

The Kaggle logo consists of the word "kaggle" in a large, lowercase, blue sans-serif font. A small trademark symbol (TM) is located at the top right of the letter "e".

Global Ai Hub Kaggle Master Online Mentorluk Programı

Çalışma Notları
Recep Aydoğdu

Dökümanın güncel haline GitHub profilimden ulaşabilirsiniz:
<https://github.com/recepayddogdu>

İçindekiler

Intro to Machine Learning	5
How Models Work (Modeller Nasıl Çalışır?)	5
Giriş.....	5
Decision Tree'nin Geliştirilmesi.....	6
Basic Data Exploration (Basit Veri Keşfi)	8
Verilerinizi Tanıtmak için Pandas Kullanımı	8
Interpreting Data Description (Verilerin Yorumlanması).....	9
Excercise: Explore Your Data.....	10
Your First Machine Learning Model	12
Selecting Data for Modeling (Modelleme için Veri Seçmek).....	12
Choosing "Features" (Özellik Seçimi)	13
Building Your Model (Model Oluşturma)	15
Exercise: Your First Machine Learning Model.....	17
Model Validation (Model Geçerliliği).....	21
Model Validation Nedir?	21
The Problem with "In-Sample" Scores	24
Coding It.....	24
Wow!	25
Exercise: Model Validation.....	25
Underfitting and Overfitting	30
Farklı Modellerle Deneme.....	30
Examples.....	32
Sonuç	33
Exercise: Underfitting and Overfitting	34
Random Forests	36
Introduction.....	36
Example	36
Sonuç	37
Exercises: Random Forest	38
Exercises: Machine Learning Competitions	40
Introduction	40
Creating a Model For the Competition	42
Make Predictions	42
Quiz: Intro to Machine Learning	43

Intermediate Machine Learning.....	49
Introduction.....	49
Exercises	50
Step 1 : Eveluate Several Models (Birkaç modeli değerlendirin)	50
Step 2: Generate Test Prediction (Test tahminleri oluşturun)	52
Missing Values (Eksik Veriler).....	52
Üç Yaklaşım.....	52
Example	53
Sonuç	56
Exercises (Missing Values).....	57
Categorical Variables.....	63
Introduction.....	63
Üç Yaklaşım.....	63
Example	64
En iyi yaklaşım hangisi?	68
Sonuç	68
Exercises: Categorical Variables	69
Pipelines	77
Introduction.....	77
Example	77
Step 1: Önişleme Adımlarını Tanımlayın	79
Step 2: Modeli tanımlayın	80
Step 3: Pipeline Oluşturun ve Değerlendirin	80
Sonuç	80
Exercise: Pipelines	81
Cross-Validation	86
Introduction.....	86
Cross-Validation Nedir?.....	86
Ne Zaman Cross-Validation Kullanmalıyız?	87
Example	87
Sonuç	89
Exercise: Cross-Validation	89
Step 1: Write a Usefull Function.....	91
Step 2: Test Different Parameter Values.....	91
Step 3: Find the Best Parameter Value	92
XGBoost.....	93

Introduction.....	93
Gradient Boosting.....	93
Example	94
Parameter Tuning (Parametre Ayarı)	96
Sonuç	98
Exercise: XGBoost.....	99
Data Leakage (Veri Sızıntısı)	103
Introduction.....	103
Target Leakage	103
Train-Test Contamination.....	104
Example	104
Sonuç	106
Exercise: Data Leakage	107
Data Visualization	108
Hello, Seaborn.....	108
Notebook Kurulumu	108
Veri Yükleme.....	108
Verileri İnceleyelim.....	110
Plot the Data (Verileri Çizin)	110
Kaynaklar.....	112

Intro to Machine Learning

Makine öğrenmesindeki temel fikirleri öğrenin ve ilk modellerinizi oluşturun.

How Models Work (Modeller Nasıl Çalışır?)

Giriş

Makine öğrenimi modellerinin nasıl çalıştığını 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:

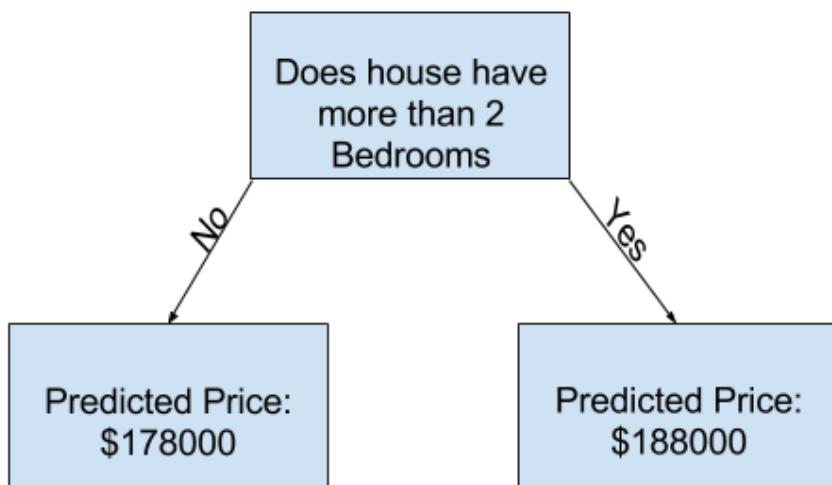
Kuzeniniz gayrimenkul konusunda spekülasyonlarla milyonlarca dolar kazandı. Veri bilimine gösterdiğiniz ilgi nedeniyle sizinle iş ortağı olmayı teklif etti. Parayı tedarik edecek ve çeşitli evlerin ne kadar değerli olduğunu tahmin eden modeller sunacaksınız.

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.

Sample Decision Tree



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

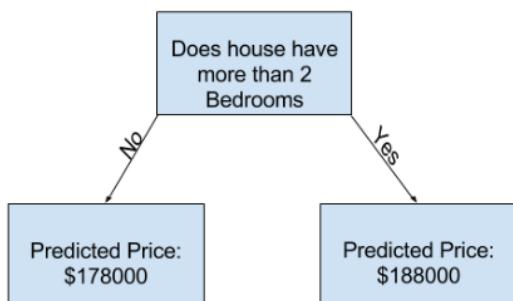
Verileri, evlerin iki gruba nasıl ayrılacağına karar vermek için ve sonra her grupta öngörülen fiyatı belirlemek için kullanıyoruz. Verilerden pattern yakalamanın bu adımına, modelin fit edilmesi(**fitting**) veya train edilmesi(**training**) denir. Modelin **fit** edilmesi için kullanılan verilere **training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

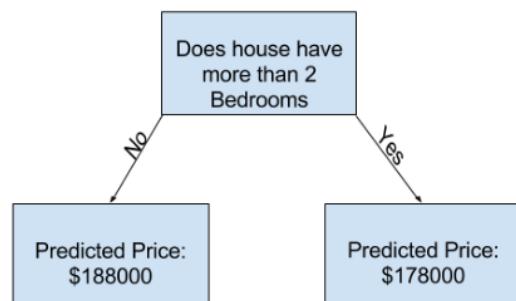
Decision Tree'nin Geliştirilmesi

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

1st Decision Tree



2nd Decision Tree



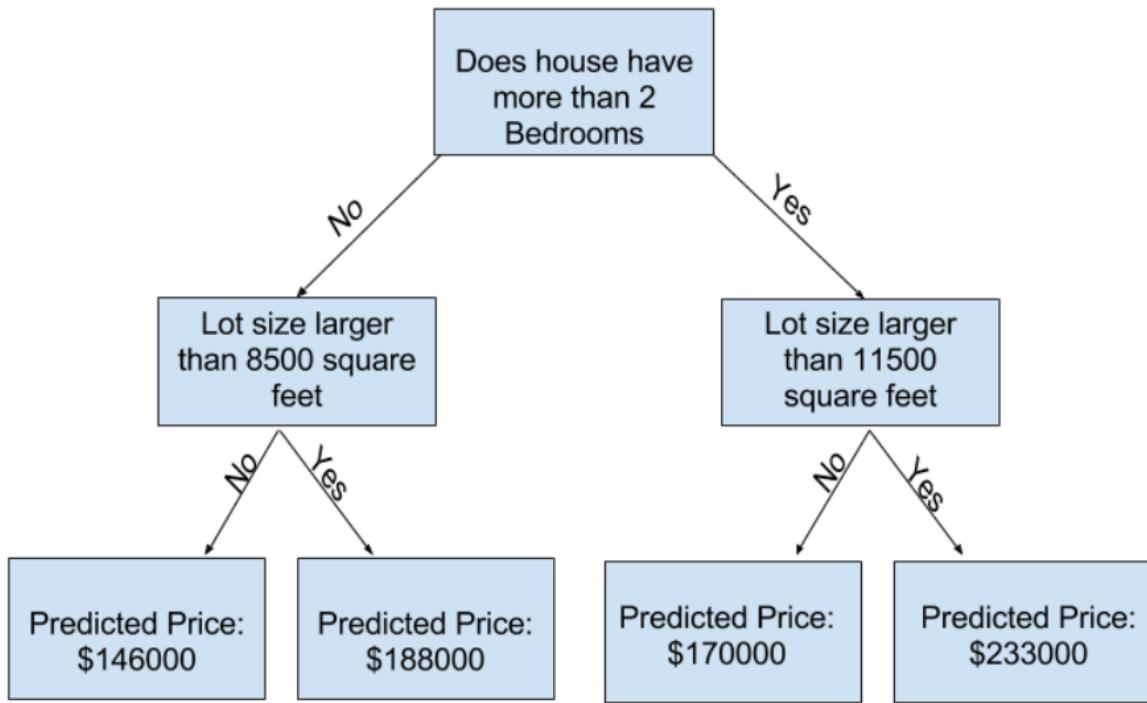
Soldaki karar ağacı (Decision Tree 1) muhtemelen daha mantıklıdır, çünkü daha fazla yatak odası olan evlerin daha az yatak odası olan evlerden daha yüksek fiyatlarıla satılma eğiliminde olduğu gerçeğini yakalar.

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 ağaçın 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 tanıtmaktır.

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:

```
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 fiyatlarılarındaki 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	Latitude	Longitude	Propertycount
.000000	13518.000000	13580.000000	7130.000000	8205.000000	13580.000000	13580.000000	13580.000000
?42	1.610075	558.416127	151.967650	1964.684217	-37.809203	144.995216	7454.417378
'12	0.962634	3990.669241	541.014538	37.273762	0.079260	0.103916	4378.581772
)00	0.000000	0.000000	0.000000	1196.000000	-38.182550	144.431810	249.000000
)00	1.000000	177.000000	93.000000	1940.000000	-37.856822	144.929600	4380.000000
)00	2.000000	440.000000	126.000000	1970.000000	-37.802355	145.000100	6555.000000
)00	2.000000	651.000000	174.000000	1999.000000	-37.756400	145.058305	10331.000000
)00	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**.

Listeyi dörde bölün, dörde bölünen bölümlerden ilkinin son elemanına bakın. Örneğin, 200 elemanlık listeyi dörde bölünce, ilk bölümün son elemanı 50 olur.

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()
```

ut[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 da yeni değil. Bunun için birkaç farklı durum olabilir:

1- Bu verilerin toplandığı yeni evler inşa etmediler.

2- Veriler uzun zaman önce toplanmıştır. Veri toplandıktan sonra inşa edilen evler görünmüyordur.

Nedeni yukarıdaki 1. açıklama ise, bu, bu verilerle oluşturduğunuz modele olan güveninizi etkiler mi? 2. açıklama 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 yazdırma için çok fazla değişkeni vardı. Bu başa çıkmaz veri miktarını anlayabileceğiniz bir şeye nasıl ayıralırsınız?

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 gereklidir. 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', 'Latitude',  
       'Longitude', '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 alacağımız ve verilerimizden eksik değere sahip evleri düşüreceğiz.

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

```
In [2]:  
# The Melbourne data has some missing values (some houses for which some variables weren't recorded.)  
# 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 yaklaşımı 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 **Seride** 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', 'Latitude', 'Longitude']
```

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	Longitude
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	Longitude
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 sıkılıkla 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 ağaçları 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 çalıştırımda 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 çalıştırıldığımızda aynı çıktıyi alabilmek için girdiğimiz bir ifade. Örneğin, validation ve training olarak datayı ayıırken Python her seferinde datayı farklı yerlerinden böler, bir random state değeri belirlediğimizde de her çalıştırıldığımızda aynı şekilde bölmüş olur ve aynı sonucu vermiş olur. Farklı değerler verdiğinde farklı sonuçlar aldığına göreceksin.

En iyi karar ağacını bulma problemi NP-Complete olarak sınıflandırılan problemlerdendir. Bu tip problemlerin çözümlerinde sezgisel algoritmalar kullanılır. Sezgisel algoritmalarla 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 çalıştırduğunda aynı ağaç elde etmek istersen **random_state** parametresini bir tamsayıya eşitlemen gereklidir. Hangi tamsayıya eşitlediğinin bir önemi yok.

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

Random_state için bir sayı belirtmek, her çalıştırımda aynı sonuçları almanızı sağlar. Bu iyi bir uygulama olarak kabul edilir.

Herhangi bir sayı kullanabilirsiniz ve model kalitesi tam olarak hangi değeri seçtiğinize bağlı olmayacağından emin olun.

Şimdi tahminler yapmak için kullanabileceğimiz uygun bir modelimiz var.

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

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  Latitude  Longitude  
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 gereklidir.

```
[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 çalışırmak 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 oluşturduğunuz. Ama bu model ne kadar iyi?

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

Model Validation Nedir?

Oluşturduğunuz hemen hemen her modeli değerlendirmek isteyeceksiniz.

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

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

Önce model kalitesini anlaşılır bir şekilde özetlemeniz gereklidir.

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 dile şöyleden 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', 'Latitude', 'Longitude']
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 veriörneğinde, yeşil kapıya sahip tüm evler çok pahalıydı.

Modelin işi, ev fiyatlarını tahmin eden pattern'ler bulmaktır, bu yüzden bu pattern'i görecekt, 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(örbüntü) tutmazsa, model pratikte kullanıldığından 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 kullanmaktadır.

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

Coding It

Scikit-learn kütüphanesi, verileri iki parçaaya 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 oluşturduğunuz. 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.

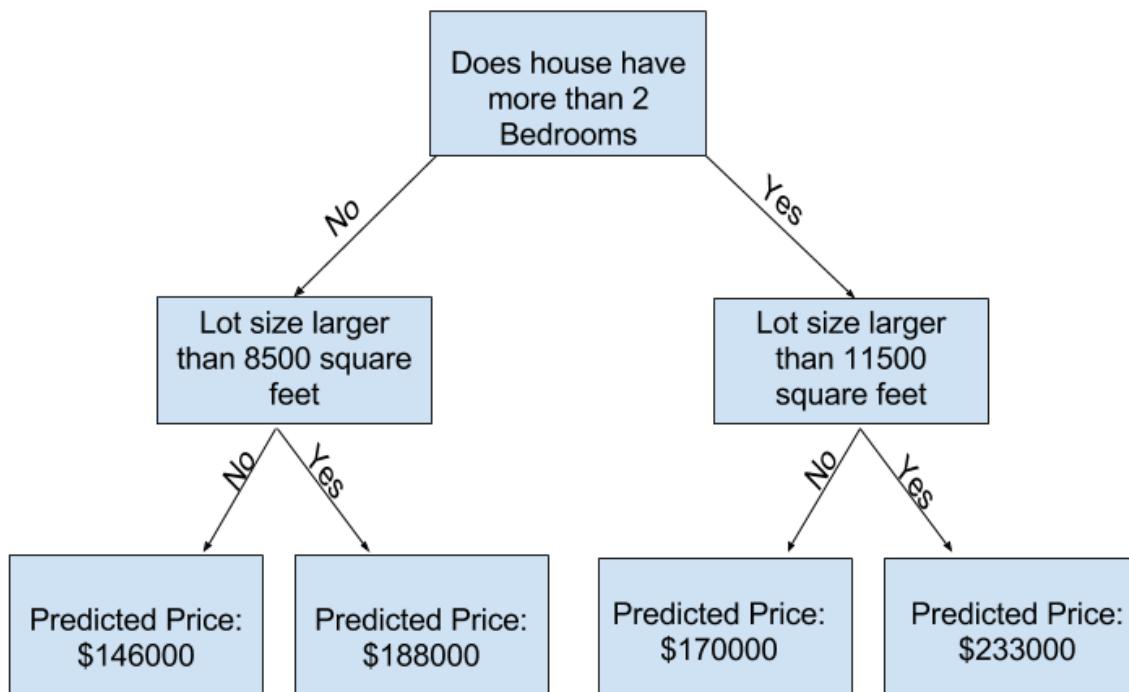
Farklı Modellerle Deneme

Artık model doğruluğunu ölçmenin güvenilir bir yoluna sahip olduğunuzu 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 seçenek sahip olduğunu görebilirsiniz (isteyeceğinizden veya ihtiyacınız olandan daha fazla).

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



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

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

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

Her grup tekrar bölünürse, 4 grup ev alındıktır. 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 da 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 ağaçımızı çok sıç yaparsak, evleri çok farklı gruptara ayırmaz.

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

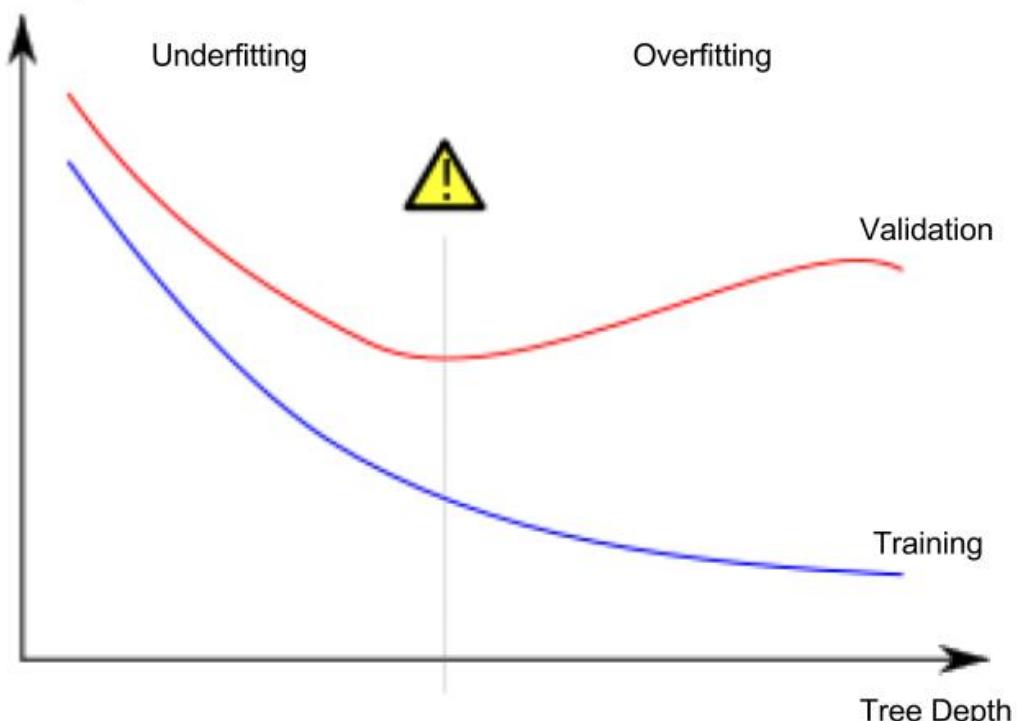
Sonuç tahminleri(predictions), train verilerinde bile çoğu ev için çok uzak olabilir (ve aynı nedenden dolayı validation(doğrulama) da kötü olacaktır).

Bir model verilerdeki önemli ayırmaları 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) önemsiyoruz için, **underfitting** ve **overfitting** arasındaki tatlı noktayı bulmak istiyoruz.

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

Mean Average Error



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 underfitting'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 yazdığınıza) 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', 'Longitude']  
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:** alaklı 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 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("\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.

```
[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 (Farklı ağaç boyutlarını karşılaştırın)

Bir dizi olası değerden **max_leaf_nodes** için aşağıdaki değerleri çalıştırın 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: {} \t\t Mean absolute error: {}".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 getirirsınız.

Yani, tüm modelleme kararlarınızı verdığınız için doğrulama verilerini saklamamız gerekmek.

```
[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 ağaçları 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

[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 başına 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> *

- 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 *

```
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 ✓

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

10/10

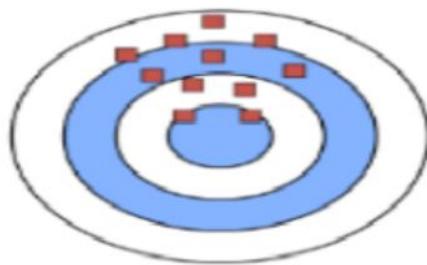
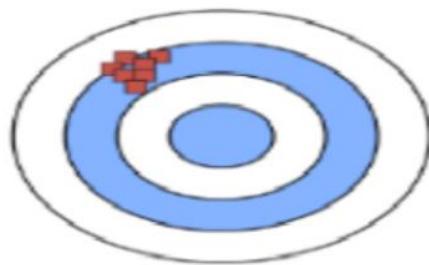
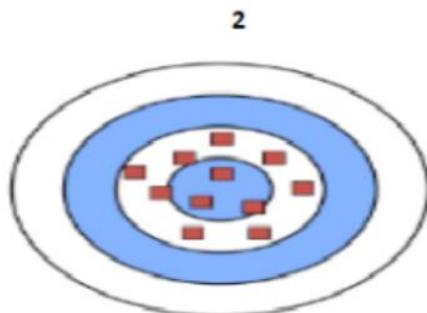
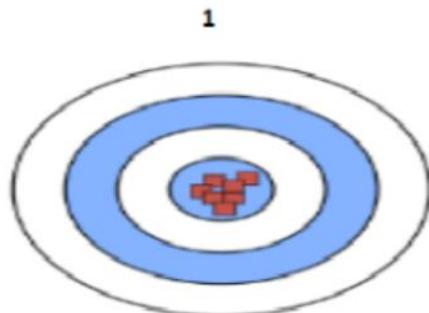
- Insufficient training (less epoch less batch size), causes underfitting.
- Training on too much epoch and batch size causes overfitting.
- Splitting dataset as train and test datasets will always be enough to prevent overfitting, no need for validation datasets. ✓
- In overfitting accuracy will be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

✓ 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 0/10 separately? The rest of the code is exactly the same. *

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. 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.

✓ Q9- Let assume, we have a data set called `home_data` with 3 features 10/10

names; `LotArea`, `YearBuilt`, `PoolArea`. How do you define non-missing values for the feature `LotArea`? *

- `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 uzmanlığını hızlandıracaksınız:

- gerçek dünya veri kümelerinde sıklıkla bulunan veri türlerini ele alır (missing values, categorical variables),
- makine öğrenme kodunuzun kalitesini artırmak için **pipeline'lar** tasarlama,
- 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 güçlendirceksiniz.

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 ilerlemeniziölçeceksiniz!

The screenshot shows the competition landing page. At the top, it says "InClass Prediction Competition" and "Housing Prices Competition for Kaggle Learn Users". Below that, a sub-headline reads "Apply what you learned in the Machine Learning course on Kaggle Learn alongside others in the course.". It displays "4,210 teams · 9 months to go · ID 10211". The navigation bar includes "Overview", "Data", "Kernels", "Discussion", "Leaderboard", "Rules", "Team", "...", "My Submissions", and "Submit Predictions". The main content area has tabs for "Overview", "Description", "Evaluation", "Frequently Asked Questions", and "Tutorials". The "Description" tab is active, showing the text: "Start here if... You have some experience with R or Python and machine learning basics. This is a perfect competition for data science students who have completed an online course in machine learning and are looking to expand their skill set before trying a featured competition." The "Competition Description" tab is also visible, containing the text: "Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this playground competition's dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence. With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home."

Exercises

Bir işinma 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]:

	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 : 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** ayında **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**'ınızı ayarlamanzı, 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 yaklaşımınların etkilerini gerçek dünyadaki bir veri kümesinde karşılaştıracaksınız.

Verilerin eksik değerlerle sonuçlanmasıın 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 düşürmektedir.



Bed	Bath	Bath
1.0	1.0	1.0
2.0	1.0	1.0
3.0	2.0	2.0
NaN	2.0	2.0

Düşürülen sütunlardaki değerlerin çoğu eksik değilse, model bu yaklaşımıla 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 yaklaşımın 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 yüklemeye adımlına 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]:
# 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 bağladıkten sonra genellikle ek bir fayda sağlamaz.

```
In [4]:
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
```

Sonuç

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 (Missing Values)

Şimdi, kayıp değerlerin işlenmesi hakkında 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]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...
Id											
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 çıkmmanın 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 gelirse 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 yaklaşımından MAE'yi karşılaştırın. Sonuçlar hakkında sizi şaşırtan bir şey var mı? Sizce neden bir yaklaşım diğerinden daha iyi performans gösteriyor?

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 *GarageYrsBlt* 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 gereklidir:

- önceden işlenmiş DataFrame'ler aynı sayıda sütuna sahiptir,
- önceden işlenmiş DataFrame'lerde eksik değer yoktur,
- *final_X_train* ve *y_train* aynı sayıda satırda sahip olmalıdır,
- *final_X_valid* ve *y_valid* aynı sayıda satırda 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 işlemeye 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` 'inde 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 satırda ele alacağız.
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 olmalıdır.  
#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!

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 karşılaşacağız.

Üç Yaklaşım

1) Drop Categorical Variables

Kategorik değişkenlerle başa çıkmadan 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.

The diagram illustrates the process of Label Encoding. On the left, there is a table titled 'Breakfast' with five rows: 'Every day', 'Never', 'Rarely', 'Most days', and 'Never'. An arrow points from this table to another table on the right, also titled 'Breakfast', which contains the numerical values 3, 0, 1, 2, and 0 respectively, corresponding to the categories in the first table.

Breakfast
Every day
Never
Rarely
Most days
Never

→

Breakfast
3
0
1
2
0

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

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

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

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

3) One-Hot Encoding

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

Bunu anlamak için bir örnek üzerinde çalışacağız.

The diagram illustrates the process of one-hot encoding. On the left, there is a vertical table with a header 'Color' and five rows labeled 'Red', 'Red', 'Yellow', 'Green', and 'Yellow'. An arrow points from this table to a larger matrix on the right. The matrix has three columns labeled 'Red', 'Yellow', and 'Green'. The first two rows of the matrix correspond to the first two 'Red' entries in the original table, with '1's in the 'Red' column and '0's in the other two. The third row corresponds to the 'Yellow' entry, with '0' in 'Red', '1' in 'Yellow', and '0' in 'Green'. The fourth row corresponds to the 'Green' entry, with '0' in all three columns. The fifth row corresponds to the last 'Yellow' entry, with '0' in 'Red', '1' in 'Yellow', and '0' in 'Green'.

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.

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

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

Example

Önceki derste olduğu gibi [Melbourne Housing dataset](#) üzerinde çalışacağımız.

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

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

# Separate target from predictors
y = data.Price
X = data.drop(['Price'], axis=1)

# Divide data into training and validation subsets
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)
```

```

# Drop columns with missing values (simplest approach)
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" means the number of unique values in a column
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
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']]

# Keep selected columns only
my_cols = low_cardinality_cols + numerical_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()

```

In [2]:
X_train.head()

Out[2]:

	Type	Method	Regionname	Rooms	Distance	Postcode	Bedroom2	Bathroom	Landsize	Lattitude	Longitu
12167	u	S	Southern Metropolitan	1	5.0	3182.0	1.0	1.0	0.0	-37.85984	144.986
6524	h	SA	Western Metropolitan	2	8.0	3016.0	2.0	2.0	193.0	-37.85800	144.900
8413	h	S	Western Metropolitan	3	12.6	3020.0	3.0	1.0	555.0	-37.79880	144.822
2919	u	SP	Northern Metropolitan	3	13.0	3046.0	3.0	1.0	265.0	-37.70830	144.915
6043	h	S	Western Metropolitan	3	13.3	3020.0	3.0	1.0	673.0	-37.76230	144.827

Longitude	Propertycount
144.9867	13240.0
144.9005	6380.0
144.8220	3755.0
144.9158	8870.0
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.

In [3]:

```
# Get list of categorical variables
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 olduğunda 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)

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

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

```
from sklearn.preprocessing import LabelEncoder

# Make copy to avoid changing original data
label_X_train = X_train.copy()
label_X_valid = X_valid.copy()

# Apply label encoder to each column with categorical data
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 hücresinde, her sütun için, her benzersiz değeri rastgele farklı bir tamsayıya atarız. Bu, özel etiketler sağlamaktan daha basit olan yaygın bir yaklaşımdır; ancak, tüm sıralı değişkenler için daha iyi bilgilendirilmiş etiketler sağlarsak, performansta ek bir artış bekleyebiliriz.

Score from Approach 3 (One-Hot Encoding)

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

- Validation verileri, training verilerinde gösterilmeyen sınıflar içerdiginde 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 faydası görünmemektedir.

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

Sonuç

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

Exercises: Categorical Variables

Kategorik değişkenleri encode ederek şimdide 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 hücresini değiştirmeden çalıştırın. Test seti X_test'e yüklenir.

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

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

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

# To keep things simple, we'll drop columns with missing values
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 kümelerinin hem sayısal hem de kategorik değişkenler içeriğine dikkat edin. Bir modeli eğitmeden önce kategorik verileri encode işlemeye tabi tutmanız gereklidir.

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

# 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 1: Drop columns with categorical data

En basit yaklaşımıyla başlayacağınız. Kategorik veriler içeren sütunları kaldırmak için X_train ve X_valid'deki verileri önceden işlemek için aşağıdaki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla drop_X_train ve drop_X_valid olarak ayarlayın.

```
# Fill in the lines below: drop columns in training and validation data
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.

```
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):
17837.82570776256
```

Step 2: Label Encoding

Label Encoding'e geçmeden önce veri kümесini araştıracağiz. Özellikle, "Condition2" sütununa bakacağiz. Aşağıdaki kod hücresi, hem eğitim hem de doğrulama kümelerindeki benzersiz girişleri yazdırır.

```
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' 'RRAe' 'RRNn' 'Artery' 'Feedr' 'PosN']
```

Şimdi buna göre kod yazarsanız:

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

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

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 atanmış bir tamsayı olmaz.

Validation verilerindeki "Condition2" sütununun 'RRAe' ve 'RRNn' değerlerini içerdigine dikkat edin, ancak bunlar eğitim verilerinde görünmez - bu nedenle, scikit-learn ile bir etiket kodlayıcı kullanmaya çalışırsak, kodu hata verir.

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

Sorunlu sütunları *bad_label_cols* Python listesine kaydetmek için aşağıdaki kod hücresini çalıştırın. Benzer şekilde, güvenli bir şekilde etiketlenebilen sütunlar *good_label_cols* içinde saklanır.

```
# All categorical columns
object_cols = [col for col in X_train.columns if X_train[col].dtype == "object"]

# Columns that can be safely label encoded
good_label_cols = [col for col in object_cols if
                   set(X_train[col]) == set(X_valid[col])]

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

X_train ve X_valid içindeki verilere label encode yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla label_X_train ve label_X_valid olarak ayarlayın.

- Kategorik sütunları veri kümesinden bad_label_cols içine çekmek için aşağıdaki kodu sağladık.
- Kategorik sütunlar içinden good_label_cols'lara label encode uygulamanız gereklidir.

```
from sklearn.preprocessing import LabelEncoder

# Drop categorical columns that will not be encoded
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()

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])      # Your code here

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

```
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 kaldırımdan daha iyi sonuçlar verdiği gördünüz.

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

```

# Get number of unique entries in each column with categorical data
object_nunique = list(map(lambda col: X_train[col].nunique(), object_cols))
d = dict(zip(object_cols, object_nunique))

# Print number of unique entries by column, in ascending order
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)]

```

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.

```

# Fill in the line below: How many categorical variables in the training data
# have cardinality greater than 10?
high_cardinality_numcols = 3

# Fill in the line below: How many columns are needed to one-hot encode the
# 'Neighborhood' variable in the training data?
num_cols_neighborhood = 25

# Check your answers
step_3.a.check()

```

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

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

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

Aşağıdaki satırları doldurmak için cevaplarınızı kullanın.

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

```
# Fill in the line below: How many entries are added to the dataset by
# replacing the column with a one-hot encoding?
OH_entries_added = 1e4*100 - 1e4

# Fill in the line below: How many entries are added to the dataset by
# replacing the column with a label encoding?
label_entries_added = 0

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

Çözüm Açıklaması: Elinizde 100 unique değeri olan 10000 tane kolonunuz var.

One Hot Encoding her unique kolon değeri için yeni kolon oluşturulması anlamına geliyor. Buradaki 10e4 aslında 10000 anlamına gelir. ($e=exponential$ yani 10^4 üzeri 4) Bu yüzden de one hot encoding'te 10000 kolonumuz zaten vardı. 100 tane unique entrymiz olduğu için $10000 * 100 - 10000$ (zaten elimizde 10000 başta vardı o yüzden çıkardık) kolon eklenecektir.

Label Encode ise her unique değer için bir sayı verilmesi demektir burada kolon eklenmez sadece var olan kolonlara sayı değerleri yazılır. O yüzden eklenecek kolon sayısı 0'dır.

Step 4: one-hot encoding

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

Low_cardinality_cols değerini one-hot encoding uygulanacak sütunları içeren bir Python listesine ayarlamak için aşağıdaki kod hücresini değiştirmeden çalıştırın. Benzer şekilde, high_cardinality_cols, veri kümeseinden bırakılacak kategorik sütunların bir listesini içerir.

```
# 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: ['Neighborhood', 'Exterior2nd', '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 oluşturmadan önce test verilerini önceden işlemeniz gereklidir.

Pipelines

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

Introduction

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

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

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

Example

Önceki derste olduğu gibi, [Melbourne Housing dataset](#) ile çalışacağız.

Veri yükleme adımına odaklanmayacağız. Bunun yerine, X_train, X_valid, y_train ve y_valid'de zaten eğitim ve doğrulama verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsınız.

```

import pandas as pd
from sklearn.model_selection import train_test_split

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

# Separate target from predictors
y = data.Price
X = data.drop(['Price'], axis=1)

# Divide data into training and validation subsets
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)

# "Cardinality" means the number of unique values in a column
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
categorical_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 ['int
64', 'float64']]

# Keep selected columns only
my_cols = categorical_cols + numerical_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()

```

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

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

```
Out[2]:
```

	Type	Method	Regionname	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	Y
12167	u	S	Southern Metropolitan	1	5.0	3182.0	1.0	1.0	1.0	0.0	Nan	1
6524	h	SA	Western Metropolitan	2	8.0	3016.0	2.0	2.0	1.0	193.0	Nan	1
8413	h	S	Western Metropolitan	3	12.6	3020.0	3.0	1.0	1.0	555.0	Nan	1
2919	u	SP	Northern Metropolitan	3	13.0	3046.0	3.0	1.0	1.0	265.0	Nan	1
6043	h	S	Western Metropolitan	3	13.3	3020.0	3.0	1.0	2.0	673.0	673.0	1

Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longitude	Propertycount
5.0	3182.0	1.0	1.0	1.0	0.0	Nan	1940.0	-37.85984	144.9867	13240.0
8.0	3016.0	2.0	2.0	1.0	193.0	Nan	Nan	-37.85800	144.9005	6380.0
12.6	3020.0	3.0	1.0	1.0	555.0	Nan	Nan	-37.79880	144.8220	3755.0
13.0	3046.0	3.0	1.0	1.0	265.0	Nan	1995.0	-37.70830	144.9158	8870.0
13.3	3020.0	3.0	1.0	2.0	673.0	673.0	1970.0	-37.76230	144.8272	4217.0

Pipeline'nın tamamını üç adımda inşa ediyoruz.

Step 1: Önişleme Adımlarını Tanımlayın

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

Aşağıdaki kod:

- **sayısal** verilerdeki eksik değerleri ifade eder ve
- eksik değerleri ifade eder ve **kategorik** verilere one-hot encoding uygular.

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Preprocessing for numerical data
numerical_transformer = SimpleImputer(strategy='constant')

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

```

Step 2: Modeli tanımlayın

Ardından, tanıdık RandomForestRegressor sınıfıyla bir Random Forest modeli tanımlarız.

In [4]:

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, random_state=0)
```

Step 3: Pipeline Oluşturun ve Değerlendirin

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

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

In [5]:

```
from sklearn.metrics import mean_absolute_error

# Bundle preprocessing and modeling code in a pipeline
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                             ('model', model)
                            ])

# Preprocessing of training data, fit model
my_pipeline.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(X_valid)

# Evaluate the model
score = mean_absolute_error(y_valid, preds)
print('MAE:', score)
```

MAE: 160679.18917034855

Sonuç

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

Exercise: Pipelines

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

Çalışmamızda [Housing Prices Competition for Kaggle Learn Users](#) datasetini kullanacağız.



X_train, X_valid, y_train ve y_valid'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X_test'e yüklenir.

```
In [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)  
  
# Break off validation set from training data  
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X_full, y,  
                                                               train_size=0.8, test_size=0.2,  
                                                               random_state=0)  
  
# "Cardinality" means the number of unique values in a column  
# Select categorical columns with relatively low cardinality (convenient but arbitrary)  
categorical_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']]  
  
# Keep selected columns only  
my_cols = categorical_cols + numerical_cols  
X_train = X_train_full[my_cols].copy()  
X_valid = X_valid_full[my_cols].copy()  
X_test = X_test_full[my_cols].copy()
```

```
In [3]: X_train.head()
```

```
Out[3]:
```

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Condition1	Condition2	...	Gar
Id												
619	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Norm	Norm	...	774
871	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	PosN	Norm	...	308
93	RL	Pave	Grvl	IR1	HLS	AllPub	Inside	Gtl	Norm	Norm	...	432
818	RL	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	Norm	Norm	...	857
303	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Norm	Norm	...	843

5 rows × 76 columns

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

```
In [4]:
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Preprocessing for numerical data
numerical_transformer = SimpleImputer(strategy='constant')

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Define model
model = RandomForestRegressor(n_estimators=100, random_state=0)

# Bundle preprocessing and modeling code in a pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('model', model)
                     ])

# Preprocessing of training data, fit model
clf.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
preds = clf.predict(X_valid)

print('MAE:', mean_absolute_error(y_valid, preds))
```

MAE: 17861.780102739725

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

Step 1: Performansı Arttırın

Part A

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

- *numerical_transformer*
- *categorical_transformer*
- *model*

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

```
In [5]:
# Preprocessing for numerical data
numerical_transformer = SimpleImputer(strategy="median") # Your code here

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))]) # Your code here

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Define model
model = RandomForestRegressor(n_estimators=100, random_state=0) # Your code here

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

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

Part B

Bu adımı geçmek için, Part A'da, yukarıdaki koddan daha düşük MAE elde eden bir pipeline tanımlamanız gereklidir.

Burada zaman ayırip MAE'yi ne kadar düşük alabileceğinizi görmek için birçok farklı yaklaşımı denemeniz önerilir! (Kodunuz geçmezse, lütfen ön işleme adımlarını ve modelini Part A'da değiştirin.)

```
In [7]:
# Bundle preprocessing and modeling code in a pipeline
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                             ('model', model)
                            ])

# Preprocessing of training data, fit model
my_pipeline.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(X_valid)

# Evaluate the model
score = mean_absolute_error(y_valid, preds)
print('MAE:', score)

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

MAE: 17487.872363013696

Correct

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

Step 2: Test Tahminleri Oluşturun

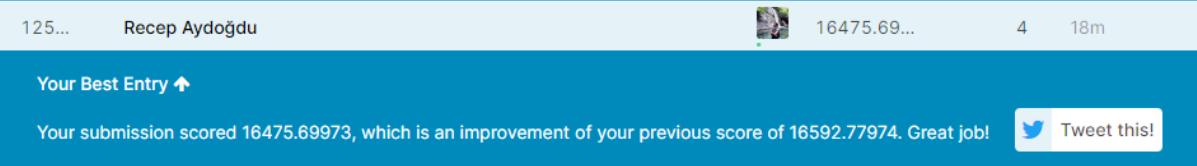
Şimdi, test verileriyle tahminler oluşturmak için eğitimli modelinizi kullanacaksınız.

In [9]:

```
# Preprocessing of test data, fit model  
preds_test = my_pipeline.predict(X_test) # Your code here  
  
# Check your answer  
step_2.check()
```

In [11]:

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



The screenshot shows a submission page for a machine learning competition. At the top, it displays the user's name, Recep Aydoğdu, and their score, 16475.69... The page also shows the number of upvotes (4) and the time since submission (18m). Below this, there is a message saying "Your Best Entry" with an upward arrow icon. A success message states: "Your submission scored 16475.69973, which is an improvement of your previous score of 16592.77974. Great job!" To the right of this message is a "Tweet this!" button with a Twitter icon.

Cross-Validation

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

Introduction

Makine öğrenmesi yinelemeli(iterative) bir süreçtir.

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

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

Ancak bu yaklaşımın bazı dezavantajları vardır.

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

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

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

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

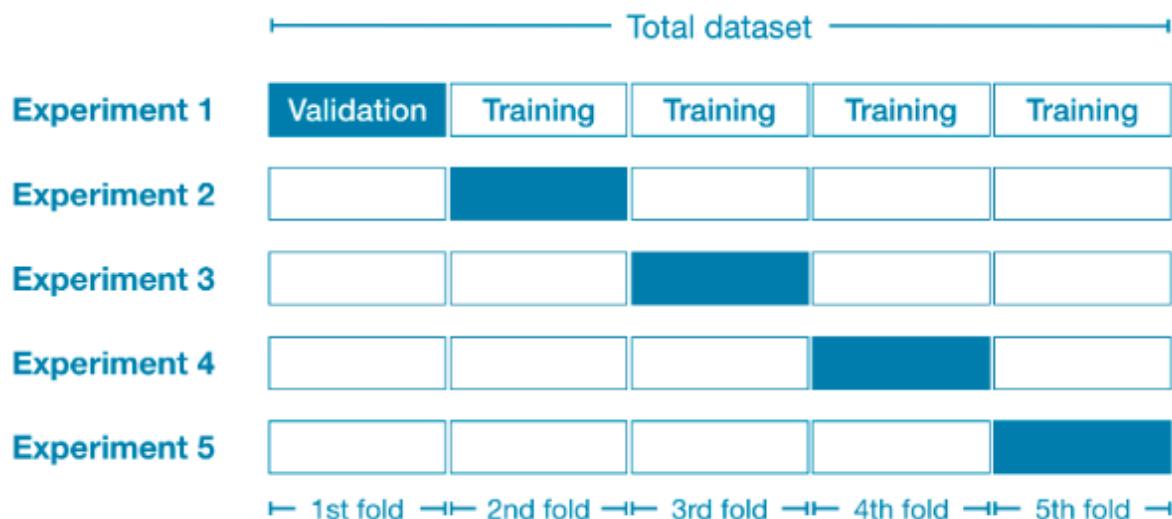
Genel olarak, validation seti ne kadar büyük olursa, model kalitesi ölçümümüzde o kadar az rastgelelik ("gürültü") olur ve o kadar güvenilir olur.

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

Cross-Validation Nedir?

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

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



Ardından, her fold için bir deneme gerçekleştiriyoruz:

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

Ne Zaman Cross-Validation Kullanmalıyız?

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

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

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

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

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

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

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

Example

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

```
In [1]:  
import pandas as pd  
  
# Read the data  
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')  
  
# Select subset of predictors  
cols_to_use = ['Rooms', 'Distance', 'Landsize', 'BuildingArea', 'YearBuilt']  
X = data[cols_to_use]  
  
# Select target  
y = data.Price
```

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

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

```
In [2]:  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.pipeline import Pipeline  
from sklearn.impute import SimpleImputer  
  
my_pipeline = Pipeline(steps=[('preprocessor', SimpleImputer()),  
                            ('model', RandomForestRegressor(n_estimators=50,  
                                              random_state=0))  
                           ])
```

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

```
In [3]:  
from sklearn.model_selection import cross_val_score  
  
# Multiply by -1 since sklearn calculates *negative* MAE  
scores = -1 * cross_val_score(my_pipeline, X, y,  
                             cv=5,  
                             scoring='neg_mean_absolute_error')  
  
print("MAE scores:\n", scores)
```

```
MAE scores:  
[301628.7893587 303164.4782723 287298.331666 236061.84754543  
 260383.45111427]
```

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

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

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

```
In [4]:  
print("Average MAE score (across experiments):")  
print(scores.mean())
```

```
Average MAE score (across experiments):  
277707.3795913405
```

Sonuç

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

Exercise: Cross-Validation

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

[Housing Prices Competition for Kaggle Learn Users](#) veri seti ile çalışacağız.



`_train`, `X_valid`, `y_train` ve `y_valid`'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti `X_test`'e yüklenir.

Basit olması için kategorik değişkenleri düşürüyoruz.

```
In [2]:  
import pandas as pd  
from sklearn.model_selection import train_test_split  
  
# Read the data  
train_data = pd.read_csv('../input/train.csv', index_col='Id')  
test_data = pd.read_csv('../input/test.csv', index_col='Id')  
  
# Remove rows with missing target, separate target from predictors  
train_data.dropna(axis=0, subset=['SalePrice'], inplace=True)  
y = train_data.SalePrice  
train_data.drop(['SalePrice'], axis=1, inplace=True)  
  
# Select numeric columns only  
numeric_cols = [cname for cname in train_data.columns if train_data[cname].dtype in ['int64',  
'float64']]  
X = train_data[numeric_cols].copy()  
X_test = test_data[numeric_cols].copy()
```

In [3]:

`x.head()`

Out[3]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
Id										
1	60	65.0	8450	7	5	2003	2003	196.0	706	0
2	20	80.0	9600	6	8	1976	1976	0.0	978	0
3	60	68.0	11250	7	5	2001	2002	162.0	486	0
4	70	60.0	9550	7	5	1915	1970	0.0	216	0
5	60	84.0	14260	8	5	2000	2000	350.0	655	0

5 rows x 36 columns

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

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

Random Forest modelindeki ağaç sayısını `n_estimators` parametresi ile ayarladık ve `random_state` ayarı tekrarlanabilirliği sağlıyor.

```
In [4]:  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.pipeline import Pipeline  
from sklearn.impute import SimpleImputer  
  
my_pipeline = Pipeline(steps=[  
    ('preprocessor', SimpleImputer()),  
    ('model', RandomForestRegressor(n_estimators=50, random_state=0))  
])
```

Cross-validation'da pipeline'ların nasıl kullanılacağını da öğrendiniz.

Aşağıdaki kod, beş farklı fold arasında ortalaması alınmış ortalama mutlak hatayı (MAE) elde etmek için `cross_val_score()` işlevini kullanır.

Fold sayısını `cv` parametresi ile ayarladığımızı hatırlayın.

```
In [5]:  
from sklearn.model_selection import cross_val_score  
  
# Multiply by -1 since sklearn calculates *negative* MAE  
scores = -1 * cross_val_score(my_pipeline, X, y,  
                             cv=5,  
                             scoring='neg_mean_absolute_error')  
  
print("Average MAE score:", scores.mean())
```

```
Average MAE score: 18276.410356164386
```

Step 1: Write a Usefull Function

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

Aşağıdakileri kullanan bir makine öğrenimi pipeline'nın MAE ortalamalarını bildiren (3 fold olacak) bir get_score () işlevi yazarak başlayın:

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

Get_score() öğesine sağlanan n_estimators parametresi, Random Forest modelindeki ağaç sayısını ayarlanırken kullanılır.

```
In [6]:  
def get_score(n_estimators):  
    my_pipeline = Pipeline(steps = [{"preprocessor": SimpleImputer()},  
                                    {"model": RandomForestRegressor(n_estimators, random_state=  
0)})  
    scores = -1 * cross_val_score(my_pipeline, X, y, cv=3, scoring="neg_mean_absolute_error")  
  
    return scores.mean()  
  
# Check your answer  
step_1.check()
```

İpucu: Pipeline sınıfıyla bir pipeline yaparak başlayın. RandomForestRegressor () içindeki n_estimators değerini get_score işlevine sağlanan bağımsız değişkene ayarladığınızdan emin olun.

Ardından, her fold için MAE'yi almak için cross_val_score() kullanın ve ortalamayı alın. Cv parametresi üzerinden fold sayısını üye ayarladığınızdan emin olun.

Step 2: Test Different Parameter Values

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

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

```
In [8]:
```

```
results = {}  
  
for i in range(1,9):  
    results[50*i] = get_score(50*i)  
# Check your answer  
step_2.check()
```

```
In [9]:
```

```
results
```

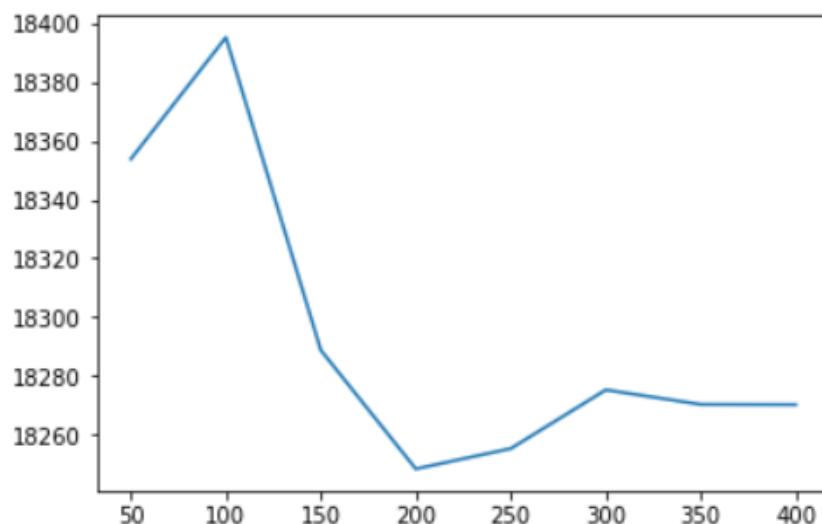
```
Out[9]:
```

```
{50: 18353.8393511688,  
100: 18395.2151680032,  
150: 18288.730020956387,  
200: 18248.345889801505,  
250: 18255.26922247291,  
300: 18275.241922621914,  
350: 18270.29183308043,  
400: 18270.197974402367}
```

Step 3: Find the Best Parameter Value

```
In [11]:
```

```
import matplotlib.pyplot as plt  
%matplotlib inline  
  
plt.plot(list(results.keys()), list(results.values()))  
plt.show()
```



Sonuçlar göz önüne alındığında, *n_estimators* için hangi değer Random Forest modeli için en iyisi olarak görünüyor? Cevabınızı *n_estimators_best* değerini ayarlamak için kullanın.

In [12]:

```
n_estimators_best = min(results, key=results.get)

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

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

[hyperparameter optimization](#) hakkında daha fazla bilgi edinmek isterseniz, bir makine öğrenimi modeli için en iyi parametre kombinasyonunu belirlemek için basit bir yöntem olan **Grid Search** ile başlamanız önerilir.

Neyse ki, scikit-learn, Grid Search kodunuzu çok verimli hale getirebilen yerleşik bir işlev olan [GridSearchCV\(\)](#) içerir!

Çeşitli veri kümelerinde son teknoloji sonuçlar elde eden güçlü bir teknik olan [gradient boosting](#) hakkında bilgi edinmeye devam edin.

XGBoost

Structured veriler için en doğru sonuçları veren modelleme tekniği.

Bu bölümde, **gradient boosting** modellerinin nasıl oluşturulacağını ve optimize edileceğini öğreneceksiniz.

Bu yöntem birçok Kaggle yarışmasında liderdir ve çeşitli veri kümelerinde ustalık derecesinde sonuçlar elde eder.

Introduction

Bu kursun çoğu bölümünde, birçok Decision Tree'nin tahminlerini ortalayarak tek bir Decision Tree'den daha iyi performans elde eden Random Forest yöntemiyle tahminler yaptınız.

Random Forest yöntemini "**ensemble method** (topluluk yöntemi)" olarak adlandırıyoruz.

Tanıma göre, ensemble(topluluk) metodları birkaç modelin tahminlerini birleştirir (örneğin, Random Forest durumunda birkaç ağaç).

Şimdi, gradient boosting adı verilen başka bir topluluk yöntemini öğreneceğiz.

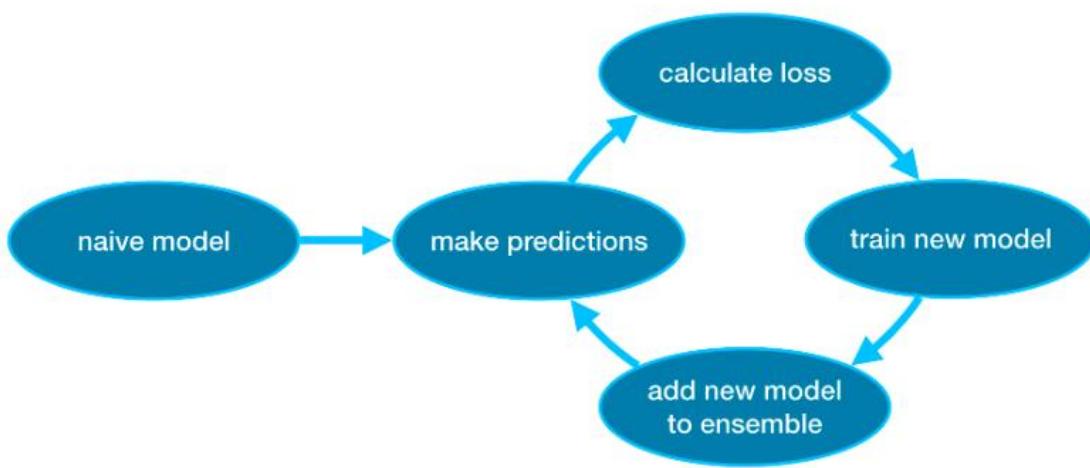
Gradient Boosting

Gradient Boosting, bir ensemble(topluluk)'a tekrarlanan modelleri eklemek için döngülerden geçen bir yöntemdir.

Topluluğun tahminleri oldukça saf olabilen tek bir modelle başlatılmasıyla başlar. (Tahminleri çılgınca yanlış olsa bile, topluluğa daha sonraki eklemeler bu hataları ele alacaktır.)

Sonra döngüye başlıyoruz:

- İlk olarak, veri grubundaki her bir gözlem için tahminler oluşturmak üzere mevcut topluluğu kullanıyoruz. Bir tahmin yapmak için, topluluktaki tüm modellerden tahminleri ekliyoruz.
- Bu tahminler bir loss(kayıp) fonksiyonunu hesaplamak için kullanılır (örneğin [mean squared error](#) gibi).
- Daha sonra, loss fonksiyonunu topluluğa eklenecek yeni bir modele uyacak şekilde kullanıyoruz. Özellikle, model parametrelerini belirleriz, böylece bu yeni modeli topluluğa eklemek kaybı azaltır. (Yan not: "gradient boosting" içindeki "gradyan", bu yeni modeldeki parametreleri belirlemek için loss fonksiyonunda [gradient descent](#) kullanacağımız anlamına gelir.)
- Son olarak, topluluğa yeni modeli ekliyoruz ve ...
- ... Tekrar!!



Example

Eğitim ve doğrulama verilerini X_train, X_valid, y_train ve y_valid'e yükleyerek başlıyoruz.

```
In [1]:  
import pandas as pd  
from sklearn.model_selection import train_test_split  
  
# Read the data  
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')  
  
# Select subset of predictors  
cols_to_use = ['Rooms', 'Distance', 'Landsize', 'BuildingArea', 'YearBuilt']  
X = data[cols_to_use]  
  
# Select target  
y = data.Price  
  
# Separate data into training and validation sets  
X_train, X_valid, y_train, y_valid = train_test_split(X, y)
```

Bu örnekte, XGBoost kütüphanesi ile çalışacaksınız. **XGBoost, extreme gradient boosting** (aşırı eğim yükseltme) anlamına gelir. Bu, performans ve hızı odaklanan çeşitli ek özelliklerle bir gradient boosting uygulamasıdır. (Scikit-learn'de gradient boosting'in başka bir versiyonu vardır, ancak XGBoost'un bazı teknik avantajları vardır.)

Bir sonraki kod hücresinde, `XGBoost (xgboost.XGBRegressor)` için scikit-learn API'sini içe aktarıyoruz.

Bu, tipki scikit-learn'de yaptığımız gibi bir model oluşturmamıza ve fit etmemize olanak tanır.

Çıktıda göreceğiniz gibi, `XGBRegressor` sınıfının birçok ayarlanabilir parametresi vardır - yakında bunları öğreneceksiniz!

In [2]:

```
from xgboost import XGBRegressor

my_model = XGBRegressor()
my_model.fit(X_train, y_train)
```

```
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarning: Serie
s.base is deprecated and will be removed in a future version
    if getattr(data, 'base', None) is not None and \
[13:37:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear i
s now deprecated in favor of reg:squarederror.
```

Out[2]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

Ayrıca tahminlerde bulunur ve modeli değerlendiririz.

In [3]:

```
from sklearn.metrics import mean_absolute_error

predictions = my_model.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 280355.04334039026
```

Parameter Tuning (Parametre Ayarı)

XGBoost, doğruluğu ve eğitim hızını önemli ölçüde etkileyebilecek birkaç parametreye sahiptir.

Anlamanız gereken ilk parametreler:

n_estimators

n_estimators, yukarıda açıklanan modelleme döngüsünden kaç kez geçileceğini belirler.

Topluluğa dahil ettigimiz model sayısına eşittir.

- Çok düşük bir değer *underfitting*'e neden olur, bu da hem eğitim verileri hem de test verileri üzerinde yanlış tahminlere yol açar.
- Çok yüksek bir değer, *overfitting*'e neden olur, bu da eğitim verileri üzerinde doğru tahminlere neden olur, ancak test verileri üzerinde yanlış tahminler yapar (bu bizim için önemli olan şeydir).

Tipik değerler 100-1000 arasındadır, ancak bu aşağıda tartışılan *learning_rate* parametresine çok bağlıdır.

Topluluktaki model sayısını ayarlamak için kod:

In [4]:

```
my_model = XGBRegressor(n_estimators=500)
my_model.fit(X_train, y_train)
```

early_stopping_rounds

early_stopping_rounds, *n_estimators* için ideal değeri otomatik olarak bulmanın bir yolunu sunar.

Early, *n_estimators* için durmak zorunda olmamamıza rağmen, doğrulama skoru iyileşmeyi bıraktığında modelin yinelemeyi durdurmasına neden olur.

n_estimators için yüksek bir değer ayırmak ve ardından yinelemeyi durdurmak için en uygun zamanı bulmak için *early_stopping_rounds* kullanmak akıllıcadır.

İş şansa bırakmanın bazen validation puanlarının iyileşmediği tek bir round'a denk geldiğinde döngüyü durdurması için, durmadan önce kaç tane doğrusal bozulma round'una izin vereceğinizi bir sayı belirtmeniz gereklidir.

early_stopping_rounds = 5 ayarı makul bir seçimdir.

Bu durumda, 5 doğrusal round boyunca kötüleşen doğrulama skorundan sonra duruyoruz.

Early_stopping_rounds kullanırken, validation puanlarını hesaplamak için bazı verileri de ayırmamanız gereklidir - bu, *eval_set* parametresini ayarlayarak yapılır.

Yukarıda yazdığımız kod örneğini, early stopping rounds'u içerecek şekilde değiştirebiliriz:

```
In [5]:  
my_model = XGBRegressor(n_estimators=500)  
my_model.fit(X_train, y_train,  
             early_stopping_rounds=5,  
             eval_set=[(X_valid, y_valid)],  
             verbose=False)  
  
[13:37:07] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.  
  
Out[5]:  
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,  
             colsample_bynode=1, colsample_bytree=1, gamma=0,  
             importance_type='gain', learning_rate=0.1, max_delta_step=0,  
             max_depth=3, min_child_weight=1, missing=None, n_estimators=500,  
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,  
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,  
             silent=None, subsample=1, verbosity=1)
```

Daha sonra tüm verilerinizle bir model fit etmek istiyorsanız, *n_estimators*'ı early stopping ile çalıştırıldığınızda en uygun bulduğunuz değere ayarlayın. Bu örneğimizde 500 bulunmuş.

learning_rate

Her bileşen modelinden tahminleri toplayarak tahminler almak yerine, eklenmeden önce her modelden gelen tahminleri küçük bir sayı ile (**learning rate** olarak bilinir) çarpabiliriz.

Bu, topluluğa eklediğimiz her ağacın bize daha az yardımcı olduğu anlamına gelir.

Bu nedenle, *n_estimators* için *overfitting* olmadan daha yüksek bir değer ayarlayabiliriz.

Early stopping kullanırsak, uygun sayıda ağaç otomatik olarak belirlenir.

Genel olarak, küçük bir learning rate ve çok sayıda tahminci ağaç daha doğru XGBoost modelleri verecektir, ancak döngü boyunca daha fazla yineleme yaptığı için modelin eğitilmesi daha uzun sürecektir.

Varsayılan olarak, XGBoost *learning_rate = 0.1* değerini ayarlar.

Learning rate'i değiştirmek için yukarıdaki örneği değiştirelim:

```
In [6]:
```

```
my_model = XGBRegressor(n_estimators=1000, learning_rate=0.05)
my_model.fit(X_train, y_train,
             early_stopping_rounds=5,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```

```
[13:37:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
Out[6]:
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=1000,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

n_jobs

Çalışma zamanının dikkate alındığı daha büyük veri kümelerinde, modellerinizi daha hızlı oluşturmak için parallelism (paralellik) kullanabilirsiniz.

n_jobs parametresini makinenizdeki çekirdek sayısına eşit olarak ayarlamak yaygındır.

Daha küçük veri kümelerinde bu pek yardımcı olmaz.

Ortaya çıkan model daha iyi olmayacağından, bu nedenle uygun zaman için mikro optimizasyon genellikle dikkat dağıtıcı bir şey değildir. Ancak, fit komutu sırasında uzun süre bekleyeceğiniz büyük veri kümelerinde yararlıdır.

Değiştirilmiş örnek:

```
In [7]:
```

```
my_model = XGBRegressor(n_estimators=1000, learning_rate=0.05, n_jobs=4)
my_model.fit(X_train, y_train,
             early_stopping_rounds=5,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```

Sonuç

[XGBoost](#), standart tablo halindeki verilerle (görüntü ve video gibi daha egzotik veri türlerinin aksine Pandas DataFrames'da depoladığınız veri türü) çalışmak için önde gelen bir yazılım kütüphanesidir.

Dikkatli parameter tuning ile son derece hassas modelleri eğitebilirisiniz.

Exercise: XGBoost

Bu alıştırmada, yeni bilgilerinizi **gradient boosting** modeli eğitmek için kullanacaksınız.

[Housing Prices Competition for Kaggle Learn Users](#) veri seti üzerinde çalışacağız.



X_train, X_valid, y_train ve y_valid'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X_test'e yüklenir.

```
In [2]:  
import pandas as pd  
from sklearn.model_selection import train_test_split  
  
# Read the data  
X = 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.dropna(axis=0, subset=['SalePrice'], inplace=True)  
y = X.SalePrice  
X.drop(['SalePrice'], axis=1, inplace=True)  
  
# Break off validation set from training data  
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)  
  
# "Cardinality" means the number of unique values in a column  
# Select categorical columns with relatively low cardinality (convenient but arbitrary)  
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 numeric columns  
numeric_cols = [cname for cname in X_train_full.columns if X_train_full[cname].dtype in ['int6  
4', 'float64']]  
  
# Keep selected columns only  
my_cols = low_cardinality_cols + numeric_cols  
X_train = X_train_full[my_cols].copy()  
X_valid = X_valid_full[my_cols].copy()  
X_test = X_test_full[my_cols].copy()  
  
# One-hot encode the data (to shorten the code, we use pandas)  
X_train = pd.get_dummies(X_train)  
X_valid = pd.get_dummies(X_valid)  
X_test = pd.get_dummies(X_test)  
X_train, X_valid = X_train.align(X_valid, join='left', axis=1)  
X_train, X_test = X_train.align(X_test, join='left', axis=1)
```

Step 1: Model Oluşturun

Bu adımda, gradient boosting ile ilk modelinizi oluşturacak ve eğiteceksiniz.

- My_model_1 öğesini bir **XGBoost** modeline ayarlayarak başlayın. XGBRegressor sınıfını kullanın ve random seed'i 0 olarak ayarlayın (*random_state = 0*). Diğer tüm parametreleri varsayılan olarak bırakın.
- X_train ve y_train ile modelinizi fit edin.

In [3]:

```
from xgboost import XGBRegressor

# Define the model
my_model_1 = XGBRegressor(random_state=0) # Your code here

# Fit the model
my_model_1.fit(X_train, y_train) # Your code here

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

Modelin validation verileri için tahminlerini *predictions_1*'de tutun. Validation verilerinin *X_valid*'de saklandığını hatırlayın.

In [5]:

```
from sklearn.metrics import mean_absolute_error

# Get predictions
predictions_1 = my_model_1.predict(X_valid) # Your code here

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

Son olarak, validation verilerinin tahminlerine karşılık gelen ortalama mutlak hatayı (MAE) hesaplamak için *mean_absolute_error ()* işlevini kullanın. Validation verilerinin doğru sonuçlarının *y_valid* içinde saklandığını unutmayın.

In [7]:

```
# Calculate MAE
mae_1 = mean_absolute_error(y_valid, predictions_1) # Your code here

# Uncomment to print MAE
print("Mean Absolute Error: " , mae_1)

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

Mean Absolute Error: 17662.736729452055

Step 2: Modelinizi İyileştirin

Artık varsayılan bir modeli temel olarak eğittiğinize göre, daha iyi performans elde edip edemeyeceğinizi görmek için parametreleri değiştirmenin zamanı geldi!

- XGBRegressor sınıfını kullanarak `my_model_2` ögesini bir XGBoost modeline ayarlayarak başlayın. Daha iyi sonuçlar almak için varsayılan parametreleri (`n_estimators` ve `learning_rate` gibi) nasıl değiştireceğinizi öğrenmek için önceki bölümde öğretiklerinizi kullanın.
- Ardından, modeli `X_train` ve `y_train`'deki training verileri ile fit edin.
- Modelin validation verileri için tahminlerini `predictions_2`'de tutun. Validation verilerinin `X_valid`'de saklandığını hatırlayın.
- Son olarak, validation verilerinin tahminlerine karşılık gelen ortalama mutlak hatayı (MAE) hesaplamak için `mean_absolute_error ()` işlevini kullanın. Validation verilerinin doğru sonuçlarının `y_valid` içinde saklandığını unutmayın.

```
In [9]: # Define the model
my_model_2 = XGBRegressor(n_estimators=500, learning_rate=0.05) # Your code here

# Fit the model
my_model_2.fit(X_train, y_train) # Your code here

# Get predictions
predictions_2 = my_model_2.predict(X_valid) # Your code here

# Calculate MAE
mae_2 = mean_absolute_error(y_valid, predictions_2) # Your code here

# Uncomment to print MAE
print("Mean Absolute Error:" , mae_2)

# Check your answer
step_2.check()

Mean Absolute Error: 16728.27523009418
```

Step 3: Modeli Kırın

Bu adımda, 1. Adımdaki orijinal modelden daha kötü performans gösteren bir model oluşturacaksınız. Bu, parametreleri nasıl ayarlayacağınızı dair sezginizi geliştirmenize yardımcı olacaktır.

Kazara daha iyi performans elde ettiğinizi bile görebilirsiniz, bu da sonuçta değerli bir öğrenme deneyimi!

```
In [11]:  
# Define the model  
my_model_3 = XGBRegressor(n_estimators=500, learning_rate=1)  
  
# Fit the model  
my_model_3.fit(X_train, y_train) # Your code here  
  
# Get predictions  
predictions_3 = my_model_3.predict(X_valid)  
  
# Calculate MAE  
mae_3 = mean_absolute_error(y_valid, predictions_3)  
  
# Uncomment to print MAE  
print("Mean Absolute Error:" , mae_3)  
  
# Check your answer  
step_3.check()
```

```
Mean Absolute Error: 27386.61764233733
```

Data Leakage (Veri Sızıntısı)

Bu bölümde, **Data Leakage**(Veri Sızıntısı)'nın ne olduğunu ve nasıl önleneceğini öğreneceksiniz. Bunu nasıl önleyeceğinizi bilmiyorsanız, sizinti sık sık ortaya çıkacak ve modellerinizi ince ve tehlikeli yollarla mahvedecektir.

Bu, veri bilimcilerin uygulamaları için en önemli kavamlardan biridir.

Introduction

Data Leakage (veri sızıntısı), training verileriniz target hakkında bilgi içerdiginde gerçekleşir, ancak model tahmin için kullanıldığından benzer veriler kullanılamaz.

Bu, training setinde (ve hatta muhtemelen validation verilerinde) yüksek performansa yol açar, ancak model üretimde kötü performans gösterecektir.

Başa bir deyişle, sizinti, bir modelle karar vermeye başlayana kadar bir modelin doğru görünmesine neden olur ve sonra model çok yanlış bir hale gelir.

İki ana sızıntı türü vardır: **target leakage** ve **train-test contamination**.

Target Leakage

Target Leakage (Hedef Sızıntısı), öngörücüleriniz, tahmin yaptığınız sırada kullanılamayacak veriler içerdiginde ortaya çıkar.

Target Leakage'ı, yalnızca bir özelliğin iyi tahminlerde bulunmasına yardımcı olup olmadığı değil, verilerin kullanılabilir hale geldiği zamanlama veya kronolojik sıraya göre düşünmek önemlidir.

Bir örnek anlamamıza yardımcı olacaktır. Pneumonia ile kimin hastalanacağını tahmin etmek istedığınızı düşünün. Ham verilerinizin ilk birkaç satır şöyledir:

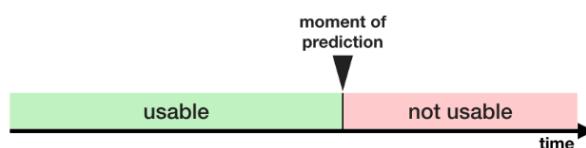
got_pneumonia	age	weight	male	took_antibiotic_medicine	...
False	65	100	False	False	...
False	72	130	True	False	...
True	58	100	False	True	...

İnsanlar pnömoni olduktan sonra iyileşmek için antibiotik ilaçlar alırlar. Ham veriler, bu sütunlar arasında güçlü bir ilişki olduğunu gösterir, ancak *got_pneumonia* değeri belirlendikten sonra *took_antibiotic_medicine* sıklıkla değiştirilir. Bu target leakage'dır.

Model, *took_antibiotic_medicine* için *False* değerine sahip olan herkesin pnömonisi olmadığını görecektir. Validation verileri training verileriyle aynı kaynaktan geldiğinden, pattern, validation'da kendini tekrar edecektir ve modelin büyük validation (veya cross-validation'da) puanları olacaktır.

Ancak, model gerçek dünyada kullanımına geçtiğinde çok büyük yanlışlar yapacaktır. Çünkü pnömoni olan hastalar tedaviye başlamadan önce antibiotik almamış olacaktır.

Bu tür veri sızıntısını önlemek için, hedef değer gerçekleştikten sonra güncellenen (veya oluşturulan) değişkenler hariç tutulmalıdır.



Train-Test Contamination

Training verilerini, validation verilerinden ayırmaya dikkat etmediğinizde farklı bir sızıntı türü oluşur.

Validation'ın modelin daha önce dikkate almadığı veriler üzerinde nasıl bir performans gösterdiğini hatırlayın. Validation verileri preprocessing davranışını etkiliyorsa bu işlemi ince yollarla bozabilirsiniz. Buna bazen **train-test contamination** denir.

Örneğin, *train_test_split()* öğesini çağrımadan önce önisleme yaptığınızı (eksik değerler için imputer kullanmak gibi) düşünün. Sonuç ne oldu? Modeliniz iyi validation puanları alabilir, bu da size büyük güven verir, ancak karar vermek için uyguladığınızda düşük performans gösterir.

Doğrulamanız basit bir train-test split'e dayanıyorsa, validation verilerini preprocessing adımlarının uygulanması da dahil olmak üzere her tür fitting işleminden hariç tutun.

Scikit-learn pipeline'i kullanıyorsanız bu daha kolaydır. Cross-validation kullanırken, preproccesing'i pipeline içinde yapmanız daha da önemlidir!

Example

Bu örnekte, target leakage'ı tespit etmenin ve kaldırmanın bir yolunu öğreneceksiniz.

Kredi kartı uygulamaları hakkında bir veri kümesi kullanacağız.

Sonuç olarak, her kredi kartı uygulamasılarındaki bilgi bir *X* dataframe'inde saklanır. Bir *y* serisini de hangi uygulamaların kabul edildiğini tahmin etmek için kullanacağız.

```
In [1]:  
import pandas as pd  
  
# Read the data  
data = pd.read_csv('../input/aer-credit-card-data/AER_credit_card_data.csv',  
                    true_values = ['yes'], false_values = ['no'])  
  
# Select target  
y = data.card  
  
# Select predictors  
X = data.drop(['card'], axis=1)  
  
print("Number of rows in the dataset:", X.shape[0])  
X.head()  
  
Number of rows in the dataset: 1319  
  
Out[1]:  


|   | reports | age      | income | share    | expenditure | owner | selfemp | dependents | months | majorcard |
|---|---------|----------|--------|----------|-------------|-------|---------|------------|--------|-----------|
| 0 | 0       | 37.66667 | 4.5200 | 0.033270 | 124.983300  | True  | False   | 3          | 54     | 1         |
| 1 | 0       | 33.25000 | 2.4200 | 0.005217 | 9.854167    | False | False   | 3          | 34     | 1         |
| 2 | 0       | 33.66667 | 4.5000 | 0.004156 | 15.000000   | True  | False   | 4          | 58     | 1         |
| 3 | 0       | 30.50000 | 2.5400 | 0.065214 | 137.869200  | False | False   | 0          | 25     | 1         |
| 4 | 0       | 32.16667 | 9.7867 | 0.067051 | 546.503300  | True  | False   | 2          | 64     | 1         |


```

Bu küçük bir veri kümesi olduğundan, model kalitesinin doğru ölçümlerini sağlamak için çapraz doğrulamayı kullanacağız.

```
In [2]:  
from sklearn.pipeline import make_pipeline  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model_selection import cross_val_score  
  
# Since there is no preprocessing, we don't need a pipeline (used anyway as best practice!)  
my_pipeline = make_pipeline(RandomForestClassifier(n_estimators=100))  
cv_scores = cross_val_score(my_pipeline, X, y,  
                           cv=5,  
                           scoring='accuracy')  
  
print("Cross-validation accuracy: %f" % cv_scores.mean())  
  
Cross-validation accuracy: 0.979525
```

Deneyim kazandıkça, % 98 doğruluk veren modeller bulmanın çok nadir olduğunu göreceksiniz. Bu olur, ancak verileri target leakage açısından daha yakından incelemeliyiz.

Data sekmesi altında da bulabileceğiniz verilerin bir özeti:

card: Kredi başvurusu kabul edilirse 1, edilmezse 0

reports: Başlıca küçültücü raporların sayısı.(Kredi kabulunu etkiler.)

age: n(yaş) + yılın onikide biri

income: Yıllık gelir (10.000'e bölünür)

share: Aylık kredi kartı harcamalarının yıllık gelire oranı

expenditure: Ortalama aylık kredi kartı harcaması

owner: Ev sahibi ise 1, ev kiralıyorsa 0

selfempl: Serbest meslek sahibi ise 1, değilse 0

dependents: 1 + bakmakla yükümlü kişi sayısı

months: Geçerli adreste yaşanan ay sayısı

majorcards: Sahip olunan kredi kartı sayısı

active: Etkin kredi hesabı sayısı

Birkaç değişken şüpheli görünüyor. Örneğin, expenditure bu kartta veya uygulamadan önce kullanılan kartlarda yapılan harcama anlamına mı geliyor?

Bu noktada, temel veri karşılaştırmaları çok yardımcı olabilir:

```
In [3]:
expenditures_cardholders = X.expenditure[y]
expenditures_noncardholders = X.expenditure[~y]

print('Fraction of those who did not receive a card and had no expenditures: %.2f' \
      %((expenditures_noncardholders == 0).mean()))
print('Fraction of those who received a card and had no expenditures: %.2f' \
      %((expenditures_cardholders == 0).mean()))

Fraction of those who did not receive a card and had no expenditures: 1.00
Fraction of those who received a card and had no expenditures: 0.02
```

Yukarıda gösterildiği gibi, kart almayan herkesin harcamaları yoktu, kart alanların sadece % 2'sinin harcamaları yoktu. Modelimizin yüksek bir doğruluğa sahip olması şaşırtıcı değil.

Ancak bu, harcamaların muhtemelen başvurdukları karttaki harcamalar anlamına geldiği bir target leakage durumu gibi görünmektedir.

Share kısmen harcama ile belirlendiğinden, hariç tutulmalıdır.

Active ve *majorcard* değişkenleri biraz daha az açıktır, ancak açıklamaya ilgili görünüyorlar.

Çoğu durumda, daha fazla bilgi edinmek için verileri oluşturan kişileri izleyemiyorsanız, üzülmektense güvende olmak daha iyidir.

Target Leakage olmayan bir modeli şu şekilde çalıştırırız:

```
In [4]:
# Drop leaky predictors from dataset
potential_leaks = ['expenditure', 'share', 'active', 'majorcards']
X2 = X.drop(potential_leaks, axis=1)

# Evaluate the model with leaky predictors removed
cv_scores = cross_val_score(my_pipeline, X2, y,
                             cv=5,
                             scoring='accuracy')

print("Cross-val accuracy: %f" % cv_scores.mean())

Cross-val accuracy: 0.830924
```

Burada accuracy biraz daha düşük, bu da hayal kırıklığı yaratabilir.

Bununla birlikte, yeni uygulamalarda kullanıldığı zaman yaklaşık % 80'inin doğru olmasını bekleyebiliriz, oysa leaky(sızdırılan) model muhtemelen bundan daha kötü sonuç verecektir (cross-validation'daki yüksek görünen puanına rağmen).

Sonuç

Veri sizintisi, birçok veri bilimi uygulamasında milyonlarca dolarlık bir hataya sebep olabilir. Eğitim ve doğrulama verilerinin dikkatlice ayrılması, train-test kontaminasyonunu önleyebilir ve pipeline'lar bu ayrılmanın uygulanmasına yardımcı olabilir.

Aynı şekilde, dikkatli olma, sağduyu ve veri keşfi birleşimi de target leakage'ı belirlemeye yardımcı olabilir.

Bu hala soyut görünebilir. Target Leakage ve train-test kontaminasyonunu belirleme becerinizi geliştirmek için aşağıdaki alıştırmadaki örnekleri gözden geçirmeyi deneyin!

Exercise: Data Leakage

<https://www.kaggle.com/recepaydogdu/exercise-data-leakage>

Data Visualization

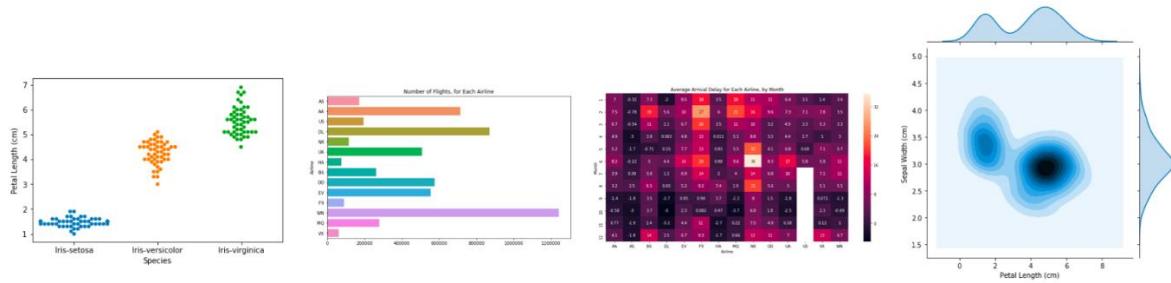
Hello, Seaborn

Bu uygulamalı mikro kursta, güçlü ama kullanımı kolay bir veri görselleştirme aracı olan [seaborn](#) ile veri görselleştirmelerinizi bir sonraki seviyeye nasıl taşıyacağınızı öğreneceksiniz.

Seaborn'u kullanmak için, popüler bir programlama dili olan Python'da kod yazmayı da öğreneceksiniz. Bahsedilen,

- mikro-kurs, önceden programlama deneyimi olmayanlara yöneliktir ve
- her grafik kısa ve basit kod kullanır, bu da seaborn'u diğer birçok veri görselleştirme aracından (örneğin Excel gibi) çok daha hızlı ve kolay hale getirir.

Yani, daha önce hiç bir kod satırı yazmamışsanız ve bugün daha hızlı, daha çekici grafikler yapmaya başlamak için asgari olanı öğrenmek istiyorsanız, doğru yerdesiniz! Yapacağınız grafiklerden bazlarına göz atmak için aşağıdaki figürlere göz atın.



Notebook Kurulumu

Kodlama ortamınızı ayarlamak için her notebook'un üzerinde çalıştırmanız gereken birkaç kod satırı vardır. Bu kod satırlarını anlamanız şuan önemli değil ve bu yüzden henüz ayrıntılara girmeyeceğiz.

```
In [1]:  
    import pandas as pd  
    pd.plotting.register_matplotlib_converters()  
    import matplotlib.pyplot as plt  
    %matplotlib inline  
    import seaborn as sns  
    print("Setup Complete")
```

Setup Complete

Veri Yükleme

Bu notebook'da altı ülke için tarihi FIFA sıralaması veri setiyle çalışacağız: Arjantin (ARG), Brezilya (BRA), İspanya (ESP), Fransa (FRA), Almanya (GER) ve İtalya (ITA). Bu veriseti CSV dosyası olarak saklanır ([comma-separated values file](#) kısıtlaması). CSV dosyasını Excel'de açmak, her ülke için bir sütunla birlikte her tarih için bir satır gösterir.

A1	Date	ARG	BRA	ESP	FRA	GER	ITA
2	8/8/93	5	8	13	12	1	2
3	9/23/93	12	1	14	7	5	2
4	10/22/93	9	1	7	14	4	3
5	11/19/93	9	4	7	15	3	1
6	12/23/93	8	3	5	15	1	2
7	2/15/94	9	2	6	14	1	7
8	3/15/94	8	2	6	15	1	11
9	4/19/94	10	1	7	15	2	13

Verileri notebook'a yüklemek için aşağıdaki kod hücresına aşağıdaki gibi uygulanan iki ayrı adım kullanacağız:

- veri kümesine erişilebileceği konumu (veya [filepath](#)) belirterek başlayın ve ardından
- veri kümesinin içeriğini not defterine yüklemek için dosya yolunu kullanın.

In [2]:

```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

comments

executable code

Path of the file to read
fifa_filepath = "../input/fifa.csv"

Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)

filepath to dataset column to use as row labels recognize the row labels as dates

Verileri İnceleyelim

Şimdi, düzgün yükleniğinden emin olmak için `fifa_data`'daki veri kümesine hızlı bir şekilde bakacağız.

In [3]:

```
# Print the first 5 rows of the data
fifa_data.head()
```

Out[3]:

	ARG	BRA	ESP	FRA	GER	ITA
Date						
1993-08-08	5.0	8.0	13.0	12.0	1.0	2.0
1993-09-23	12.0	1.0	14.0	7.0	5.0	2.0
1993-10-22	9.0	1.0	7.0	14.0	4.0	3.0
1993-11-19	9.0	4.0	7.0	15.0	3.0	1.0
1993-12-23	8.0	3.0	5.0	15.0	1.0	2.0

İlk beş satırın yukarıdaki Excel görüntüsüyle aynı olduğunu kontrol edin.

Plot the Data (Verileri Çizin)

Bu kursta, birçok farklı plot türü ile ilgili bilgi edineceksiniz. Öğrendiklerinize bir göz atmak için aşağıdaki line chart(çizgi grafiği) oluşturan kodu inceleyin.

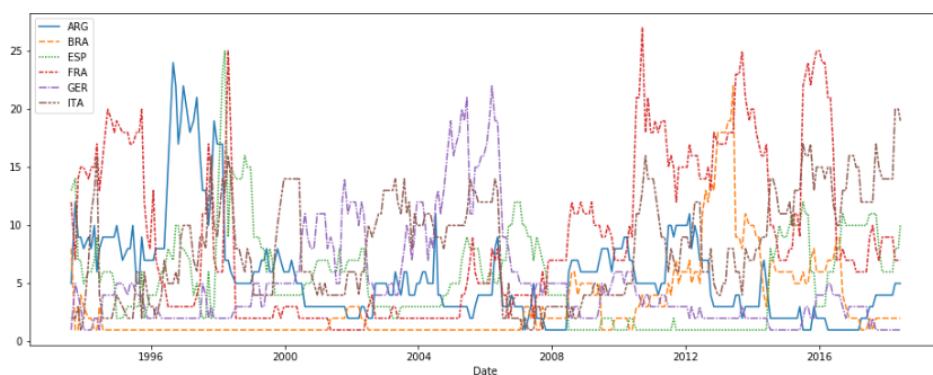
In [4]:

```
# Set the width and height of the figure
plt.figure(figsize=(16,6))

# Line chart showing how FIFA rankings evolved over time
sns.lineplot(data=fifa_data)
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22bfac3fd0>
```



Bu kod henüz anlam ifade etmiyor olabilir, ilerleyen eğitimlerde kod hakkında daha fazla bilgi edineceksiniz.

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>
- Kaggle – Data Visualization Course
<https://www.kaggle.com/learn/data-visualization>