İçindekiler

Intro to Machine Learning

Makine öğrenmesindeki temel fikirleri öğrenin ve ilk modellerinizi oluşturun.

How Models Work (Modeller Nasıl Çalışır?)

Giriş

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

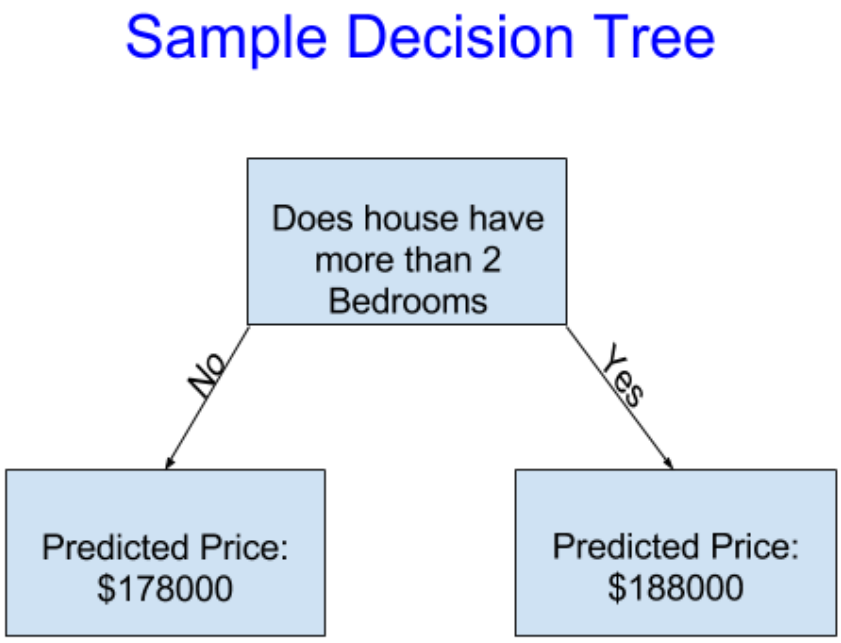
Bu mikro kurs, aşağıdaki senaryodan geçerken modeller oluşturmanızı sağlayacaktır:

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

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

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

Basitlik için, mümkün olan en basit karar ağacıyla başlayacağız.



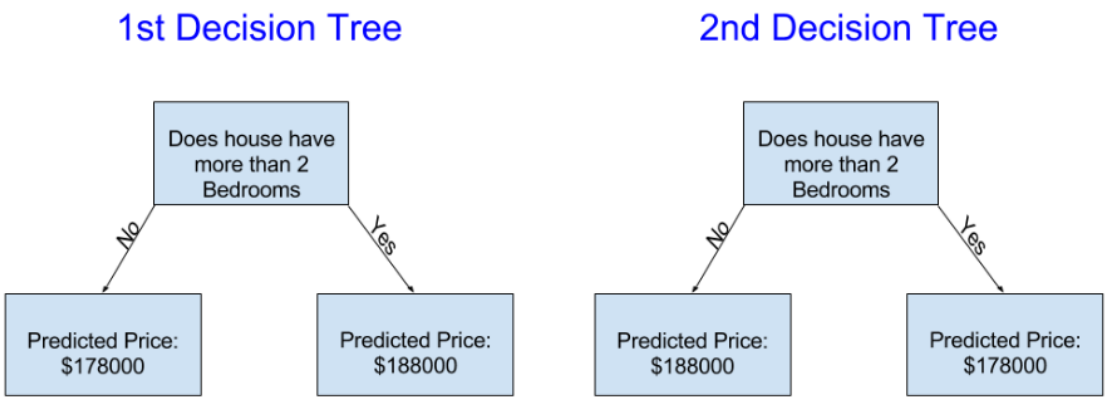
Evleri sadece iki kategoriye ayırır. Dikkate alınan herhangi bir ev için tahmini fiyat, aynı kategorideki evlerin tarihsel ortalama fiyatıdır.

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

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

Decision Tree'nin Geliştirilmesi

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



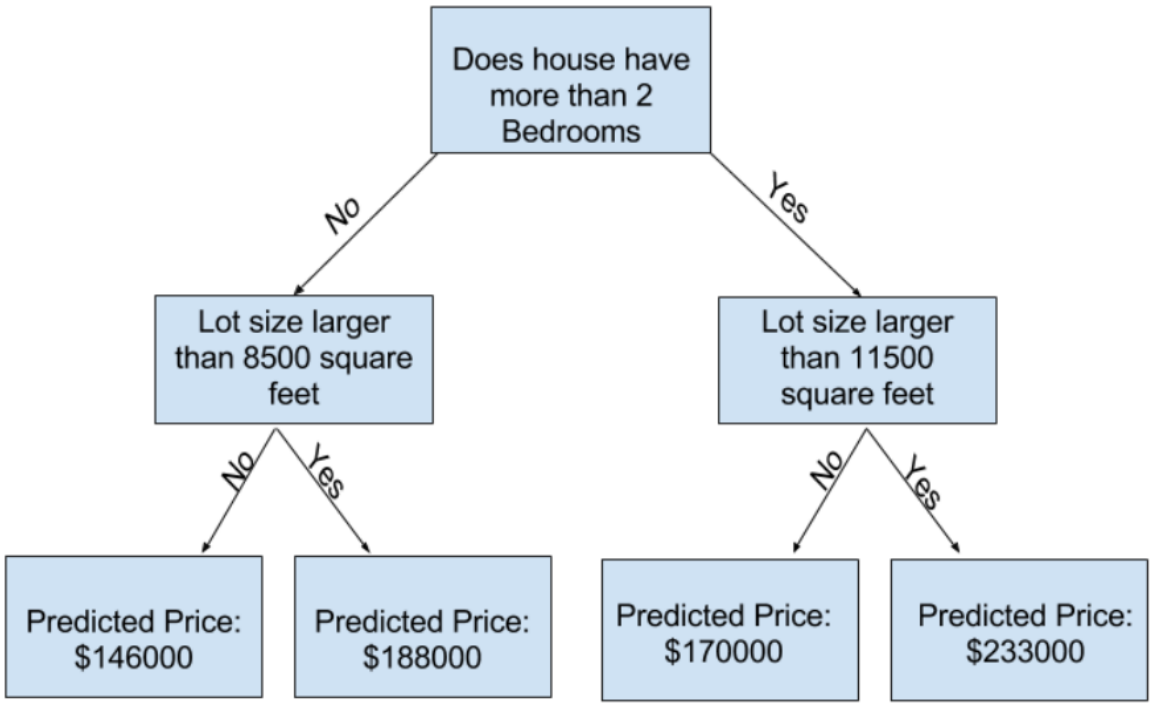
Soldaki karar ağacı (Decision Tree 1) muhtemelen daha mantıklıdır, çünkü daha fazla yatak odası olan evlerin daha az yatak odası olan evlerden daha yüksek fiyatlarla satılma eğiliminde olduğu gerçeğini yakalar.

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

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

Bunlara "deeper(daha derin)" ağaçlar denir.

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



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

Ev için tahmini fiyat ağacın altındadır.

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

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

Basic Data Exploration (Basit Veri Keşfi)

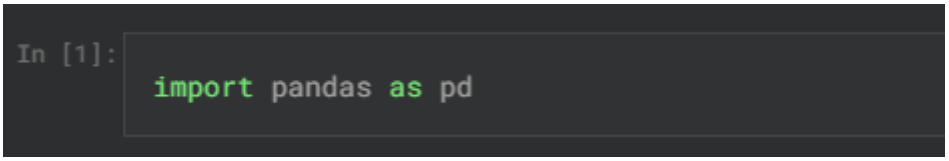
Verilerinizi Tanımak için Pandas Kullanımı

Herhangi bir makine öğrenimi projesinin ilk adımı, verileri tanımaktır.

Bunun için Pandas kütüphanesini kullanacaksınız.

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

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



Pandas kütüphanesinin en önemli kısmı DataFrame'dir.

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

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

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

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

Örnek (Melbourne) verileri ../input/melbourne-housing-snapshot/melb_data.csv dosya yolundadır.

Verileri aşağıdaki komutlarla yükler ve keşfederiz:

In [2]:

```
# save filepath to variable for easier access
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
# read the data and store data in DataFrame titled melbourne_data
melbourne_data = pd.read_csv(melbourne_file_path)
# print a summary of the data in Melbourne data
melbourne_data.describe()
```

Out[2]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000

Out[2]:

om	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
.000000	13518.000000	13580.000000	7130.000000	8205.000000	13580.000000	13580.000000	13580.000000
242	1.610075	558.416127	151.967650	1964.684217	-37.809203	144.995216	7454.417378
712	0.962634	3990.669241	541.014538	37.273762	0.079260	0.103916	4378.581772
000	0.000000	0.000000	0.000000	1196.000000	-38.182550	144.431810	249.000000
000	1.000000	177.000000	93.000000	1940.000000	-37.856822	144.929600	4380.000000
000	2.000000	440.000000	126.000000	1970.000000	-37.802355	145.000100	6555.000000
000	2.000000	651.000000	174.000000	1999.000000	-37.756400	145.058305	10331.000000
000	10.000000	433014.000000	44515.000000	2018.000000	-37.408530	145.526350	21650.000000

Interpreting Data Description (Verilerin Yorumlanması)

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

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

Eksik değerler birçok nedenden dolayı ortaya çıkar.

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

Eksik veriler konusuna geri döneceğiz.

İkinci değer, **mean** olan ortalamadır.

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

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

İlk (en küçük) değer min.

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

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

Excercise: Explore Your Data

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

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

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

Step 1: Loading Data (Veri Yükleme)

Iowa veri dosyasını home_data adlı bir Pandas DataFrame'de okuyun.

```
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 × 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 çıkılamaz veri miktarını anlayabileceğiniz bir şeye nasıl ayırabilirsiniz?

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

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

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

melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
melbourne_data.columns
```

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

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

Daha sonraki bir derste eksik değerleri ele almayı öğreneceğiz.

Iowa verileriniz, kullandığınız sütunlarda eksik değerlere sahip değildi.

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

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

In [2]:
```
# The Melbourne data has some missing values (some houses for which some variables weren't record
ed.)
# We'll learn to handle missing values in a later tutorial.
# Your Iowa data doesn't have missing values in the columns you use.
# So we will take the simplest option for now, and drop houses from our data.
# Don't worry about this much for now, though the code is:

# dropna drops missing values (think of na as "not available")
melbourne_data = melbourne_data.dropna(axis=0)
```

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.

```
In [3]: y = melbourne_data.Price
```

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:

```
In [4]: melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
```

Kural olarak, bu verilere X denir.

```
In [5]: X = melbourne_data[melbourne_features]
```

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

In [6]:
```
X.describe()
```

Out[6]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

In [7]:
```
X.head()
```

Out[7]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
1	2	1.0	156.0	-37.8079	144.9934
2	3	2.0	134.0	-37.8093	144.9944
4	4	1.0	120.0	-37.8072	144.9941
6	3	2.0	245.0	-37.8024	144.9993
7	2	1.0	256.0	-37.8060	144.9954

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

In [8]:
```
from sklearn.tree import DecisionTreeRegressor

# Define model. Specify a number for random_state to ensure same results each run
melbourne_model = DecisionTreeRegressor(random_state=1)

# Fit model
melbourne_model.fit(X, y)
```

Out[8]:
```
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=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.

Şimdi tahminler yapmak için kullanabileceğimiz uygun bir modelimiz var.

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

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

In [9]:
```
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
1      2       1.0     156.0   -37.8079    144.9934
2      3       2.0     134.0   -37.8093    144.9944
4      4       1.0     120.0   -37.8072    144.9941
6      3       2.0     245.0   -37.8024    144.9993
7      2       1.0     256.0   -37.8060    144.9954
The predictions are
[1035000. 1465000. 1600000. 1876000. 1636000.]
```

Exercise: Your First Machine Learning Model

Özet

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

```
# Code you have previously used to load data
import pandas as pd

# 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)

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex3 import *

print("Setup Complete")
```

```
Setup Complete
```

Exercises

Step 1: Prediction Target Belirleme

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

[8]:
```
# print the list of columns in the dataset to find the name of the prediction target
home_data.columns
```

Out[8]:
```
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')
```

Prediction Target'i y'ye tanımladık.

```
y = home_data.SalePrice

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

Correct

Step 2: X Oluştur

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

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

Listede yalnızca aşağıdaki sütunları kullanacaksınız :

```
* LotArea
* YearBuilt
* 1stFlrSF
* 2ndFlrSF
* FullBath
* BedroomAbvGr
* TotRmsAbvGrd
```

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

```
# 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()
```

Correct

Verinin İncelenmesi

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

```
# 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: Modelin belirlenmesi ve fit edilmesi

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

[27]:
```
from sklearn.tree import DecisionTreeRegressor
#specify the model.
#For model reproducibility, set a numeric value for random_state when specifying the model
iowa_model = DecisionTreeRegressor(random_state=7)

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

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

Correct

Step 4: Tahmin Yapma

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

[30]:
```
predictions = iowa_model.predict(X)
print(predictions)

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

```
[208500. 181500. 223500. ... 266500. 142125. 147500.]
```

Correct

+ Code + Markdown

[33]:
```
home_data.SalePrice.head()
```

Out[33]:
```
0    208500
1    181500
2    223500
3    140000
4    250000
Name: SalePrice, dtype: int64
```

Model Validation (Model Geçerliliği)

Bir model oluşturdunuz. Ama bu model ne kadar iyi?

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

Model Validation Nedir?

Oluşturduğunuz hemen hemen her modeli değerlendirmek isteyeceksiniz.

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

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

Birçok kişi, tahmin doğruluğunu ölçerken büyük bir hata yapar.

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

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

Önce model kalitesini anlaşılır bir şekilde özetlemeniz gerekir.

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

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

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

Son sözcükten başlayarak bu metriği inceleyelim, error.

Her ev için tahmin hatası:

```
error=actual-predicted
```

hata = gerçek değer – tahmin edilen değer

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

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

Daha sonra bu mutlak hataların ortalamasını alırız.

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

Ortalama olarak, tahminlerimiz yaklaşık X civarında.

MAE'yi hesaplamak için önce bir modele ihtiyacımız var.

In [1]:
```
# 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)
```

Out[1]:
```
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')
```

Bir modelimiz olduğunda, ortalama mutlak hatayı şu şekilde hesaplıyoruz:

In [2]:
```
from sklearn.metrics import mean_absolute_error

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

Out[2]:
```
434.71594577146544
```

The Problem with "In-Sample" Scores

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

Büyük emlak piyasasında kapı renginin ev fiyatıyla ilgisi olmadığını düşünün.

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

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

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

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

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

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

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

Coding It

Scikit-learn kütüphanesi, verileri iki parçaya bölmek için **train_test_split** fonksiyonuna sahiptir.

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

```
In [3]:
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)
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(train_X, train_y)

# get predicted prices on validation data
val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))
```

```
260991.8108457069
```

Wow!

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

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

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

Yani yeni verilerdeki hata ortalama ev değerinin dörtte biri kadardır.

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

Exercise: Model Validation

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

[1]:

```
# Code you have previously used to load data
import pandas as pd
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)
y = home_data.SalePrice
feature_columns = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[feature_columns]

# Specify Model
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())

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex4 import *
print("Setup Complete")
```

```
First in-sample predictions: [208500. 181500. 223500. 140000. 250000.]
Actual target values for those homes: [208500, 181500, 223500, 140000, 250000]
Setup Complete
```

Exercises

Step 1: Split Your Data (Verinizi Ayırın)

Verilerinizi bölmek için **train_test_split** işlevini kullanın.

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

[3]:
```
# 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 (Modeli belirleme ve fit etme)

DecisionTreeRegressor modeli oluşturun ve modeli ilgili veriler ile fit edin.

[5]:
```
# 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

[6]:
```
# Predict with all validation observations
val_predictions = iowa_model.predict(val_X)

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

Correct

Inspect your predictions and actual values from validation data.

+ Code + Markdown

[16]:
```
# print the top few validation predictions
print(val_predictions[:5],"\n")
# print the top few actual prices from validation data
print(val_y.head())
```

```
[186500. 184000. 130000.  92000. 164500.]

258     231500
267     179500
288     122000
649      84500
1233    142000
Name: SalePrice, dtype: int64
```

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

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

Step 4: Calculate the Mean Absolute Error in Validation Data

```
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()
```

```
29652.931506849316
```

Correct

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

Underfitting and Overfitting

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

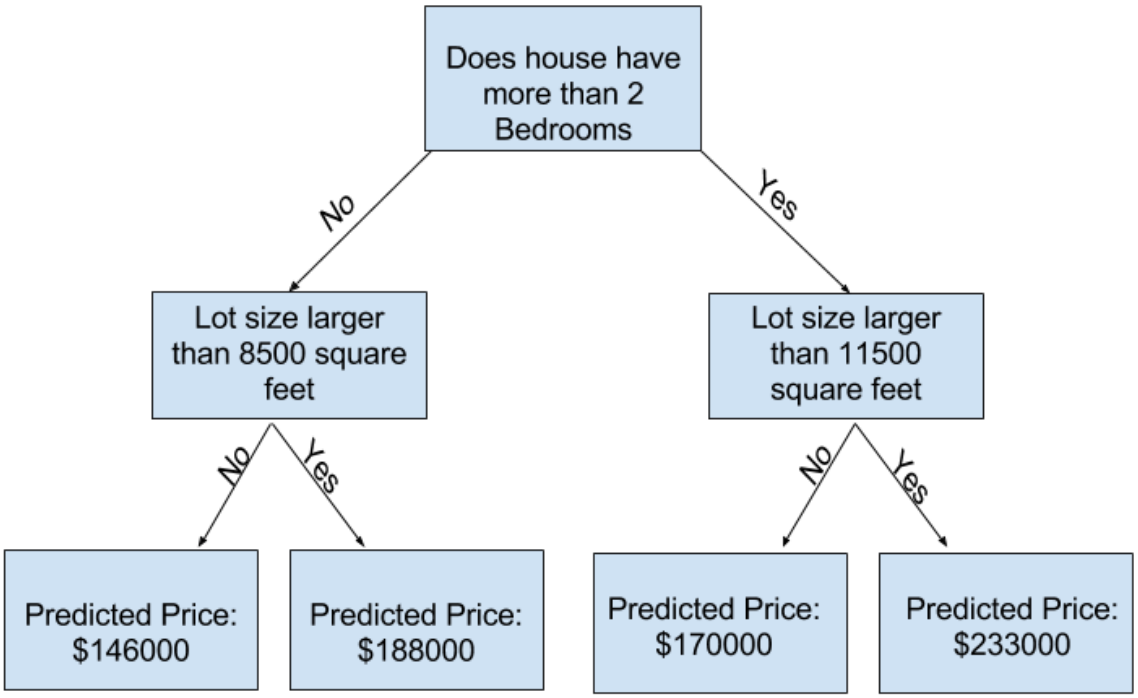
Farklı Modellerle Deneme

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

Peki modeller için hangi alternatifleriniz var?

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

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



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

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

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

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

Her seviyede daha fazla bölme ekleyerek grup sayısını ikiye katlamaya devam edersek, 10. seviyeye ulaştığımızda 2^{10} ev grubumuz olacak. Bu 1024 yaprak yapar.

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

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

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

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

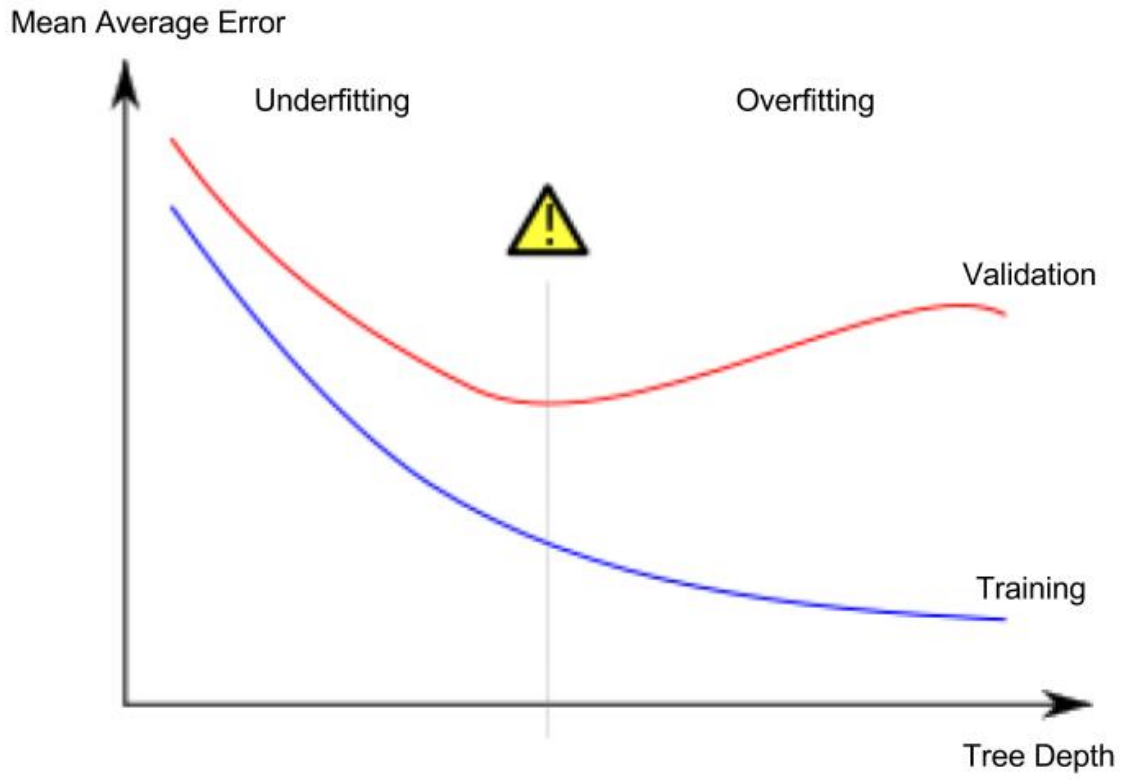
Extreme olarak, bir ağaç evleri sadece 2 veya 4'e ayırırsa, her grubun hala çok çeşitli evleri vardır.

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

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

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

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



Examples

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

Ancak max_leaf_nodes argümanı, overfitting ve underfiting'i kontrol etmek için çok mantıklı bir yol sağlar.

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

Max_leaf_nodes için farklı değerlerden MAE puanlarını karşılaştırmaya yardımcı olması için bir yardımcı program işlevi kullanabiliriz:

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.

In [2]:
```
# Data Loading Code Runs At This Point
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
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.model_selection import train_test_split

# split data into training and validation data, for both features and target
train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

Max_leaf_nodes için farklı değerlerle oluşturulan modellerin doğruluğunu karşılaştırmak için bir for-loop kullanabiliriz.

In [3]:
```
# compare MAE with differing values of max_leaf_nodes
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
- **Underfitting:** alakalı pattern'leri yakalayamama, yine daha az doğru tahminlere yol açma.

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

Exercise: Underfitting and Overfitting

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

```
# 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("\nSetup complete")
```

```
Validation MAE: 29,653

Setup complete
```

Exercises

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

[]:

```
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 (Farklı ağaç boyutlarını karşılaştırın)

Bir dizi olası değerden ***max_leaf_nodes*** için aşağıdaki değerleri çalıştıran bir döngü yazın.

Her max_leaf_nodes değerinde get_mae işlevini çağırın. Çıktıyı, verilerinizde en doğru modeli veren max_leaf_nodes değerini seçmenize izin verecek şekilde saklayın.

[10]:
```
candidate_max_leaf_nodes = [5, 25, 50, 100, 250, 500]
# Write loop to find the ideal tree size from candidate_max_leaf_nodes
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))

# Store the best value of max_leaf_nodes (it will be either 5, 25, 50, 100, 250 or 500)
best_tree_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
```

Correct

Step 2: Fit Model Using All Data

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

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

[14]:
```
# 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()
```

Correct

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

Random Forests

Introduction

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

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

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

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

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

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

Example

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

- train_X
- val_X
- train_y
- val_y

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/fores
t.py:245: FutureWarning: The default value of n_estimators wil
l 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'ın da performansını değiştirmenize izin veren parametreler var.

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

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

Exercises: Random Forest

Şimdiye kadar yazdığımız kod:

```
# 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

[28]:
```
from sklearn.ensemble import RandomForestRegressor

# Define the model. Set random_state to 1
rf_model = RandomForestRegressor(random_state=1)

# fit your model
rf_model.fit(train_X, train_y)

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

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

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

```
Validation MAE for Random Forest Model: 21,857
```

Correct

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

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

Exercises: Machine Learning Competitions

Introduction

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

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

Bu notebook'daki adımlar:

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

Şimdiye kadar yazdığımız kod:

```
# 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 competition
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', '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))

# 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 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 * 10/10

- ○ criterion
- ◉ max_depth ✓
- ○ random_state
- ○ splitter

✓ Q2- Which of the below can be said definitely according to the results 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 * 10/10

```
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	1
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 ✓

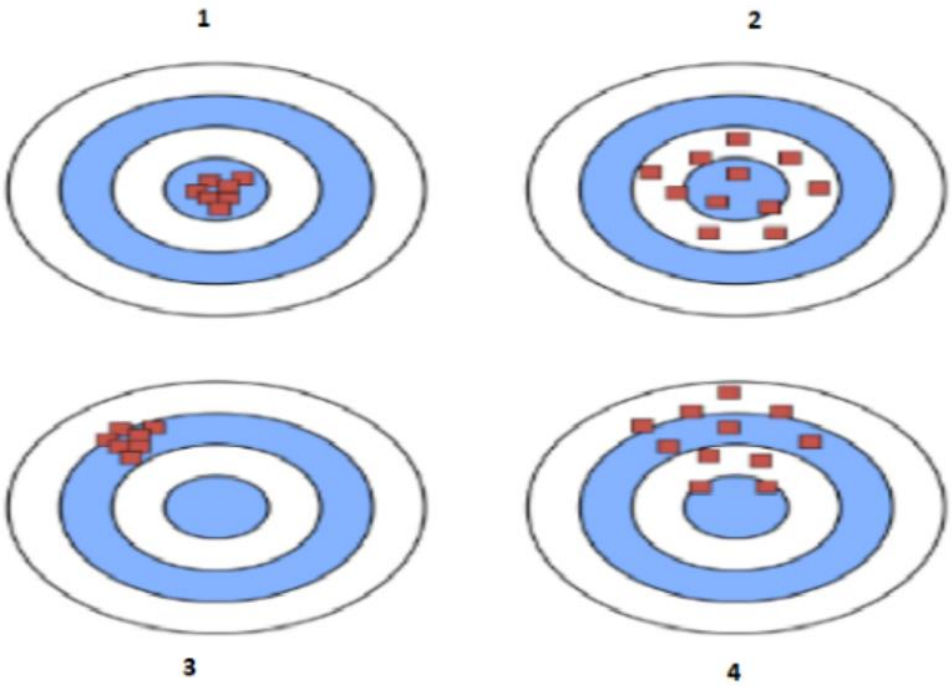
✓ Q3- Which one is false about overfitting and underfitting? * 10/10

- 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.

✓ 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.

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



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

✓ Q6- 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

✗ 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. * 0/10

```
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.

✓ Q8- Trees have their length and we call that the depth of the tree. 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? * 10/10

- ○ 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.

✓ 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()

✓ 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

Intermediate Machine Learning

Introduction

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

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

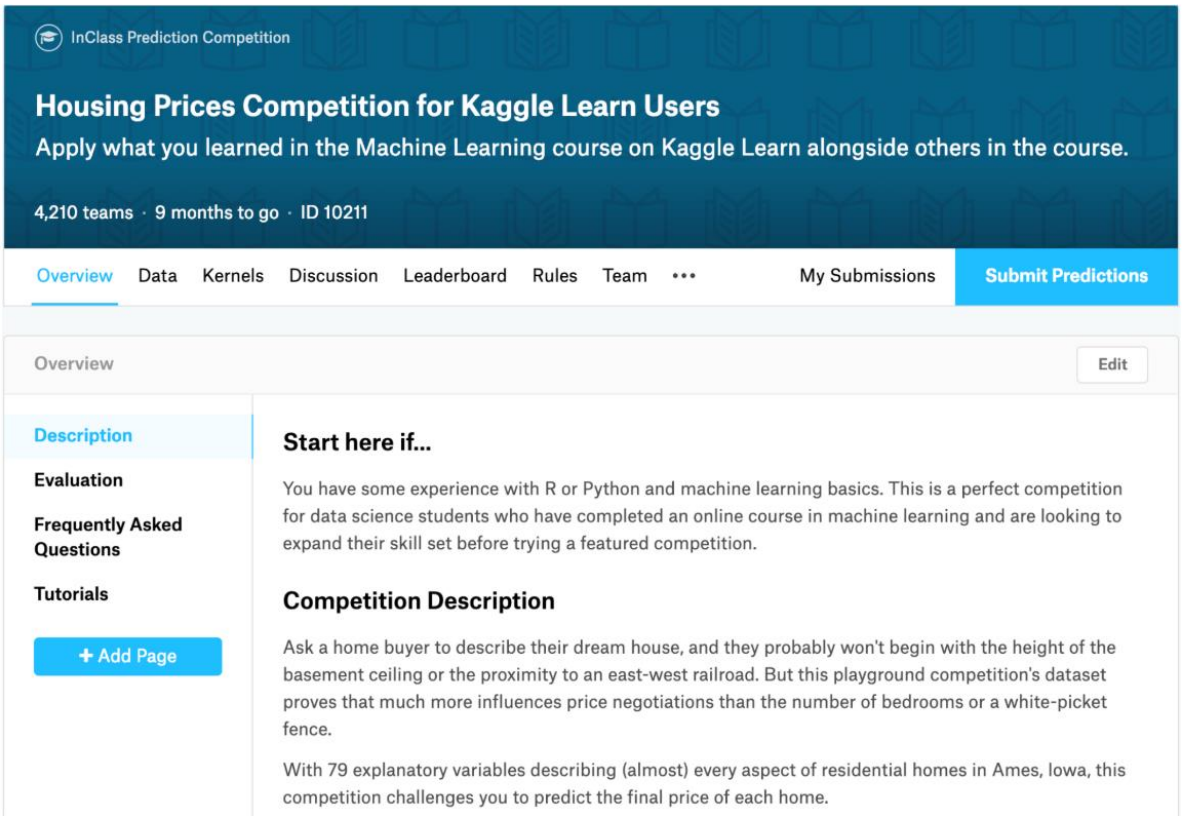
Bu mikro kursta, aşağıdakileri nasıl yapacağınızı öğrenerek makine öğrenimi uzmanlığınızı hızlandıracaksınız:

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

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

Uygulamalı alıştırmalar Housing Prices Competition for Kaggle Learn Users'dan elde edilen verileri kullanır, burada ev fiyatlarını tahmin etmek için 79 farklı açıklayıcı değişken (type of roof, number of bedrooms, and number of bathrooms gibi) kullanacaksınız.

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



Exercises

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

Housing Prices Competition for Kaggle Learn Users'dan elde edilen verilerle, evlerin her yönünü (neredeyse) tanımlayan 79 açıklayıcı değişkeni kullanarak Iowa'daki ev fiyatlarını tahmin etmek için çalışacaksınız.

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

# Read the data
X_full = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Obtain target and predictors
y = X_full.SalePrice
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = X_full[features].copy()
X_test = X_test_full[features].copy()

# 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)
```

[2]:
```
X_train.head()
```

Out[2]:

Id	LotArea	YearBuilt	1stFlrSF	2ndFlrSF	FullBath	BedroomAbvGr	TotRmsAbvGrd
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 : Eveluate Several Models (Birkaç modeli değerlendirin)

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

[3]:
```
from sklearn.ensemble import RandomForestRegressor

# Define the models
model_1 = RandomForestRegressor(n_estimators=50, random_state=0)
model_2 = RandomForestRegressor(n_estimators=100, random_state=0)
model_3 = RandomForestRegressor(n_estimators=100, criterion='mae', random_state=0)
model_4 = RandomForestRegressor(n_estimators=200, min_samples_split=20, random_state=0)
model_5 = RandomForestRegressor(n_estimators=100, max_depth=7, random_state=0)

models = [model_1, model_2, model_3, model_4, model_5]
```

Burada kullandığımız parametrelere göz atalım;

n_estimators : Random Forest içerisinde oluşturulacak ağaç sayısı. Default=10

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

min_samples_split : Bir bölünmenin gerçekleşmesi için verilerinizde bulunması gereken minimum örnek sayısını ayarlar. Eğer bir float ise o zaman min_samples_split*n_samples ile hesaplanır.

> **Not:** İyi sonuçlar genellikle **max_depth=None** ayarında **min_samples_split=1** ile birlikte yapılır. Bu değerleri kullanmanın belleği çok fazla işgal eden modellerle sonuçlanabileceğini unutmayın.

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

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

[4]:
```
from sklearn.metrics import mean_absolute_error

# Function for comparing different models
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)
    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 %d MAE: %d" % (i+1, mae))
```

```
Model 1 MAE: 24015
Model 2 MAE: 23740
Model 3 MAE: 23528
Model 4 MAE: 23996
Model 5 MAE: 23706
```

Aşağıdaki satırı doldurmak için yukarıdaki sonuçları kullanın. Hangi model en iyi modeldir? Cevabınız model_1, model_2, model_3, model_4 veya model_5'ten biri olmalıdır.

```
# Fill in the best model
best_model = model_3

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

Correct

Step 2: Generate Test Prediction (Test tahminleri oluşturun)

```
[8]:
# Define a model
my_model = RandomForestRegressor(n_estimators=100, criterion="mae", random_state=0) # Your code here

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

Correct

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

```
[9]:
# Fit the model to the training data
my_model.fit(X, y)

# Generate test predictions
preds_test = my_model.predict(X_test)

# Save predictions in format used for competition scoring
output = pd.DataFrame({'Id': X_test.index,
                       'SalePrice': preds_test})
output.to_csv('submission.csv', index=False)
```

Missing Values (Eksik Veriler)

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

Verilerin eksik değerlerle sonuçlanmasının birçok yolu vardır. Örneğin,

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

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

Üç Yaklaşım

1) Basit Bir Seçenek: Eksik Değerli Sütunları Düşürme

En basit seçenek, eksik değerlere sahip sütunları düşürmektir.

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

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

2) Daha İyi Bir Seçenek: Imputation

Empütasyon eksik değerleri bir sayı ile 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 bırakmanızdan daha doğru modellere yol açar.

3) An Extension To Imputation

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

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.7

Example

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, zaten X_train, X_valid, y_train ve y_valid'de train ve validation verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.

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

# Load the data
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')

# Select target
y = data.Price

# To keep things simple, we'll use only numerical predictors
melb_predictors = data.drop(['Price'], axis=1)
X = melb_predictors.select_dtypes(exclude=['object'])

# Divide data into training and validation subsets
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.
2,
                                                      random_state=0)
```

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

Eksik değerlerle başa çıkmada farklı yaklaşımları karşılaştırmak için **score_dataset()** işlevini tanımlarız.

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

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, aynı sütunları her iki DataFrame'de de düşürmeye dikkat ediyoruz.

In [3]:

```python
# Get names of columns with missing values
cols_with_missing = [col for col in X_train.columns
                     if X_train[col].isnull().any()]

# 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)

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

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

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

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

In [4]:

```python
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("MAE from Approach 2 (Imputation):")
print(score_dataset(imputed_X_train, imputed_X_valid, y_train, y_valid))
```

```
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.

In [5]:
```
# Make copy to avoid changing original data (when imputing)
X_train_plus = X_train.copy()
X_valid_plus = X_valid.copy()

# Make new columns indicating what will be imputed
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 removed column names; put them back
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))
```

```
MAE from Approach 3 (An Extension to Imputation):
178927.503183954
```

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 alıştırmada, Housing Prices Competition for Kaggle Learn Users verileri ile çalışacaksınız.

[2]:

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

# Read the data
X_full = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
X_full.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X_full.SalePrice
X_full.drop(['SalePrice'], axis=1, inplace=True)

# To keep things simple, we'll use only numerical predictors
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)
```

```
X_train.head()
```

Out[3]:

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

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)

```
# 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])
```

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

Part A

[6]:

```
# Fill in the line below: How many rows are in the training data?
num_rows = 1168

# Fill in the line below: How many columns in the training data
# have missing values?
num_cols_with_missing = 3

# Fill in the line below: How many missing entries are contained in
# all of the training data?
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.

```
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)
```

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

Bu adımda, eksik değerlere sahip sütunları kaldırmak için X_train ve X_valid'deki verileri önceden işlersiniz. Önceden işlenmiş DataFrames değerini sırasıyla low_X_train ve low_X_valid olarak ayarlayın.

[10]:
```
# Fill in the line below: get names of columns with missing values
cols_with_missing = [col for col in X_train.columns if X_train[col].isnull().any()] # Your code here

# 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()
```

```
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.

```
from sklearn.impute import SimpleImputer

# Fill in the lines below: 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 olmadan sonraki kod hücresini çalıştırın.

[13]:
```
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? Sizce neden bir yaklaşım diğerinden daha iyi performans gösteriyor?

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

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

Step 4: Generate test predictions

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

Part A

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

- önceden işlenmiş DataFrame'ler aynı sayıda sütuna sahiptir,
- önceden işlenmiş DataFrame'lerde eksik değer yoktur,
- final_X_train ve y_train aynı sayıda satıra sahip olmalıdır,
- final_X_valid ve y_valid aynı sayıda satıra sahip olmalıdır.

```
# Preprocessed training and validation features
final_X_train = reduced_X_train
final_X_valid = reduced_X_valid

# Check your answers
step_4.a.check()
```

Eksik değer içeren sütunları drop işlemine tabi tuttuğumuz durumu seçtik.

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

```
# Define and fit model
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(final_X_train, y_train)

# Get validation predictions and MAE
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 hücresini kullanın. Eğitim ve doğrulama verilerini nasıl önceden işleme koyduğunuzu kabul eden bir yöntem kullandığınızdan emin olun ve önceden işlenmiş test feature'larını `final_X_test` olarak ayarlayın.

Ardından, `preds_test 'içinde test tahminleri oluşturmak için önceden işlenmiş test feature'larını ve eğitimli modeli kullanın.

[63]:
```
#X_train'den düşürdüğümüz kolonları X_test'den de düşürmeliyiz.
final_X_test = X_test.drop(cols_with_missing, axis=1)
```

[69]:
```
#X_test icerisinde hala eksik deger iceren kolonlar mevcut.
#bu eksik degerleri bir sonraki satirda ele alacagiz.
final_miss = [col for col in final_X_test.columns if final_X_test[col].isnull().any()]
final_miss
```

Out[69]
```
['BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 'BsmtFullBath',
 'BsmtHalfBath',
 'GarageCars',
 'GarageArea']
```

[75]:
```
#Eksik degerleri drop etmiyoruz. Cunku X_train ile aynı kolonlara sahip olmalilar.
#Eksik degerleri ortalama degerler ile dolduruyoruz.
final_X_test.fillna(final_X_test[final_miss].mean(), inplace=True)
```

```
# Fill in the line below: preprocess test data
final_X_test

# Fill in the line below: get test predictions
preds_test = model.predict(final_X_test)

step_4.b.check()
```

Correct

149...	Recep Aydoğdu		16592.77...	3	2m

Your Best Entry ↑

You advanced 11,921 places on the leaderboard!
Your submission scored 16592.77974, which is an improvement of your previous score of 20998.83780. Great job!

Tweet this!

Kaynaklar

- Kaggle – Intro to Machine Learning Course
  https://www.kaggle.com/learn/intro-to-machine-learning
- Kaggle – Intermediate Machine Learning Course
  https://www.kaggle.com/learn/intermediate-machine-learning