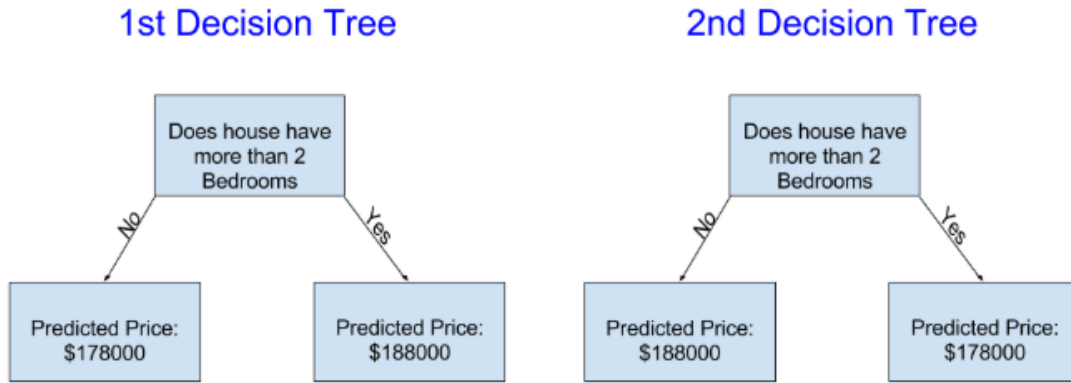
Evleri sadece iki kategoriye ayırır. Dikkate alınan herhangi bir ev için tahmini fiyat, aynı kategorideki evlerin tarihsel ortalama fiyatıdır.

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

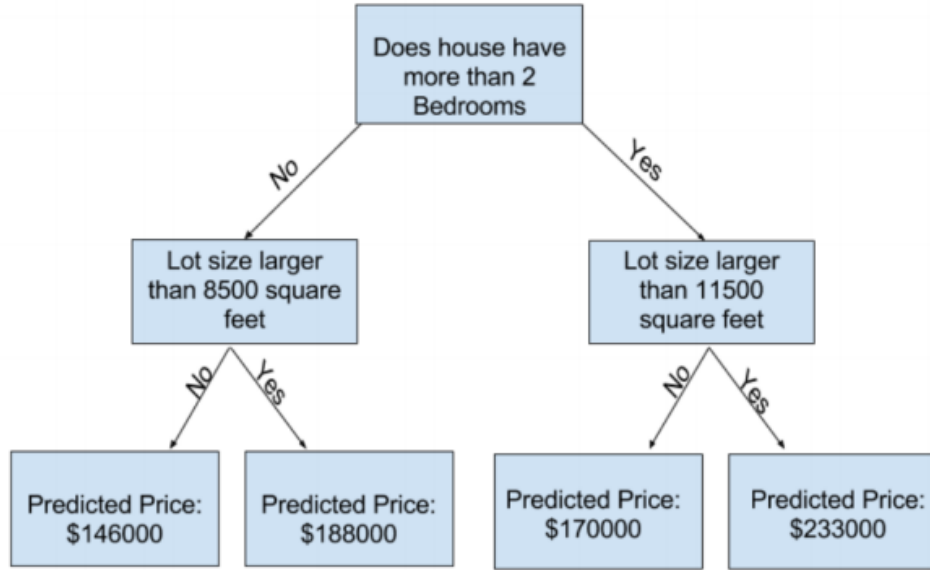
Aşağıdaki iki karardan hangisinin gayrimenkul eğitim verilerinin fit edilmesinden kaynaklanması daha olasıdır?



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 tanımdır. 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:

```
In [1]: import pandas as pd
```

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

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

Verileri aşağıdaki komutlarla yükler ve inceleriz:

- # kolay erişim için dosya yolunu değişkene kaydet

```
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
```

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

```
melbourne_data = pd.read_csv(melbourne_file_path)
```

```
melbourne_data.describe()
```

[14]:

|       | Rooms        | Price         | Distance     | Postcode     | Bedroom2     | Bathroom     | Car          | Landsize      | BuildingArea | YearBuilt   | Latitude     | Longitude    | Propertycount |
|-------|--------------|---------------|--------------|--------------|--------------|--------------|--------------|---------------|--------------|-------------|--------------|--------------|---------------|
| count | 13580.000000 | 1.3580000e+04 | 13580.000000 | 13580.000000 | 13580.000000 | 13580.000000 | 13518.000000 | 13580.000000  | 7130.000000  | 8205.000000 | 13580.000000 | 13580.000000 | 13580.000000  |
| mean  | 2.937997     | 1.075684e+06  | 10.137776    | 3105.301915  | 2.914728     | 1.534242     | 1.610075     | 558.416127    | 151.967650   | 1964.684217 | -37.809203   | 144.995216   | 7454.417378   |
| std   | 0.955748     | 6.393107e+05  | 5.868725     | 90.676964    | 0.965921     | 0.691712     | 0.962634     | 3990.669241   | 541.014538   | 37.273762   | 0.079260     | 0.103916     | 4378.581772   |
| min   | 1.000000     | 8.500000e+04  | 0.000000     | 3000.000000  | 0.000000     | 0.000000     | 0.000000     | 0.000000      | 0.000000     | 1196.000000 | -38.182550   | 144.431810   | 249.000000    |
| 25%   | 2.000000     | 6.500000e+05  | 6.100000     | 3044.000000  | 2.000000     | 1.000000     | 1.000000     | 177.000000    | 93.000000    | 1940.000000 | -37.856822   | 144.929600   | 4380.000000   |
| 50%   | 3.000000     | 9.030000e+05  | 9.200000     | 3084.000000  | 3.000000     | 1.000000     | 2.000000     | 440.000000    | 126.000000   | 1970.000000 | -37.802355   | 145.000100   | 6555.000000   |
| 75%   | 3.000000     | 1.330000e+06  | 13.000000    | 3148.000000  | 3.000000     | 2.000000     | 2.000000     | 651.000000    | 174.000000   | 1999.000000 | -37.756400   | 145.058305   | 10331.000000  |
| max   | 10.000000    | 9.000000e+06  | 48.100000    | 3977.000000  | 20.000000    | 8.000000     | 10.000000    | 433014.000000 | 44515.000000 | 2018.000000 | -37.408530   | 145.526350   | 21650.000000  |

## 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 düşüktен en yüksek değere doğru sıraladığınızı düşünün.

İlk (en küçük) değer **min**. Liste'nin dörtte birini incellerseniz, 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 alıştırmada, 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.



```
import pandas as pd

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

# Fill in the line below to read the file into a variable home_data
home_data = pd.read_csv(iowa_file_path)

# Call line below with no argument to check that you've loaded the data correctly
step_1.check()
```

Correct

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

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

```
# Print summary statistics in next line
home_data.describe()
```

Out[3]:

|       | Id          | MSSubClass  | LotFrontage | LotArea       | OverallQual | OverallCond | YearBuilt   | YearRemodAdd | MasVnrArea  | BsmtFinSF1  | ... |
|-------|-------------|-------------|-------------|---------------|-------------|-------------|-------------|--------------|-------------|-------------|-----|
| count | 1460.000000 | 1460.000000 | 1201.000000 | 1460.000000   | 1460.000000 | 1460.000000 | 1460.000000 | 1460.000000  | 1452.000000 | 1460.000000 | ... |
| mean  | 730.500000  | 56.897260   | 70.049958   | 10516.828082  | 6.099315    | 5.575342    | 1971.267808 | 1984.865753  | 103.685262  | 443.639726  | ... |
| std   | 421.610009  | 42.300571   | 24.284752   | 9981.264932   | 1.382997    | 1.112799    | 30.202904   | 20.645407    | 181.066207  | 456.098091  | ... |
| min   | 1.000000    | 20.000000   | 21.000000   | 1300.000000   | 1.000000    | 1.000000    | 1872.000000 | 1950.000000  | 0.000000    | 0.000000    | ... |
| 25%   | 365.750000  | 20.000000   | 59.000000   | 7553.500000   | 5.000000    | 5.000000    | 1954.000000 | 1967.000000  | 0.000000    | 0.000000    | ... |
| 50%   | 730.500000  | 50.000000   | 69.000000   | 9478.500000   | 6.000000    | 5.000000    | 1973.000000 | 1994.000000  | 0.000000    | 383.500000  | ... |
| 75%   | 1095.250000 | 70.000000   | 80.000000   | 11601.500000  | 7.000000    | 6.000000    | 2000.000000 | 2004.000000  | 166.000000  | 712.250000  | ... |
| max   | 1460.000000 | 190.000000  | 313.000000  | 215245.000000 | 10.000000   | 9.000000    | 2010.000000 | 2010.000000  | 1600.000000 | 5644.000000 | ... |

8 rows x 38 columns

| ... | WoodDeckSF  | OpenPorchSF | EnclosedPorch | 3SsnPorch   | ScreenPorch | PoolArea    | MiscVal      | MoSold      | YrSold      | SalePrice     |
|-----|-------------|-------------|---------------|-------------|-------------|-------------|--------------|-------------|-------------|---------------|
| ... | 1460.000000 | 1460.000000 | 1460.000000   | 1460.000000 | 1460.000000 | 1460.000000 | 1460.000000  | 1460.000000 | 1460.000000 | 1460.000000   |
| ... | 94.244521   | 46.660274   | 21.954110     | 3.409589    | 15.060959   | 2.758904    | 43.489041    | 6.321918    | 2007.815753 | 180921.195890 |
| ... | 125.338794  | 66.256028   | 61.119149     | 29.317331   | 55.757415   | 40.177307   | 496.123024   | 2.703626    | 1.328095    | 79442.502883  |
| ... | 0.000000    | 0.000000    | 0.000000      | 0.000000    | 0.000000    | 0.000000    | 0.000000     | 1.000000    | 2006.000000 | 34900.000000  |
| ... | 0.000000    | 0.000000    | 0.000000      | 0.000000    | 0.000000    | 0.000000    | 0.000000     | 5.000000    | 2007.000000 | 129975.000000 |
| ... | 0.000000    | 25.000000   | 0.000000      | 0.000000    | 0.000000    | 0.000000    | 0.000000     | 6.000000    | 2008.000000 | 163000.000000 |
| ... | 168.000000  | 68.000000   | 0.000000      | 0.000000    | 0.000000    | 0.000000    | 0.000000     | 8.000000    | 2009.000000 | 214000.000000 |
| ... | 857.000000  | 547.000000  | 552.000000    | 508.000000  | 480.000000  | 738.000000  | 15500.000000 | 12.000000   | 2010.000000 | 755000.000000 |

```
[10]: # What is the average lot size (rounded to nearest integer)?
      avg_lot_size = 10517

      # As of today, how old is the newest home (current year - the date in which it was built)
      newest_home_age = 10

      # Checks your answers
      step_2.check()

Correct
```

### 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 çıkılmaz 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.)

```
[1]: import pandas as pd
      melbourne_file_path='melb_data.csv'
      melbourne_data=pd.read_csv(melbourne_file_path)
      melbourne_data.columns

[1]: Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
          'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
          'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Latitude',
          'Longitude', 'Regionname', 'Propertycount'],
          dtype='object')
```



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

```
[10]: melbourne_data.isna().sum()
```

```
[10]: Suburb          0
      Address        0
      Rooms          0
      Type           0
      Price          0
      Method         0
      SellerG        0
      Date           0
      Distance       0
      Postcode       0
      Bedroom2       0
      Bathroom       0
      Car            62
      Landsize       0
      BuildingArea   6450
      YearBuilt      5375
      CouncilArea    1369
      Lattitude       0
      Longitude      0
      Regionname     0
      Propertycount  0
      dtype: int64
```

# Daha sonraki bir derste eksik değerleri ele almayı öğreneceğiz.

# Iowa verileriniz, kullandığınız sütunlarda eksik değerlere sahip değildi.

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

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

```
[3]: melbourne_data=melbourne_data.dropna(axis=0)
```

# Column'ların içinde kaç tane eksik veri var ona baktık.

```
[8]: melbourne_data.isna().sum()
```

```
[8]: Suburb          0
     Address        0
     Rooms          0
     Type           0
     Price          0
     Method         0
     SellerG        0
     Date           0
     Distance       0
     Postcode       0
     Bedroom2       0
     Bathroom       0
     Car            0
     Landsize       0
     BuildingArea   0
     YearBuilt      0
     CouncilArea    0
     Lattitude      0
     Longitude      0
     Regionname     0
     Propertycount  0
     dtype: int64
```

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.

```
y=melbourne_data.Price
y
```

|       |           |
|-------|-----------|
| 0     | 1480000.0 |
| 1     | 1035000.0 |
| 2     | 1465000.0 |
| 3     | 850000.0  |
| 4     | 1600000.0 |
| ...   |           |
| 13575 | 1245000.0 |
| 13576 | 1031000.0 |
| 13577 | 1170000.0 |
| 13578 | 2500000.0 |
| 13579 | 1285000.0 |

Name: Price, Length: 13580, dtype: float64

## 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:

```
[15]: melbourne_features=['Rooms','Bathroom','Landsize','Latitude','Longitude']
```

Kural olarak, bu verilere X denir.

```
[16]: X=melbourne_data[melbourne_features]
```

```
[19]:
```

|       | Rooms | Bathroom | Landsize | Latitude  | Longitude |
|-------|-------|----------|----------|-----------|-----------|
| 0     | 2     | 1.0      | 202.0    | -37.79960 | 144.99840 |
| 1     | 2     | 1.0      | 156.0    | -37.80790 | 144.99340 |
| 2     | 3     | 2.0      | 134.0    | -37.80930 | 144.99440 |
| 3     | 3     | 2.0      | 94.0     | -37.79690 | 144.99690 |
| 4     | 4     | 1.0      | 120.0    | -37.80720 | 144.99410 |
| ...   | ...   | ...      | ...      | ...       | ...       |
| 13575 | 4     | 2.0      | 652.0    | -37.90562 | 145.16761 |
| 13576 | 3     | 2.0      | 333.0    | -37.85927 | 144.87904 |
| 13577 | 3     | 2.0      | 436.0    | -37.85274 | 144.88738 |
| 13578 | 4     | 1.0      | 866.0    | -37.85908 | 144.89299 |
| 13579 | 4     | 1.0      | 362.0    | -37.81188 | 144.88449 |

13580 rows × 5 columns

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

```
[20]: X.describe()
```

```
[20]:
```

|       | Rooms        | Bathroom     | Landsize      | Latitude     | Longitude    |
|-------|--------------|--------------|---------------|--------------|--------------|
| count | 13580.000000 | 13580.000000 | 13580.000000  | 13580.000000 | 13580.000000 |
| mean  | 2.937997     | 1.534242     | 558.416127    | -37.809203   | 144.995216   |
| std   | 0.955748     | 0.691712     | 3990.669241   | 0.079260     | 0.103916     |
| min   | 1.000000     | 0.000000     | 0.000000      | -38.182550   | 144.431810   |
| 25%   | 2.000000     | 1.000000     | 177.000000    | -37.856822   | 144.929600   |
| 50%   | 3.000000     | 1.000000     | 440.000000    | -37.802355   | 145.000100   |
| 75%   | 3.000000     | 2.000000     | 651.000000    | -37.756400   | 145.058305   |
| max   | 10.000000    | 8.000000     | 433014.000000 | -37.408530   | 145.526350   |

```
[21]: X.head()
```

```
[21]:
```

|   | Rooms | Bathroom | Landsize | Lattitude | Longitude |
|---|-------|----------|----------|-----------|-----------|
| 0 | 2     | 1.0      | 202.0    | -37.7996  | 144.9984  |
| 1 | 2     | 1.0      | 156.0    | -37.8079  | 144.9934  |
| 2 | 3     | 2.0      | 134.0    | -37.8093  | 144.9944  |
| 3 | 3     | 2.0      | 94.0     | -37.7969  | 144.9969  |
| 4 | 4     | 1.0      | 120.0    | -37.8072  | 144.9941  |

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

```
[23]: from sklearn.tree import DecisionTreeRegressor

#Modeli tanımlayın. Her çalıştırmada aynı sonuçları sağlamak için random_state için bir sayı belirtin
melbourne_model=DecisionTreeRegressor(random_state=1)
#Fit model
melbourne_model.fit(X,y)
```

```
[23]: DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, presort='deprecated',
                           random_state=1, splitter='best')
```

**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.

```
[24]: print("Making predictions for the following 5 houses:")
      print(X.head())
      print("The predictions are")
      print(melbourne_model.predict(X.head()))
```

```
Making predictions for the following 5 houses:
   Rooms  Bathroom  Landsize  Lattitude  Longtitude
0       2         1.0     202.0    -37.7996     144.9984
1       2         1.0     156.0    -37.8079     144.9934
2       3         2.0     134.0    -37.8093     144.9944
3       3         2.0      94.0    -37.7969     144.9969
4       4         1.0     120.0    -37.8072     144.9941
The predictions are
[1480000. 1035000. 1465000.  850000. 1600000.]
```

## 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

```
1: # print the list of columns in the dataset to find the name of the prediction target
home_data.columns
```

```
2: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
        'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
        'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
        'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
        'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
        'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
        'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
        'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
        'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
        'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
        'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
        'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
        'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
        'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
        'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
        'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
        'SaleCondition', 'SalePrice'],
        dtype='object')
```



```
y = home_data.SalePrice
# Check your answer
step_1.check()
```

# 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 :

- LotArea
- YearBuilt
- 1stFlrSF
- 2ndFlrSF
- FullBath
- BedroomAbvGr
- TotRmsAbvGrd

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

```
[19]: # Create the list of features below
feature_names = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']

# Select data corresponding to features in feature_names
X = home_data[feature_names]

# Check your answer
step_2.check()
```

### Review Data

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

```
[22]: # Review data
# print description or statistics from X
print(X.describe())

# print the top few lines
print("\n", X.head())
```



|       | LotArea       | YearBuilt   | 1stFlrSF    | 2ndFlrSF    | FullBath    | \ |
|-------|---------------|-------------|-------------|-------------|-------------|---|
| count | 1460.000000   | 1460.000000 | 1460.000000 | 1460.000000 | 1460.000000 |   |
| mean  | 10516.828082  | 1971.267808 | 1162.626712 | 346.992466  | 1.565068    |   |
| std   | 9981.264932   | 30.202904   | 386.587738  | 436.528436  | 0.550916    |   |
| min   | 1300.000000   | 1872.000000 | 334.000000  | 0.000000    | 0.000000    |   |
| 25%   | 7553.500000   | 1954.000000 | 882.000000  | 0.000000    | 1.000000    |   |
| 50%   | 9478.500000   | 1973.000000 | 1087.000000 | 0.000000    | 2.000000    |   |
| 75%   | 11601.500000  | 2000.000000 | 1391.250000 | 728.000000  | 2.000000    |   |
| max   | 215245.000000 | 2010.000000 | 4692.000000 | 2065.000000 | 3.000000    |   |

|       | BedroomAbvGr | TotRmsAbvGrd |
|-------|--------------|--------------|
| count | 1460.000000  | 1460.000000  |
| mean  | 2.866438     | 6.517808     |
| std   | 0.815778     | 1.625393     |
| min   | 0.000000     | 2.000000     |
| 25%   | 2.000000     | 5.000000     |
| 50%   | 3.000000     | 6.000000     |
| 75%   | 3.000000     | 7.000000     |
| max   | 8.000000     | 14.000000    |

|   | LotArea | YearBuilt | 1stFlrSF | 2ndFlrSF | FullBath | BedroomAbvGr | \ |
|---|---------|-----------|----------|----------|----------|--------------|---|
| 0 | 8450    | 2003      | 856      | 854      | 2        | 3            |   |
| 1 | 9600    | 1976      | 1262     | 0        | 2        | 3            |   |
| 2 | 11250   | 2001      | 920      | 866      | 2        | 3            |   |
| 3 | 9550    | 1915      | 961      | 756      | 1        | 3            |   |
| 4 | 14260   | 2000      | 1145     | 1053     | 2        | 4            |   |

|   | TotRmsAbvGrd |
|---|--------------|
| 0 | 8            |
| 1 | 6            |
| 2 | 6            |
| 3 | 7            |
| 4 | 9            |

### Step 3: Specify and Fit Model:

**DecisionTreeRegressor** oluştur ve `iowa_model`'e kaydet. Bu komutu çalıştırmak için **sklearn**'de ilgili import işlemini yaptığınızdan emin olun.

```
[23]: from sklearn.tree import DecisionTreeRegressor
#modeli belirtin
#Model tekrarlanabilirliği için, modeli belirtirken random_state için sayısal bir değer belirleyin
iowa_model = DecisionTreeRegressor(random_state=1)

# Fit the model
iowa_model.fit(X,y)

# Check your answer
step_3.check()
```

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

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

### Notlar:

- **predict**: Regresyon, sınıflandırma, kümeleme gibi yöntemler kullanarak yapacağınız çalışmalarda tahmin edilen etiket bilgisini **predict** fonksiyonuyla elde edebilirsiniz. Sınıflandırma problemlerinde gözlemlerin sınıflara ait olma olasılıklarını elde etmek istiyorsanız **predict\_proba** fonksiyonunu kullanmanız gerekiyor.

**LinearRegression** nesnesi ile modelimizi oluşturalım ve train datamız ile de modelimizi besleyelim.

```
model = LinearRegression()
model.fit(X_train, y_train)
```

Şimdi ise test datamızla modelimize tahmin ürettirelim.

```
prediction = model.predict(X_test)
print(prediction)

[ 55870.35248488 124217.47107547  53061.56678938 114854.85209045
 55870.35248488 115791.11398896  63360.44767289  92384.56652643
 63360.44767289 102683.44740994]
```

Lightshot  
Ekran Gö

## 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 "Training data" daki hedef değerlerle karşılaştırırlar.

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

Önce model kalitesini 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 değerlerin 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:

```
error=actual-predicted
```

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'Yi 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:

```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

**434.71594577146544**

```
# Data Loading Code Hidden Here
import pandas as pd

# Load data
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
# Filter rows with missing price values
filtered_melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = filtered_melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                      'YearBuilt', 'Lattitude', 'Longtitude']
X = filtered_melbourne_data[melbourne_features]

from sklearn.tree import DecisionTreeRegressor
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(X, y)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,  
                      max_leaf_nodes=None, min_impurity_decrease=0.0,  
                      min_impurity_split=None, min_samples_leaf=1,  
                      min_samples_split=2, min_weight_fraction_leaf=0.0,  
                      presort=False, random_state=None, splitter='best')
```

## 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:

```
[41]: from sklearn.model_selection import train_test_split  
#verileri hem özellikler hem de hedef için eğitim ve doğrulama verilerine bölün  
#Bölünmüş bir rasgele sayı üretici dayanmaktadır. Sayısal bir değer sağlama  
#random_state argümanı, her seferinde aynı bölünmeyi elde ettiğimizi garanti eder  
#bu komut dosyasını çalıştırın.  
train_X, val_X, train_y, val_y=train_test_split(X,y,random_state=0)  
#Modeli tanımla  
melbourne_model=DecisionTreeRegressor()  
# fit model  
melbourne_model.fit(train_X,train_y)  
#doğrulama verilerinde öngörülen fiyatları alın  
val_predictions = melbourne_model.predict(val_X)  
print(mean_absolute_error(val_y, val_predictions))
```

246802.63033873343

]:

## 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 alıştırma modelinizin ne kadar iyi olduğunu test edeceksiniz.

Önceki alıştırmanın kaldığı kodlama ortamınızı ayarlamak için aşağıdaki hücreyi çalıştırın.

```
# Daha önce veri yüklemek için kullandığınız kod
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
# Okunacak dosya yolu
iowa_file_path='melb_data.csv'
home_data = pd.read_csv(iowa_file_path)
y = home_data.SalePrice
feature_columns = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[feature_columns]
# Modeli Belirle
iowa_model = DecisionTreeRegressor()
# Fit Model
iowa_model.fit(X, y)
print("First in-sample predictions:", iowa_model.predict(X.head()))
print("Actual target values for those homes:", y.head().tolist())
# Kod kontrolünü ayarlama
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex4 import *
print("Setup Complete")
```

### 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.

```
[13]: #Import the train_test_split function and uncomment
      from sklearn.model_selection import train_test_split

      # fill in and uncomment
      train_X, val_X, train_y, val_y = train_test_split(X,y,random_state=1)

      # Check your answer
      step_1.check()
```

Correct

### Step 2: Specify and Fit the Model

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

```
[15]: # You imported DecisionTreeRegressor in your last exercise
# and that code has been copied to the setup code above. So, no need to
# import it again

# Specify the model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit iowa_model with the training data.
iowa_model.fit(train_X, train_y)

# Check your answer
step_2.check()
```

```
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
262000.]
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
262000.]
```

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

```
[17]: # Predict with all validation observations
val_predictions = val_predictions = iowa_model.predict(val_X)

# Check your answer
step_3.check()
```

Doğrulama verilerinden tahminlerinizi ve gerçek değerlerinizi inceleyin.

```
[21]: # print the top few validation predictions
print(mean_absolute_error(val_y, val_predictions))
# print the top few actual prices from validation data
print(val_X.head())
```

```
29652.931506849316
   LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
258    12435     2001      963      829         2             3
267     8400     1939     1052      720         2             4
288     9819     1967      900         0         1             3
649     1936     1970      630         0         1             1
1233    12160     1959     1188         0         1             3

   TotRmsAbvGrd
258             7
267             8
288             5
649             3
1233            6
```

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)

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

+ Code

+ Markdown

```
[1]: from sklearn.metrics import mean_absolute_error
val_mae = mean_absolute_error(val_y, val_predictions)

# uncomment following line to see the validation_mae
print(val_mae)

# Check your answer
step_4.check()
```

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.

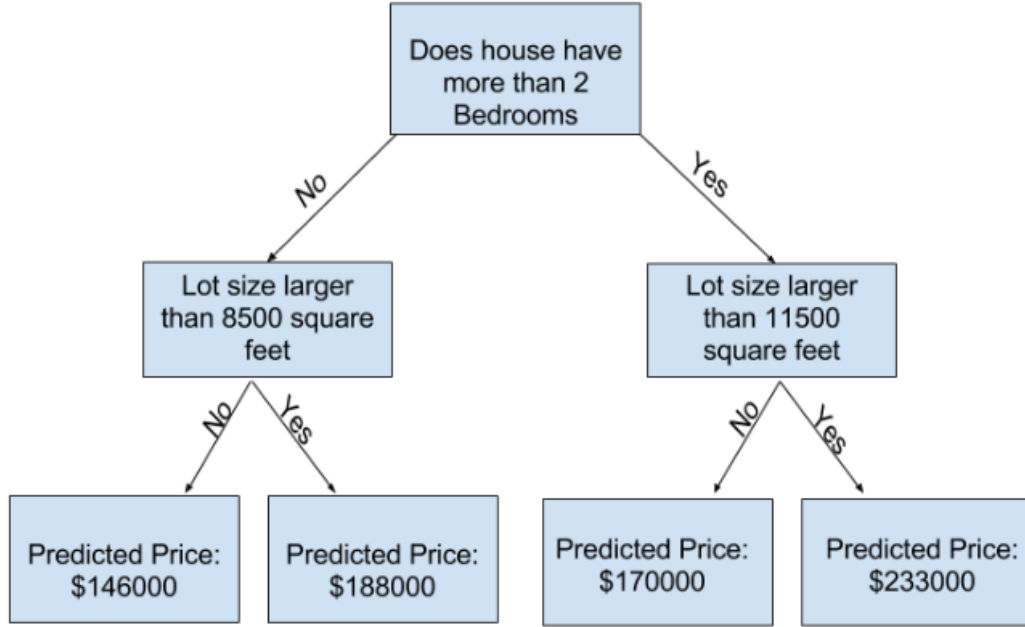
## **Experimenting With Different Models**

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  $2^{10}$  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 güvenilir 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.

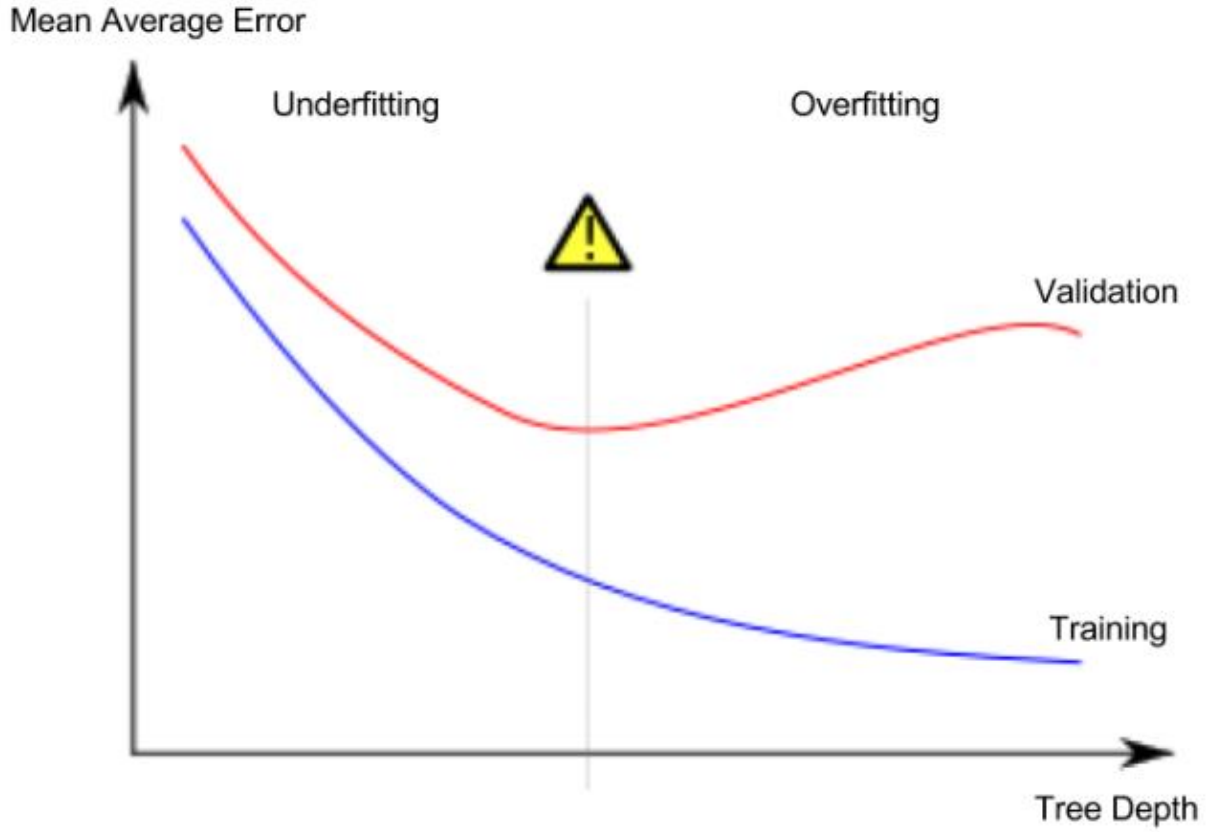
Aşırı derecede, eğer bir ağaç evleri sadece 2 veya 4'e bölerse, her grup hala çok çeşitli evlere sahiptir.

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 önemseydiğimizden, **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:

```
In [1]:
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor

def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=0)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)
```

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 karşılaştırmak

```
for max_leaf_nodes in [5, 50, 500, 5000]:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes: %d \t\t Mean Absolute Error: %d" % (max_leaf_nodes, my_mae))
```

|                      |                             |
|----------------------|-----------------------------|
| Max leaf nodes: 5    | Mean Absolute Error: 347380 |
| Max leaf nodes: 50   | Mean Absolute Error: 258171 |
| Max leaf nodes: 500  | Mean Absolute Error: 243495 |
| Max leaf nodes: 5000 | Mean Absolute Error: 254983 |

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:

*Burada eğitim seti geçmiş 10 yılın soruları, model sınavın geçmiş senelere benzeyeceğini düşünmeniz, test seti hiç görmediğimiz istatistik sınavı, başarı kriteri aldığınız not. Sınav soruları beklendiğiniz gibi gelmez de kötü not alırsanız bu olaya overfitting denir.*

## Underfitting:

*Hoca bu konuyu sormaz şu konuyu sormaz diye diye kafanıza göre konuları çalışmaktan vazgeçip düşük not alırsanız buna da underfitting denir.*

*Diğer bir deyişle eğer modelimizi eğitim (training) veri seti üzerinde çok basit olarak kurguladıysak hiç görmediğimiz test verisi üzerinde başarısız tahminler (sallama) yaparız ve gerçek değerle tahmin ettiğimiz değer arasındaki fark çok olur.*

## Exercise: Underfitting and Overfitting

### Tekrarlamak:

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

```
# Code you have previously used to load data
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)

# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE: {:.0f}".format(val_mae))

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex5 import *
print("Setup complete")
```

Validation MAE: 29,653  
Setup complete

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

```
[1]: def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=0)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)
```

### 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.

```
candidate_max_leaf_nodes = [5, 25, 50, 100, 250, 500]
# Candidate_max_leaf_nodes'dan ideal ağaç boyutunu bulmak için döngü yaz
for max_leaf_nodes in candidate_max_leaf_nodes:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes:{0} \t\t Mean Absolute Error:{1}".format(max_leaf_nodes, my_mae))
# Max_leaf_nodes en iyi değerini saklayın (5, 25, 50, 100, 250 veya 500 olacaktır)
best_size=100

# Check your answer
step_1.check()
```

|                    |  |
|--------------------|--|
| Max leaf nodes:5   | Mean Absolute Error:35044.51299744237  |
| Max leaf nodes:25  | Mean Absolute Error:29016.41319191076  |
| Max leaf nodes:50  | Mean Absolute Error:27405.930473214907 |
| Max leaf nodes:100 | Mean Absolute Error:27282.50803885739  |
| Max leaf nodes:250 | Mean Absolute Error:27893.822225701646 |
| Max leaf nodes:500 | Mean Absolute Error:29454.18598068598  |

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

En iyi ağaç boyutunu biliyorsunuz. 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.

```
19]: # Fill in argument to make optimal size and uncomment
final_model = DecisionTreeRegressor(max_leaf_nodes=best_tree_size, random_state=1)

# fit the final model and uncomment the next two lines
final_model.fit(X,y)

# Check your answer
step_2.check()
```

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

```
In [1]: import pandas as pd

# Load data
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
# Filter rows with missing values
melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                      'YearBuilt', 'Lattitude', 'Longtitude']
X = melbourne_data[melbourne_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and target
# The split is based on a random number generator. Supplying a numeric value to
# the random_state argument guarantees we get the same split every time we
# run this script.
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)
```

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

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/forests.py:245: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

202888.18157951365
```

### 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'in 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:

```

# Code you have previously used to load data
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)

# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE when not specifying max_leaf_nodes: {:.0f}".format(val_mae))

# Using best value for max_leaf_nodes
iowa_model = DecisionTreeRegressor(max_leaf_nodes=100, random_state=1)
iowa_model.fit(train_X, train_y)
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE for best value of max_leaf_nodes: {:.0f}".format(val_mae))

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex6 import *
print("\nSetup complete")

```

```

Validation MAE when not specifying max_leaf_nodes: 29,653
Validation MAE for best value of max_leaf_nodes: 27,283

Setup complete

```

## 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

```

[5]: from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
# Modeli tanımlayın. Random_state ögesini 1 olarak ayarla
rf_model = RandomForestRegressor(random_state=1)

# fit your model
rf_model.fit(train_X, train_y)
rf_model_predictions = rf_model.predict(val_X)
# Calculate the mean absolute error of your Random Forest model on the validation data
rf_val_mae = mean_absolute_error(val_y, rf_model_predictions)

print("Validation MAE for Random Forest Model: {}".format(rf_val_mae))

# Check your answer
step_1.check()

```

```

Validation MAE for Random Forest Model: 21857.15912981083

```

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 alıştırma, 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 etkilendiğini görmek için yeniden gönderebilirsiniz.

Şimdiye kadar yazdığımız kod:



```

# Code you have previously used to load data
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Set up code checking
import os
if not os.path.exists("../input/train.csv"):
    os.symlink("../input/home-data-for-ml-course/train.csv", "../input/train.csv")
    os.symlink("../input/home-data-for-ml-course/test.csv", "../input/test.csv")
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex7 import *

# Path of the file to read. We changed the directory structure to simplify submitting to a competi
iowa_file_path = '../input/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['lotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbv
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)

# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE when not specifying max_leaf_nodes: {:.0f}".format(val_mae))

# Using best value for max_leaf_nodes
iowa_model = DecisionTreeRegressor(max_leaf_nodes=100, random_state=1)
iowa_model.fit(train_X, train_y)
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE for best value of max_leaf_nodes: {:.0f}".format(val_mae))

# Define the model. Set random_state to 1
rf_model = RandomForestRegressor(random_state=1)
rf_model.fit(train_X, train_y)
rf_val_predictions = rf_model.predict(val_X)
rf_val_mae = mean_absolute_error(rf_val_predictions, val_y)

print("Validation MAE for Random Forest Model: {:.0f}".format(rf_val_mae))

```

```

Validation MAE when not specifying max_leaf_nodes: 29.653
Validation MAE for best value of max_leaf_nodes: 27.283
Validation MAE for Random Forest Model: 21.857

```

## Creating a Model For the Competition

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

```

In [2]:
# To improve accuracy, create a new Random Forest model which you will train on all training data
rf_model_on_full_data = RandomForestRegressor(random_state=1)

# fit rf_model_on_full_data on all data from the training data
rf_model_on_full_data.fit(X, y)

Out[2]:
RandomForestRegressor(random_state=1)

```

## Make Predictions

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

```
In [3]: # path to file you will use for predictions
test_data_path = '../input/test.csv'

# read test data file using pandas
test_data = pd.read_csv(test_data_path)

# create test_X which comes from test_data but includes only the columns you used for prediction.
# The list of columns is stored in a variable called features
test_X = test_data[features]
# make predictions which we will submit.
test_preds = rf_model_on_full_data.predict(test_X)

# The lines below shows how to save predictions in format used for competition scoring
# Just uncomment them.
output = pd.DataFrame({'Id': test_data.Id,
                       'SalePrice': test_preds})

output.to_csv('submission.csv', index=False)
```

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

- ✓ Q1- After training our decision tree model, we saw that the model is 10/10 overfitted on the training data and it has bad performance on the test data. Which hyper-parameter could help us to get rid of this problem? Note: You can use `sklearn.tree.DecisionTreeClassifier` documentation. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier> \*
- ☐ criterion
- ☒ max\_depth ✓
- ☐ random\_state
- ☐ splitter

#### AÇIKLAMA:

Karar ağacı modelimizi eğittikten sonra, modelin eğitim verilerine fazla uyduğunu ve test verilerinde kötü performans gösterdiğini gördük. Hangi hiper parametre bu problemten kurtulmamıza yardımcı olabilir? Not: `sklearn.tree.DecisionTreeClassifier` belgelerini kullanabilirsiniz. <http://scikit->

learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier \*

### ➤ **max\_depth**-(integer or none)- Default=None

Bu, ağaçlarınızı ne kadar derin yapmak istediğinizi seçer. Max\_depth'inizi ayarlamanızı önerilir, çünkü overfitting baş etmek için önerilir.

- Ağacın kökü ve yapraklar arasındaki maksimum bağlantı sayısı. Küçük olmalı.

- ✓ Q2- Which of the below can be said definitely according to the results 10/10 table taken from the data.describe() method? I. 75% of the values in the Rooms column are greater than 2. II. There are some houses with a land size of 0. III. There are missing values in the BuildingArea column. IV. There is no house with 9 rooms in the data set \*

```
In [2]: import pandas as pd
```

```
data = pd.read_csv("/home/fatih/Desktop/melb_data.csv")
```

```
data.describe()
```

Out[2]:

|       | Rooms        | Price        | Distance     | Postcode     | Bedroom2     | Bathroom     | Car          | Landsize      | BuildingArea | YearBuilt   |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|---------------|--------------|-------------|
| count | 13580.000000 | 1.358000e+04 | 13580.000000 | 13580.000000 | 13580.000000 | 13580.000000 | 13518.000000 | 13580.000000  | 7130.000000  | 8205.000000 |
| mean  | 2.937997     | 1.075684e+06 | 10.137776    | 3105.301915  | 2.914728     | 1.534242     | 1.610075     | 558.416127    | 151.967650   | 1964.684217 |
| std   | 0.955748     | 6.393107e+05 | 5.868725     | 90.676964    | 0.965921     | 0.691712     | 0.962634     | 3990.669241   | 541.014538   | 37.273762   |
| min   | 1.000000     | 8.500000e+04 | 0.000000     | 3000.000000  | 0.000000     | 0.000000     | 0.000000     | 0.000000      | 0.000000     | 1196.000000 |
| 25%   | 2.000000     | 6.500000e+05 | 6.100000     | 3044.000000  | 2.000000     | 1.000000     | 1.000000     | 177.000000    | 93.000000    | 1940.000000 |
| 50%   | 3.000000     | 9.030000e+05 | 9.200000     | 3084.000000  | 3.000000     | 1.000000     | 2.000000     | 440.000000    | 126.000000   | 1970.000000 |
| 75%   | 3.000000     | 1.330000e+06 | 13.000000    | 3148.000000  | 3.000000     | 2.000000     | 2.000000     | 651.000000    | 174.000000   | 1999.000000 |
| max   | 10.000000    | 9.000000e+06 | 48.100000    | 3977.000000  | 20.000000    | 8.000000     | 10.000000    | 433014.000000 | 44515.000000 | 2018.000000 |

☐ I, II

☐ II, III

☐ II, III, IV

☒ I, II, III



#### AÇIKLAMA

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

- I. Rooms sütunundaki değerlerin %75'i 2'den büyüktür.
- II. II. land size'ı 0 olan bazı evler vardır.
- III. III. BuildingArea sütununda eksik değerler vardır.

IV. IV. veri kümesinde 9 odalı bir ev yoktur

✓ Q3- Which one is false about overfitting and underfitting? \*

10/10

- ☐ Insufficient training (less epoch less batch size), causes underfitting.
- ☐ Training on too much epoch and batch size causes overfitting.
- ☒ Splitting dataset as train and test datasets will always be enough to prevent overfitting, no need for validation datasets. ✓
- ☐ In overfitting accuracy will be very good at train data but will be very bad at unseen data.

#### 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.

✓ Q4- Which of the following is false regarding pandas and scikit-learn methods? \*

10/10

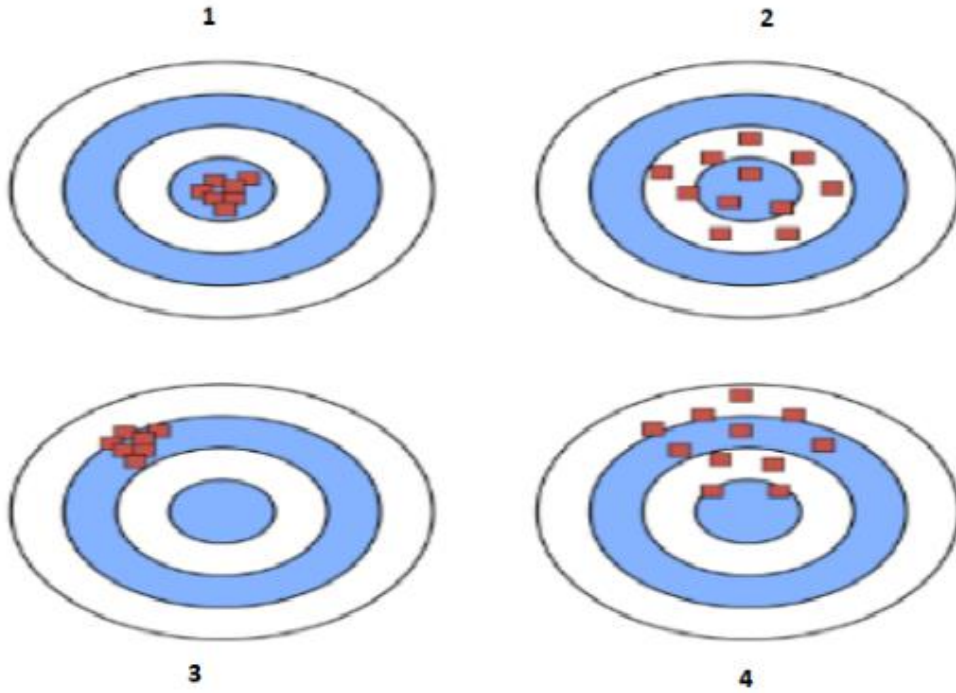
- ☐ DataFrame.head(x) shows x samples in the DataFrame from the beginning.
- ☐ DataFrame.describe() shows summary of the data.
- ☒ model.predict() determines how accurate the model's predictions are. ✓
- ☐ DataFrame.dropna(axis=0) drops missing values.

#### 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.
- **predict**: Regresyon, sınıflandırma, kümeleme gibi yöntemler kullanarak yapacağınız çalışmalarda tahmin edilen etiket bilgisini **predict** fonksiyonuyla elde edebilirsiniz.

✓ Q5- According to the shooting clusters scheme above, for each figure 10/10 which statements are true? Notice that, shooting targets are the centers. \*



☒ 1:Low Bias- Low Variance 2:Low Bias-High Variance 3:High Bias-Low Variance 4: High Bias-High Variance ✓

☐ 1:Low Bias- High Variance 2:Low Bias-Low Variance 3:High Bias-High Variance 4: High Bias-Low Variance

○ 1:High Bias- Low Variance 2: High Bias-High Variance 3:Low Bias-Low Variance  
4:Low Bias-High Variance

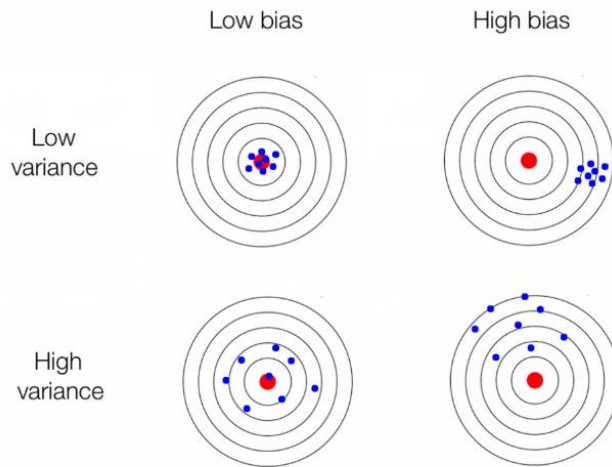
○ 1:High Bias- High Variance 2:High Bias-Low Variance 3:Low Bias-High Variance  
4:Low Bias-Low Variance

#### 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

- **Bias.** Bu modelin eğitim seti üzerindeki %15'lik hatasını içeriyor. Kabaca, *modelin yanlılığı* olarak düşünebiliriz.
- **Variance.** Bu da modelin test setindeki performansının, eğitim setine göre ne kadar değiştiğini, kötüleştiğini gösteriyor. Bunu da kabaca, *modelin varyansı (değişkenliği)* olarak kaydedelim.





Hedef tahtası görseli ile örneklerimizi ilişkilendirecek olursak, hedefimiz düşük bias ve düşük variance yani sol üst köşedeki gibi büyük oranda hedefi vuruyor olmamız. Orijinal örneğimizde variance düşük (atışlarımız arasındaki değişkenlik düşük) ancak yüksek bias problemi var yani sağ üst köşedeki gibi *yanlı* atıyoruz. İkinci örneğimizdeyse, eğitim setinde attığımızı vuruyoruz, *yanlı* atmıyoruz bu yüzden bias düşük ancak test sırasında performansımız *değişkenlik* gösteriyor yani variance yüksek.

✓ Q6- According to the random forests algorithm, which of the below statements are true? \* 10/10

- I - It is an algorithm that aims to increase the classification value by producing multiple decision trees.
- II - It was created by combining Bagging and Random Subspace methods.
- III - While creating the tree, it is made performance evaluation with 2/3 of the data set.

☐ I, III

☐ II, III

☒ I, II



☐ I, II, III

#### 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

✓ Q7- What do you think about train\_X when line 1 and line 2 are executed separately? The rest of the code is exactly the same. \*

10/10

```
Line 1. train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 2,shuffle=False)
Line 2. train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 1,shuffle=False)
```

- ☐ They generate different random number so the train\_X differs from each other.
- ☐ They generate different same number and the train\_X is equal to each other.
- ☐ They generate different random number so the train\_X is equal to each other.
- ☒ They generate different random number ,but the train\_X is equal to each other. ✓

#### 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.

#### **shuffle : bool, default = Doğru**

Bölmeden önce verilerin karıştırılıp karıştırılmayacağı. Shuffle = False ise, katmanlama Yok olmalıdır.

#### **random\_state : int veya RandomState örneği, default = Yok**

Bölmeyi uygulamadan önce verilere uygulanan karıştırma işlemini kontrol eder. Birden çok işlev çağrısında tekrarlanabilir çıkış için bir int iletin. Bkz. Sözlük .



`shuffle` False olarak ayarlarsanız `train_test_split`, verilerinizi orijinal düzeninde okur. Bu nedenle parametre `random_state` tamamen yok sayılır.

Misal:

```
X = [k for k in range(0, 50)] # create array with numbers ranging from 0 to 49
y = X # just for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42,
print(X_train) // prints [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
```

`shuffle` True olarak ayarladığınız anda `random_state`, rastgele sayı üretici için tohum olarak kullanılır. Sonuç olarak, veri kümeniz rastgele olarak tren ve test kümesine ayrılır.

Random\_state = 42 ile örnek:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42,
print(X_train) // prints [8, 3, 6, 41, 46, 47, 15, 9, 16, 24, 34, 31, 0, 44, 27, 33, 5, 29,
```

Random\_state = 44 ile örnek:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=44,
print(X_train) // prints [13, 11, 2, 12, 34, 41, 30, 16, 39, 28, 24, 8, 18, 9, 4, 10, 0, 19
```

- ✓ Q8- Trees have their length and we call that the depth of the tree. 10/10  
RandomForestRegressor, in scikit-learn library, has a maximum leaf ( `max_depth` ) parameter which is None as default which means nodes are expanded until all leaves are pure. What can be said if we change the number of maximum leaf nodes of a random forest? \*
- ☐ Length of a tree does not affect any of the results.
  - ☒ Model may overfit for large depth values. ✓
  - ☐ The longer tree is the better tree.
  - ☐ Short trees more precise than long trees.

#### AÇIKLAMA

Ağaçların uzunluğu var ve buna ağacın derinliği diyoruz. RandomForestRegressor, scikit-learn kütüphanesinde, varsayılan olarak hiçbirisi 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.

✓ Q9- Let assume, we have a data set called home\_data with 3 features names; LotArea, YearBuilt, PoolArea. How do you define non-missing values for the feature LotArea? \* 10/10

- ☐ non\_missings = home\_data["LotArea"].mean()
- ☐ non\_missings = home\_data.count()
- ☒ non\_missings = home\_data["LotArea"].count() ✓
- ☐ non\_missings = home\_data.mean()

#### AÇIKLAMA

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

✓ Q10- What is the aim of the below code pieces? \* 10/10

```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

- ☐ For splitting the data as test and train
- ☐ For interpreting the data description
- ☒ For summarizing model quality ✓
- ☐ For data modelling

## 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 artıracığınızı 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** (boru hatları) 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 alıştırma 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.

```
[1]: # Set up code checking
import os
if not os.path.exists("../input/train.csv"):
    os.symlink("../input/home-data-for-ml-course/train.csv", "../input/train.csv")
    os.symlink("../input/home-data-for-ml-course/test.csv", "../input/test.csv")
from learntools.core import binder
binder.bind(globals())
from learntools.ml_intermediate.ex1 import *
print("Setup Complete")
```

Setup Complete

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 hücrelerini 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.)

```
[1]: import pandas as pd
      from sklearn.model_selection import train_test_split

[3]: # Verileri oku
      X_full=pd.read_csv("train.csv",index_col='Id')
      X_test_full=pd.read_csv('test.csv')
      # Hedef ve tahmincileri elde edin
      y=X_full.SalePrice
      features=['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
      X=X_full[features].copy()
      X_test=X_test_full[features].copy()
      # Training verilerinden validation verilerini ayırın
      X_train, X_valid, y_train, y_valid=train_test_split(X,y,train_size=0.8,test_size=0.2,
                                                         random_state=0)
```

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

```
[4]: X_train.head()
```

```
[4]:
```

|     | LotArea | YearBuilt | 1stFlrSF | 2ndFlrSF | FullBath | BedroomAbvGr | TotRmsAbvGrd |
|-----|---------|-----------|----------|----------|----------|--------------|--------------|
| Id  |         |           |          |          |          |              |              |
| 619 | 11694   | 2007      | 1828     | 0        | 2        | 3            | 9            |
| 871 | 6600    | 1962      | 894      | 0        | 1        | 2            | 5            |
| 93  | 13360   | 1921      | 964      | 0        | 1        | 2            | 5            |
| 818 | 13265   | 2002      | 1689     | 0        | 2        | 3            | 7            |
| 303 | 13704   | 2001      | 1541     | 0        | 2        | 3            | 6            |

```
[ ]:
```

### Step 1: birkaç modeli değerlendirin

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

Kod hücrelerini değişiklik yapmadan çalıştırın.

```
[5]: from sklearn.ensemble import RandomForestRegressor
# Modelleri tanımlayın
model1=RandomForestRegressor(n_estimators=50,random_state=0)
model2=RandomForestRegressor(n_estimators=100,random_state=0)
model3=RandomForestRegressor(n_estimators=100,criterion='mae',random_state=0)
model4=RandomForestRegressor(n_estimators=200,min_samples_split=20,random_state=0)
model5=RandomForestRegressor(n_estimators=100,max_depth=7,random_state=0)
models=[model1,model2,model3,model4,model5]
```

Buradaki parametrelere göz atalım;

#### ➤ **n\_estimators**-(integer)- Default=10

Tahminleri için, bir Rasgele Orman içinde inşa etmek istediğiniz ağaç sayısı. Sayı ne kadar yüksekse, kodunuzun çalışması için daha uzun süreceğini bilmek önemlidir. Bilgisayarınızın işlem hızıyla ilgili önceki bilgilere dayanarak, bu hızla orantılı bir **n\_estimator** oluşturmak gerekli.

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

#### ➤ **min\_samples\_split**-(integer, float)-Default=2

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

#### ➤ **max\_depth**-(integer or none)- Default=None

Bu, ağaçlarınızı ne kadar derin yapmak istediğinizi seçer. Max\_depth'inizi ayarlamanızı önerilir, çünkü overfitting baş etmek için önerilir.

- Ağacın kökü ve yapraklar arasındaki maksimum bağlantı sayısı. Küçük olmalı.

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'yi elde edeceğini hatırlayın. (Ortalama mutlak hatayı gözden geçirmek için buraya bakın .) <https://www.kaggle.com/dansbecker/model-validation>

Kod hücrelerini değişiklik yapmadan çalıştırın.

```
[9]: from sklearn.metrics import mean_absolute_error
# Farklı modelleri karşılaştırmak için function(fonksiyon)
def score_model(model,X_t=X_train,X_v=X_valid,y_t=y_train,y_v=y_valid):
    model.fit(X_t,y_t)# model fit ediliyor train(eğitim) verileri ile
    preds=model.predict(X_v)
    return mean_absolute_error(y_v,preds)
for i in range(0,len(models)):
    mae=score_model(models[i])
    print("Model {} MAE: {}".format(i+1,mae))

Model 1 MAE: 24015.492818003917
Model 2 MAE: 23740.979228636657
Model 3 MAE: 23528.78421232877
Model 4 MAE: 23996.676789668687
Model 5 MAE: 23706.672864217904
```

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

```
[10]: # En iyi model ile doldurun
best_model=model3
```

### Step 2: Generate test predictions (Test tahminleri oluşturun)

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.

```
[11]: # Bir model tanımlayın
my_model=RandomForestRegressor(random_state=0)
```

Bir sonraki kod hücrelerini 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!

```
] : # Fit the model to the training data
my_model.fit(X,y)
# Generate test predictions
preds_test=my_model.predict(X_test)
output = pd.DataFrame({'Id': X_test.index,
                       'SalePrice': preds_test})
output.to_csv('submission.csv', index=False)
```



## 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

#### 1) [A Simple Option: Drop Columns with Missing Values \(basit bir seçenek: eksik değerlere sahip sütunları drop\(düşürmek\)\)](#)

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

| Bed | Bath |
|-----|------|
| 1.0 | 1.0  |
| 2.0 | 1.0  |
| 3.0 | 2.0  |
| NaN | 2.0  |



| Bath |
|------|
| 1.0  |
| 1.0  |
| 2.0  |
| 2.0  |

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

Aşırı bir örnek olarak, önemli bir sütunun tek bir girişi eksik olduğu 10.000 satır içeren bir veri kümesini düşünün.


Bu yaklaşım sütunu tamamen düşürecektir!

#### 2) [A Better Option: Imputation \(Daha iyi bir seçenek\)](#)

Imputation, eksik değerleri bazı sayılarla doldurur.

Örneğin, her sütun boyunca ortalama değeri doldurabiliriz.

| Bed | Bath |
|-----|------|
| 1.0 | 1.0  |
| 2.0 | 1.0  |
| 3.0 | 2.0  |
| NaN | 2.0  |



| Bed | Bath |
|-----|------|
| 1.0 | 1.0  |
| 2.0 | 1.0  |
| 3.0 | 2.0  |
| 2.0 | 2.0  |



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

### 3) *An Extension To Imputation*

Imputation 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.



The diagram illustrates the process of adding a binary indicator column to a dataset with missing values. On the left, a table with columns 'Bed' and 'Bath' is shown. The 'Bed' column has values 1.0, 2.0, 3.0, and NaN. The 'Bath' column has values 1.0, 1.0, 2.0, and 2.0. A blue arrow points to the right, where the same data is shown with an additional column, 'Bed\_was\_missing'. This column contains 'FALSE' for the first three rows and 'TRUE' for the row where 'Bed' was NaN.

| Bed | Bath |
|-----|------|
| 1.0 | 1.0  |
| 2.0 | 1.0  |
| 3.0 | 2.0  |
| NaN | 2.0  |

| Bed | Bath | Bed_was_missing |
|-----|------|-----------------|
| 1.0 | 1.0  | FALSE           |
| 2.0 | 1.0  | FALSE           |
| 3.0 | 2.0  | FALSE           |
| 2.0 | 2.0  | TRUE            |

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](#) 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.

```
In [2]:  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean_absolute_error  
  
# Function for comparing different approaches  
def score_dataset(X_train, X_valid, y_train, y_valid):  
    model = RandomForestRegressor(n_estimators=10, random_state=0)  
    model.fit(X_train, y_train)  
    preds = model.predict(X_valid)  
    return mean_absolute_error(y_valid, preds)
```

### 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.

```
# Eksik değerlere sahip sütunların adlarını alın
cols_with_missing=[col for col in X_train.columns
                    if X_train[col].isnull().any()]
# Training ve validation verilerinde sütunları düşür
reduced_X_train=X_train.drop(cols_with_missing,axis=1)
reduced_X_valid=X_valid.drop(cols_with_missing,axis=1)
print(score_dataset(reduced_X_train,reduced_X_valid,y_train,y_valid))
```

MAE from Approach 1 (Drop columns with missing values):  
183550.22137772635

### 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

```
: from sklearn.impute import SimpleImputer
#Imputation
my_imputer=SimpleImputer()
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_valid = pd.DataFrame(my_imputer.transform(X_valid))
# Imputation removed column names; put them back
imputed_X_train.columns = X_train.columns
imputed_X_valid.columns = X_valid.columns
print(score_dataset(imputed_X_train, imputed_X_valid,y_train,y_valid))
```

- **transform**: Veriyle ilgili yapacağımız dönüştürme işlemlerinde ise **transform** fonksiyonunu kullanacağız. Dönüştürme işlemleri eksik veriyi doldurma, veriyi ölçeklendirme gibi alanlarda karşımıza çıkıyor. Aynı zamanda bir matrisi çarpanlarına ayırmak gibi işlemler için de **transform** fonksiyonu kullanılıyor.

Eksik değerleri doldurmak için aşağıdaki satırı kullanabiliriz:

```
X_test = mean_imputer.transform(X_test)
```

Burada **fit** metodu, eğitim veri setine uygulandığında, model parametrelerini öğrenir (örneğin, ortalama ve standart sapma). Daha sonra , dönüştürülmüş (ölçeklendirilmiş) eğitim veri setini elde etmek için **dönüşüm** yöntemini eğitim veri setine uygulamamız gerekir. Eğitim veri setinde **fit\_transform** uygulayarak bu adımların her ikisini de bir adımda gerçekleştirebiliriz .

MAE from Approach 2 (Imputation):  
178166.46269899711

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.

```
]# Original verileri değiştirmekten kaçınmak için kopya yapın ( imputing yaparken)
X_train_plus = X_train.copy()
X_valid_plus = X_valid.copy()
# Neyin dahil edileceğini gösteren yeni sütunlar oluşturun
for col in cols_with_missing:
    X_train_plus[col + '_was_missing'] = X_train_plus[col].isnull()
    X_valid_plus[col + '_was_missing'] = X_valid_plus[col].isnull()
# Imputation
my_imputer = SimpleImputer()
imputed_X_train_plus = pd.DataFrame(my_imputer.fit_transform(X_train_plus))
imputed_X_valid_plus = pd.DataFrame(my_imputer.transform(X_valid_plus))
# Imputation sütun adlarını kaldırın; onları geri koy
imputed_X_train_plus.columns = X_train_plus.columns
imputed_X_valid_plus.columns = X_valid_plus.columns
print("MAE from Approach 3 (An Extension to Imputation):")
print(score_dataset(imputed_X_train_plus, imputed_X_valid_plus, y_train, y_valid))
```

```
# Eksik değerlere sahip sütunların adlarını alın
cols_with_missing=[col for col in X_train.columns
                    if X_train[col].sinull.any()]
```

MAE from Approach 3 (An Extension to Imputation):  
178927.583183954

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.

```
In [6]: # Shape of training data (num_rows, num_columns)
print(X_train.shape)

# Number of missing values in each column of training data
missing_val_count_by_column = (X_train.isnull().sum())
print(missing_val_count_by_column[missing_val_count_by_column > 0])
```

```
(10864, 12)
Car          49
BuildingArea 5156
YearBuilt    4307
dtype: int64
```

### 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 alıştırma, [Housing Prices Competition for Kaggle Learn Users](#) verileri ile çalışacaksınız.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Verileri oku
X_full = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Eksik hedefi olan satırları kaldırın, hedefi belirleyicilerden ayırın
X_full.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X_full.SalePrice
X_full.drop(['SalePrice'], axis=1, inplace=True)

# İşleri basit tutmak için sadece sayısal tahminicileri kullanacağız
X = X_full.select_dtypes(exclude=['object'])
X_test = X_test_full.select_dtypes(exclude=['object'])

# Break off validation set from training data
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,
                                                    random_state=0)
```

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

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

▶ X\_train.head()

Out[2]:

|     | MSSubClass | LotFrontage | LotArea | OverallQual | OverallCond | YearBuilt | YearRemodAdd | MasVnrArea | BsmtFinSF1 | BsmtFinSF2 | ... | GarageArea | Woc |
|-----|------------|-------------|---------|-------------|-------------|-----------|--------------|------------|------------|------------|-----|------------|-----|
| Id  |            |             |         |             |             |           |              |            |            |            |     |            |     |
| 619 | 20         | 90.0        | 11694   | 9           | 5           | 2007      | 2007         | 452.0      | 48         | 0          | ... | 774        |     |
| 871 | 20         | 60.0        | 6600    | 5           | 5           | 1962      | 1962         | 0.0        | 0          | 0          | ... | 308        |     |
| 93  | 30         | 80.0        | 13360   | 5           | 7           | 1921      | 2006         | 0.0        | 713        | 0          | ... | 432        |     |
| 818 | 20         | NaN         | 13265   | 8           | 5           | 2002      | 2002         | 148.0      | 1218       | 0          | ... | 857        |     |
| 303 | 20         | 118.0       | 13704   | 7           | 5           | 2001      | 2002         | 150.0      | 0          | 0          | ... | 843        |     |

5 rows × 36 columns

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 hücrelerini herhangi bir değişiklik yapmadan çalıştırın.

```
# Shape of training data (num_rows, num_columns)
print(X_train.shape)

# Eğitim verilerinin her sütununda eksik değerlerin sayısı
missing_val_count_by_column = (X_train.isnull().sum())
print(missing_val_count_by_column[missing_val_count_by_column > 0])
```

```
(1168, 36)
LotFrontage    212
MasVnrArea      6
GarageYrBlt    58
dtype: int64
```

#### Part A

```
# Aşağıdaki satırı doldurun: eğitim verilerinde kaç satır var?
num_rows = 1168

# Aşağıdaki satırı doldurun: eğitim verisinde kaç sütun var
# eksik değerler var mı ?
num_cols_with_missing = 3

# Aşağıdaki satırı doldurun: kaç eksik giriş bulunur
# tüm eğitim verileri?
tot_missing = 276

# Check your answers
step_1.a.check()
```

#### 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.

```
[13]: from sklearn.ensemble import RandomForestRegressor
      from sklearn.metrics import mean_absolute_error

      # Farklı yaklaşımları karşılaştırmak için function
      def score_dataset(X_train, X_valid, y_train, y_valid):
          model = RandomForestRegressor(n_estimators=100, random_state=0)
          model.fit(X_train, y_train)
          preds = model.predict(X_valid)
          return mean_absolute_error(y_valid, preds)
```

### 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.

```
[17]: # Aşağıdaki satırı doldurun: eksik değerlere sahip sütunların adlarını alın
cols_with_missing=[col for col in X_train.columns if X_train[col].isnull().any()]# Kodunuz burada

# Fill in the lines below: drop columns in training and validation data
reduced_X_train = X_train.drop(cols_with_missing,axis=1)
reduced_X_valid = X_valid.drop(cols_with_missing,axis=1)

# Check your answers
step_2.check()
```

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

```
[18]: print("MAE (Drop columns with missing values):")
print(score_dataset(reduced_X_train, reduced_X_valid, y_train, y_valid))
```

```
MAE (Drop columns with missing values):
17837.82570776256
```

### 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.

```
9]: from sklearn.impute import SimpleImputer

# Aşağıdaki satırları doldurun: imputation
my_imputer=SimpleImputer() # Your code here
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_valid = pd.DataFrame(my_imputer.transform(X_valid))

# Fill in the lines below: imputation removed column names; put them back
imputed_X_train.columns = X_train.columns
imputed_X_valid.columns = X_valid.columns

# Check your answers
step_3.a.check()
```

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

```
[20]: print("MAE (Imputation):")
print(score_dataset(imputed_X_train, imputed_X_valid, y_train, y_valid))
```

```
MAE (Imputation):
18062.894611872147
```

#### Part B

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

İpucu: Kayıp değerlerin kaldırılması, imputasyondan 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 değer 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 eğit ve değerlendirirsiniz. Ardından, yarışmaya sunulabilecek tahminler oluşturmada önce test verilerini önceden işlersiniz!

#### Part A

Training ve validation verilerini önceden işlemek için sonraki kod hücrelerini 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.

```
[22]: # Preprocessed training and validation features
final_X_train = reduced_X_train
final_X_valid = reduced_X_valid
#eksik değerler içeren sütunları drop işlemine tabi tuttuğumuz durumu seçtik
# Check your answers
step_4.a.check()
```

Correct

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



```
# Modeli tanımlayın ve fit edin
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(final_X_train, y_train)

# Doğrulama tahminleri ve MAE alın
preds_valid = model.predict(final_X_valid)
print("MAE (Your approach):")
print(mean_absolute_error(y_valid, preds_valid))
```

```
MAE (Your approach):
17837.82570776256
```

## Part B

Test verilerinizi önceden işlemek için bir sonraki kod hücrelerini 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 eğitilmiş modeli kullanın.

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


```
#X_train'den drop ettiğimiz verileri X_test'den de düşürmeliyiz
final_X_test = X_test.drop(cols_with_missing,axis=1)
#X_test içerisinde hala eksik deger içeren kolonlar mevcut
#Çünkü train verisinde boş olmayan ama test verisinde boş olan satırlardan kaynaklı
#test verisinde eksik deger bulunan column'ları bulalım
final_miss=[col for col in final_X_test.columns if final_X_test[col].isnull().any()]
#eksik degerleri drop etmiyoruz cunku X_train ile column'lara sahip olmalılar
# eksik degerleri ortalama degerler ile dolduruyoruz
final_X_test=pd.DataFrame(my_imputer.fit_transform(final_X_test))
preds_test = model.predict(final_X_test)

step_4.b.check()
```

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

```
[ ]: # Save test predictions to file
output = pd.DataFrame({'Id': X_test.index,
                       'SalePrice': preds_test})
output.to_csv('submission.csv', index=False)
```



 InClass Prediction Competition

## Housing Prices Competition for Kaggle Learn Users

Apply what you learned in the Machine Learning course on Kaggle Learn alongside others in the course.

39,301 teams · 9 years to go

[Overview](#) [Data](#) [Notebooks](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#) [My Submissions](#) [Submit Predictions](#)

Your most recent submission

| Name           | Submitted      | Wait time | Execution time | Score       |
|----------------|----------------|-----------|----------------|-------------|
| submission.csv | 20 minutes ago | 0 seconds | 0 seconds      | 16592.77974 |

Complete

[Jump to your position on the leaderboard](#)

## 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 karşılaştıracacağız.

### Üç Yaklaşım

#### 1) [Drop Categorical Variables](#)

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

#### 2) [Label Encoding](#)

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

| Breakfast | Breakfast |
|-----------|-----------|
| Every day | 3         |
| Never     | 0         |
| Rarely    | 1         |
| Most days | 2         |
| Never     | 0         |

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

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

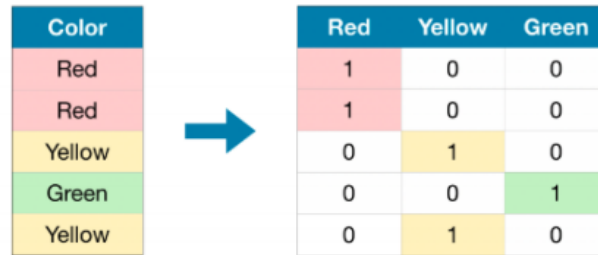
Tüm kategorik değişkenlerin değerlerde açık bir sırası yoktur, ancak **ordinal**(sıralı) değişkenler olarak adlandırılanlara atıfta bulunuruz.

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

### 3) One-Hot Encoding

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

Bunu anlamak için bir örnek üzerinde çalışacağız.



The diagram illustrates the one-hot encoding process. On the left, a table with a single column 'Color' contains five rows: 'Red', 'Red', 'Yellow', 'Green', and 'Yellow'. A blue arrow points to the right, where a new table is shown with three columns: 'Red', 'Yellow', and 'Green'. Each row in the new table corresponds to a row in the original table, with a '1' in the column corresponding to the color and '0' in the others.

| Color  | Red | Yellow | Green |
|--------|-----|--------|-------|
| Red    | 1   | 0      | 0     |
| Red    | 1   | 0      | 0     |
| Yellow | 0   | 1      | 0     |
| Green  | 0   | 0      | 1     |
| Yellow | 0   | 1      | 0     |

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

İşsel 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](#) 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.

```

import pandas as pd
from sklearn.model_selection import train_test_split
# Verileri okuyun
data=pd.read_csv('melb_data.csv')
# Hedefi tahmin ediciden ayırın
y=data.Price
X=data.drop(['Price'],axis=1)
# Verileri training ve validation alt kümelerine bölün
X_train_full,X_valid_full,y_train,y_valid=train_test_split(X,y,train_size=0.8,test_size=0.2, random_state=0)
# Eksik değerlere sahip sütunları drop edin (en basit yaklaşım)
cols_with_missing=[col for col in X_train_full.columns if X_train_full[col].isnull().any()]
X_train_full.drop(cols_with_missing, axis=1,inplace=True)
X_valid_full.drop(cols_with_missing, axis=1, inplace=True)
# "Cardinality", bir sütundaki benzersiz değerlerin sayısı anlamına gelir
# Nispeten düşük kardinaliteye sahip kategorik sütunlar seçin (uygun ancak keyfi)
low_cardinality_cols = [cname for cname in X_train_full.columns if X_train_full[cname].nunique() < 10 and
                        X_train_full[cname].dtype == "object"]
# Select numerical columns
numerical_cols = [cname for cname in X_train_full.columns if X_train_full[cname].dtype in ['int64', 'float64']]
# Sadece seçili sütunları tut
my_cols = low_cardinality_cols + numerical_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()

```

[7]: X\_train.head()

|       | Type | Method | Regionname            | Rooms | Distance | Postcode | Bedroom2 | Bathroom | Landsize | Lattitude | Longitude | Propertycount |
|-------|------|--------|-----------------------|-------|----------|----------|----------|----------|----------|-----------|-----------|---------------|
| 12167 | u    | S      | Southern Metropolitan | 1     | 5.0      | 3182.0   | 1.0      | 1.0      | 0.0      | -37.85984 | 144.9867  | 13240.0       |
| 6524  | h    | SA     | Western Metropolitan  | 2     | 8.0      | 3016.0   | 2.0      | 2.0      | 193.0    | -37.85800 | 144.9005  | 6380.0        |
| 8413  | h    | S      | Western Metropolitan  | 3     | 12.6     | 3020.0   | 3.0      | 1.0      | 555.0    | -37.79880 | 144.8220  | 3755.0        |
| 2919  | u    | SP     | Northern Metropolitan | 3     | 13.0     | 3046.0   | 3.0      | 1.0      | 265.0    | -37.70830 | 144.9158  | 8870.0        |
| 6043  | h    | S      | Western Metropolitan  | 3     | 13.3     | 3020.0   | 3.0      | 1.0      | 673.0    | -37.76230 | 144.8272  | 4217.0        |

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.

```

[8]: # Kategorik değişkenlerin listesini alın
s = (X_train.dtypes == 'object')
object_cols = list(s[s].index)

print("Categorical variables:")
print(object_cols)

Categorical variables:
['Type', 'Method', 'Regionname']

```

### 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!

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Function for comparing different approaches
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)

```

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

Object sütunlarını select\_dtypes () yöntemiyle düşürürüz.

```

drop_X_train = X_train.select_dtypes(exclude=['object'])
drop_X_valid = X_valid.select_dtypes(exclude=['object'])

print("MAE from Approach 1 (Drop categorical variables):")
print(score_dataset(drop_X_train, drop_X_valid, y_train, y_valid))

```

```

MAE from Approach 1 (Drop categorical variables):
175703.48185157913

```

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

```

from sklearn.preprocessing import LabelEncoder
# Orijinal verileri değiştirmekten kaçınmak için kopya yapın
label_X_train = X_train.copy()
label_X_valid = X_valid.copy()
# Her sütuna kategorik verilerle etiket kodlayıcı uygulayın
label_encoder = LabelEncoder()
for col in object_cols:
    label_X_train[col] = label_encoder.fit_transform(X_train[col])
    label_X_valid[col] = label_encoder.transform(X_valid[col])
print("MAE from Approach 2 (Label Encoding):")
print(score_dataset(label_X_train, label_X_valid, y_train, y_valid))

```

```

MAE from Approach 2 (Label Encoding):
165936.40548390493

```

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

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

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

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

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

```
from sklearn.preprocessing import OneHotEncoder

# Apply one-hot encoder to each column with categorical data
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(X_train[object_cols]))
OH_cols_valid = pd.DataFrame(OH_encoder.transform(X_valid[object_cols]))

# One-hot encoding removed index; put it back
OH_cols_train.index = X_train.index
OH_cols_valid.index = X_valid.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train = X_train.drop(object_cols, axis=1)
num_X_valid = X_valid.drop(object_cols, axis=1)

# Add one-hot encoded columns to numerical features
OH_X_train = pd.concat([num_X_train, OH_cols_train], axis=1)
OH_X_valid = pd.concat([num_X_valid, OH_cols_valid], axis=1)

print("MAE from Approach 3 (One-Hot Encoding):")
print(score_dataset(OH_X_train, OH_X_valid, y_train, y_valid))
```

```
MAE from Approach 3 (One-Hot Encoding):
166089.4893009678
```

### 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 farkı 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](#) ile çalışacağız.



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

```
[1]: import pandas as pd
      from sklearn.model_selection import train_test_split

      # Verileri oku
      X = pd.read_csv('../input/train.csv', index_col='Id')
      X_test = pd.read_csv('../input/test.csv', index_col='Id')

      # Eksik hedefi olan satırları kaldırın, hedefi belirleyicilerden ayırın
      X.dropna(axis=0, subset=['SalePrice'], inplace=True)
      y = X.SalePrice
      X.drop(['SalePrice'], axis=1, inplace=True)

      # İşleri basit tutmak için, eksik değerlere sahip sütunları drop edeceğiz.
      cols_with_missing = [col for col in X.columns if X[col].isnull().any()]
      X.drop(cols_with_missing, axis=1, inplace=True)
      X_test.drop(cols_with_missing, axis=1, inplace=True)

      # Break off validation set from training data
      X_train, X_valid, y_train, y_valid = train_test_split(X, y,
                                                            train_size=0.8, test_size=0.2,
                                                            random_state=0)
```



X\_train.head()

Out[4]:

|                     | MSSubClass | MSZoning | LotArea | Street | LotShape | LandContour | Utilities | LotConfig | LandSlope | Neighborhood | ... |
|---------------------|------------|----------|---------|--------|----------|-------------|-----------|-----------|-----------|--------------|-----|
| Id                  |            |          |         |        |          |             |           |           |           |              |     |
| 619                 | 20         | RL       | 11694   | Pave   | Reg      | Lvl         | AllPub    | Inside    | Gtl       | NridgHt      | ... |
| 871                 | 20         | RL       | 6600    | Pave   | Reg      | Lvl         | AllPub    | Inside    | Gtl       | NAmes        | ... |
| 93                  | 30         | RL       | 13360   | Pave   | IR1      | HLS         | AllPub    | Inside    | Gtl       | Crawfor      | ... |
| 818                 | 20         | RL       | 13265   | Pave   | IR1      | Lvl         | AllPub    | CulDSac   | Gtl       | Mitchel      | ... |
| 303                 | 20         | RL       | 13704   | Pave   | IR1      | Lvl         | AllPub    | Corner    | Gtl       | CollgCr      | ... |
| 5 rows x 60 columns |            |          |         |        |          |             |           |           |           |              |     |

| ... | OpenPorchSF | EnclosedPorch | 3SsnPorch | ScreenPorch | PoolArea | MiscVal | MoSold | YrSold | SaleType | SaleCondition |
|-----|-------------|---------------|-----------|-------------|----------|---------|--------|--------|----------|---------------|
| ... | 108         | 0             | 0         | 260         | 0        | 0       | 7      | 2007   | New      | Partial       |
| ... | 0           | 0             | 0         | 0           | 0        | 0       | 8      | 2009   | WD       | Normal        |
| ... | 0           | 44            | 0         | 0           | 0        | 0       | 8      | 2009   | WD       | Normal        |
| ... | 59          | 0             | 0         | 0           | 0        | 0       | 7      | 2008   | WD       | Normal        |
| ... | 81          | 0             | 0         | 0           | 0        | 0       | 1      | 2006   | WD       | Normal        |

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.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# farklı yaklaşımları karşılaştırmak için fonksiyon
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)
```

### 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 hücrelerini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla `drop_X_train` ve `drop_X_valid` olarak ayarlayın.

```
[13]: # Aşağıdaki satırları doldurun: training ve validation verilerinde sütunları drop edin.
drop_X_train = X_train.select_dtypes(exclude=["object"])
drop_X_valid = X_valid.select_dtypes(exclude=["object"])

# Check your answers
step_1.check()
```

Bu yaklaşım için MAE hesaplayalım.

```
MAE from Approach 1 (Drop categorical variables):
17837.82570776256
```

### 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.

```
[15]: print("Unique values in 'Condition2' column in training data:", X_train['Condition2'].unique())
print("\nUnique values in 'Condition2' column in validation data:", X_valid['Condition2'].unique())
```

```
Unique values in 'Condition2' column in training data: ['Norm' 'PosA' 'Feedr' 'PosN' 'Artery' 'RRae']
Unique values in 'Condition2' column in validation data: ['Norm' 'RRan' 'RRNn' 'Artery' 'Feedr' 'PosN']
```

+ Code

+ Markdown



Şimdi buna göre kod yazarsanız:

- label encoder'ı 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 karşılaşılabilecek 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 hücrelerini çalıştırın. Benzer şekilde, güvenli bir şekilde etiketlenebilen sütunlar good\_label\_cols içinde saklanır.

```
[17]: # Tüm kategorik sütunlar
object_cols = [col for col in X_train.columns if X_train[col].dtype == "object"]

# Columns that can be safely label encoded(# Güvenli bir şekilde etiketlenebilen sütunlar kodlanmış)
good_label_cols = [col for col in object_cols if
                    set(X_train[col]) == set(X_valid[col])]

# Veri kümesinden atılacak sorunlu sütunlar
bad_label_cols = list(set(object_cols)-set(good_label_cols))

print('Categorical columns that will be label encoded:', good_label_cols)
print('\nCategorical columns that will be dropped from the dataset:', bad_label_cols)
```

```
Categorical columns that will be label encoded: ['MSZoning', 'Street', 'LotShape', 'LandContour', 'LotConfig', 'BldgType',
'HouseStyle', 'ExterQual', 'CentralAir', 'KitchenQual', 'PavedDrive', 'SaleCondition']

Categorical columns that will be dropped from the dataset: ['Condition2', 'LandSlope', 'SaleType', 'Exterior2nd', 'ExterCond',
'HeatingQC', 'Neighborhood', 'RoofStyle', 'Exterior1st', 'RoofMatl', 'Condition1', 'Utilities', 'Foundation', 'Heating', 'Functional']
```

X\_train ve X\_valid içindeki verilere label encode yapmak için sonraki kod hücrelerini 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.



```
[19]: from sklearn.preprocessing import LabelEncoder

# Kodlanmayacak kategorik sütunları drop et
label_X_train = X_train.drop(bad_label_cols, axis=1)
label_X_valid = X_valid.drop(bad_label_cols, axis=1)

# Apply label encoder
label_encoder=LabelEncoder()# Your code here
for col in good_label_cols:
    label_X_train[col]=label_encoder.fit_transform(X_train[col])
    label_X_valid[col]=label_encoder.transform(X_valid[col])

# Check your answer
step_2.b.check()
```

Correct

```
print("MAE from Approach 2 (Label Encoding):")
print(score_dataset(label_X_train, label_X_valid, y_train, y_valid))
```

```
MAE from Approach 2 (Label Encoding):
17575.291883561644
```

### 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 hücreğini değişiklik olmadan çalıştırarak başlayın.

```
# Kategorik verilerle her sütundaki benzersiz girdilerin sayısını alın
object_nunique = list(map(lambda col: X_train[col].nunique(), object_cols))
d = dict(zip(object_cols, object_nunique))

# Benzersiz girişlerin sayısını sütuna göre artan sırada yazdırın
sorted(d.items(), key=lambda x: x[1])
```

```
[('Street', 2),
 ('Utilities', 2),
 ('CentralAir', 2),
 ('LandSlope', 3),
 ('PavedDrive', 3),
 ('LotShape', 4),
 ('LandContour', 4),
 ('ExterQual', 4),
 ('KitchenQual', 4),
 ('MSZoning', 5),
 ('LotConfig', 5),
 ('BldgType', 5),
 ('ExterCond', 5),
 ('HeatingQC', 5),
 ('Condition2', 6),
 ('RoofStyle', 6),
 ('Foundation', 6),
 ('Heating', 6),
 ('Functional', 6),
 ('SaleCondition', 6),
 ('RoofMatl', 7),
 ('HouseStyle', 8),
 ('Condition1', 9),
 ('SaleType', 9),
 ('Exterior1st', 15),
 ('Exterior2nd', 16),
 ('Neighborhood', 25)]
```

Zip():

```
1 >>>
2 >>> list(zip([1, 2, 3, 4], "pow"))
3 [(1, 'p'), (2, 'o'), (3, 'w')]
4 >>>
```

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.

```
[22]: # Aşağıdaki satırı doldurun: Egzersiz verilerinde kaç kategorik değişken
# cardinality 10'dan büyük mü ?
high_cardinality_numcols = 3

# Aşağıdaki satırı doldurun: Tek bir etkin kodlama için kaç sütun gerekli
# Eğitim verilerinde 'Neighborhood' değişkeni?
num_cols_neighborhood = 25

# Check your answers
step_3.a.check()
```

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.

```
3): # Aşağıdaki satırı doldurun: Veri kümesine kaç giriş eklendi:
# replacing the column with a one-hot encoding?(# sütunu tek yeni kodlamayla değiştirmek mi?)
OH_entries_added = 1e4*100-1e4

# Aşağıdaki satırı doldurun: Veri kümesine kaç giriş eklendi:
# sütunu bir etiket kodlaması ile değiştirme?
label_entries_added = 0

# Check your answers
step_3.b.check()
```

Correct

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 sayısı ile 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 hücreğini değiştirmeden çalıştırın. Benzer şekilde, high\_cardinality\_cols, veri kümesinden bırakılacak kategorik sütunların bir listesini içerir.

```
[24]: # Columns that will be one-hot encoded
low_cardinality_cols = [col for col in object_cols if X_train[col].nunique() < 10]

# Columns that will be dropped from the dataset
high_cardinality_cols = list(set(object_cols)-set(low_cardinality_cols))

print('Categorical columns that will be one-hot encoded:', low_cardinality_cols)
print('\nCategorical columns that will be dropped from the dataset:', high_cardinality_cols)
```

```
Categorical columns that will be one-hot encoded: ['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'ExterQual', 'ExterCond', 'Foundation', 'Heating', 'HeatingQC', 'CentralAir', 'KitchenQual', 'Functional', 'PavedDrive', 'SaleType', 'SaleCondition']
Categorical columns that will be dropped from the dataset: ['Exterior2nd', 'Neighborhood', 'Exterior1st']
```

X\_train ve X\_valid içindeki verilere one-hot encoding yapmak için sonraki kod hücreğini 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.

```
from sklearn.preprocessing import OneHotEncoder

# Apply one-hot encoder to each column with categorical data
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(X_train[low_cardinality_cols]))
OH_cols_valid = pd.DataFrame(OH_encoder.transform(X_valid[low_cardinality_cols]))

# One-hot encoding removed index; put it back
OH_cols_train.index = X_train.index
OH_cols_valid.index = X_valid.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train = X_train.drop(object_cols, axis=1)
num_X_valid = X_valid.drop(object_cols, axis=1)

# Add one-hot encoded columns to numerical features
OH_X_train = pd.concat([num_X_train, OH_cols_train], axis=1)
OH_X_valid = pd.concat([num_X_valid, OH_cols_valid], axis=1)

# Check your answer
step_4.check()
```

```
print("MAE from Approach 3 (One-Hot Encoding):")
print(score_dataset(OH_X_train, OH_X_valid, y_train, y_valid))
```

```
MAE from Approach 3 (One-Hot Encoding):
17525.345719178084
```

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

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





## **KAYNAKLAR**

- Kaggle – Intro to Machine Learning 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://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/>

