

İçindekiler

İçindekiler	1
(Intro to Machine Learning) Makine Öğrenimine Giriş	4
How Models Work (Modeller Nasıl Çalışır):	4
Giriş:.....	4
Decision Tree'nin Geliştirilmesi	5
Basic Data Exploration (Temel Veri Keşfi)	6
Using Pandas to Get Familiar With Your Data (Verilerinizi Öğrenmek için Pandas kullanma):.....	6
Interpreting Data Description (Veri Açıklamalarını Yorumlama):.....	7
Excercise: Explore Your Data	7
Your First Machine Learning Model:	9
Selecting Data for Modeling (Modelleme için Veri Seçmek):.....	9
Selecting The Prediction Target (Tahmin Hedefini Seçme)	12
Choosing "Features" (Özellik Seçimi):	12
Building Your Model (Model Oluşturma):	14
Exercises: Your First Machine Learning Model.....	16
Model Validation(Model geçerliliği):.....	19
What is Model Validation: (Model Validation Nedir)	19
The Problem with "In-Sample" Scores("In-Sample(Örnek İçi)" Puanlarla İlgili Sorun).....	21
Coding It	21
Wow!	22
Exercises: Model Validation	22
Underfitting and Overfitting.....	24
Experimenting With Different Models	24
Examples:.....	26
Sonuç:	27
Exercise: Underfitting and Overfitting.....	28
Random Forests:.....	30
Giriş:.....	30
Example	30
Sonuç:	31
Exercises: Random Forest.....	31
Exercises: Machine Learning Competitions.....	33
Introduction.....	33
Creating a Model For the Competition	34

Make Predictions.....	34
Quiz: Intro to Machine Learning.....	35
Intermediate Machine Learning (Orta Düzey Makine Öğrenimi).....	44
Introduction (Giriş).....	44
Prerequisites (Önkoşullar).....	45
Your Turn(Sıra Sende).....	45
Exercise: Introduction	45
Setup.....	45
Missing Values (Eksik Değerler).....	49
Introduction (Giriş).....	49
Three Approaches	49
Example(Örnek).....	50
Exercises	53
Categorical Variables.....	58
Introduction.....	58
Üç Yaklaşım.....	58
Example	59
En iyi yaklaşım hangisi?	62
Sonuç	63
Exercises	63
Pipelines	70
Introduction.....	70
Example	70
Sonuç	73
Exercise: Pipelines	73
Cross-Validation	76
Introduction.....	76
Cross-Validation Nedir?.....	77
Ne Zaman Cross-Validation Kullanmalıyız?	78
Example	78
Sonuç	79
Exercise: Cross-Validation	79
XGBoost.....	83
Introduction.....	83
Gradient Boosting.....	83
Example	84

Parameter Tuning (Parametre Ayarı)	85
Sonuç	87
Exercise: XGBoost	87
Data Leakage (Veri Sızıntısı)	90
Introduction.....	90
Example	91
Sonuç	93
Exercise: Data Leakage	94
Quiz2:.....	97
KAYNAKLAR	103

(Intro to Machine Learning) Makine Öğrenimine Giriş

Makine öğrenmesindeki temel fikirleri öğrenin ve ilk modellerinizi oluşturun.

How Models Work (Modeller Nasıl Çalışır):

Giriş:

Makine öğrenimi modellerinin nasıl 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:

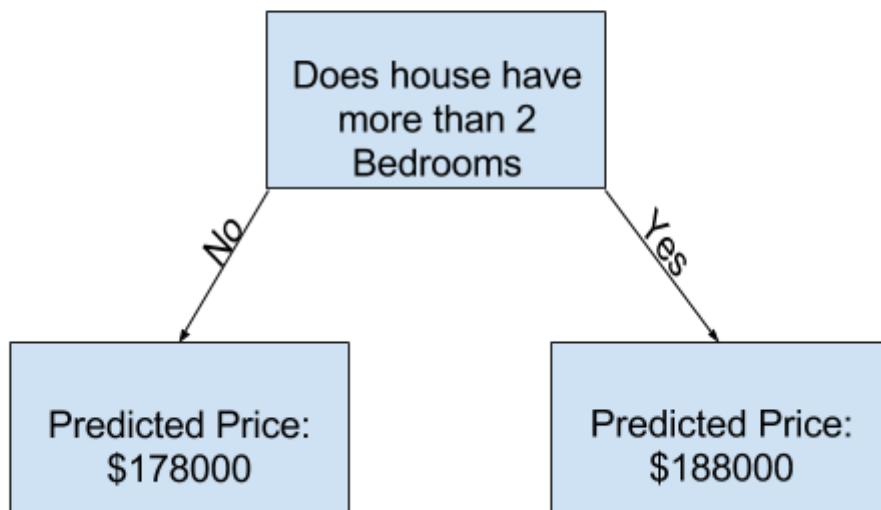
Kuzeniniz gayrimenkul konusunda speküasyonlarla milyonlarca dolar kazandı. Veri bilimine gösterdiğiniz ilgi nedeniyle sizinle iş ortağı olmayı teklif etti. Parayı tedarik edecek ve çeşitli evlerin ne kadar değerli olduğunu tahmin eden modeller sunacaksınız.

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

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

Basitlik için, mümkün olan en basit karar ağacıyla başlayacağız.

Sample Decision Tree



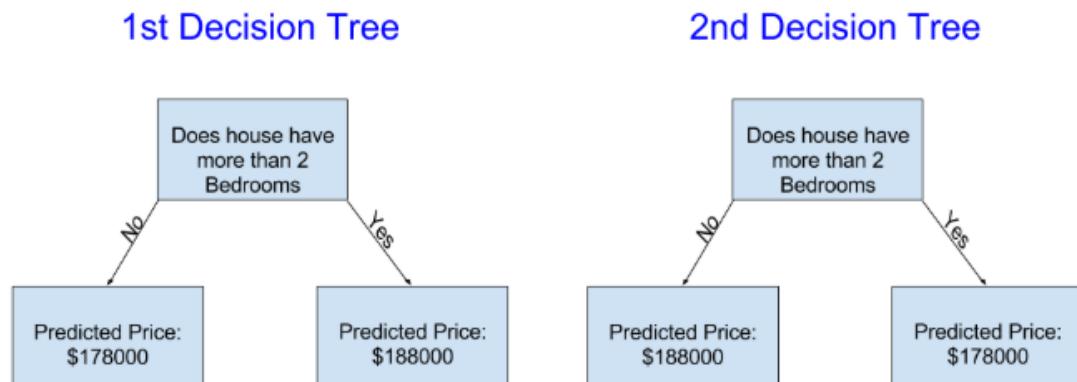
Evleri sadece iki kategoriye ayırır. Dikkate alınan herhangi bir ev için tahmini fiyat, aynı kategorideki evlerin tarihsel ortalama fiyatıdır.

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

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

Decision Tree'nin Geliştirilmesi

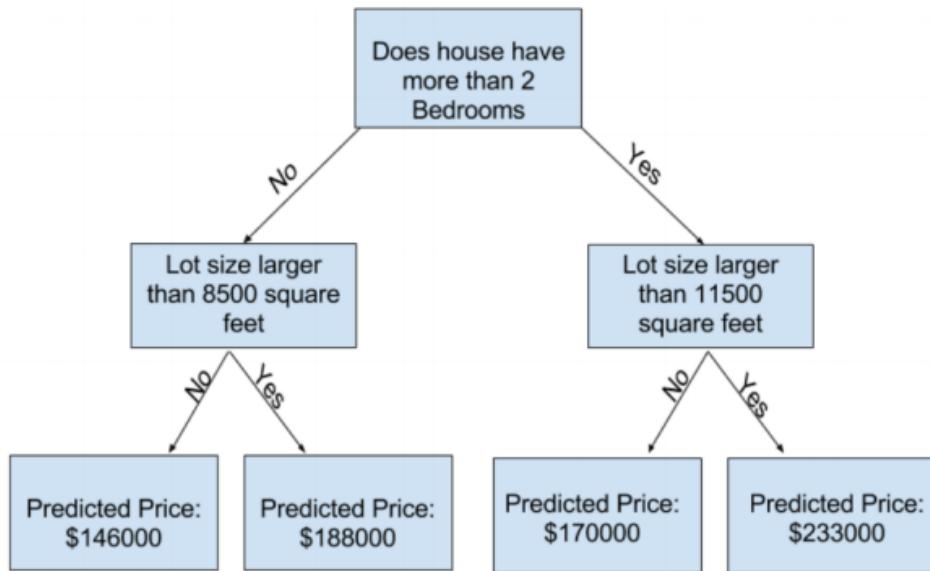
Aşağıdaki iki karardan hangisinin gayrimenkul eğitim verilerinin fit edilmesinden kaynaklanması daha olasıdır?



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

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

Daha fazla "**splits(bölme)**" olan bir ağaç kullanarak daha fazla faktör yakalayabilirsiniz. Bunlara "**deeper(daha derin)**" ağaçlar denir. Her evin toplam lot büyüklüğünü de dikkate alan bir karar ağacı şöyle görünebilir:



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

Ev için tahmini fiyat 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 (Temel Veri Keşfi)

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

Herhangi bir makine öğrenimi projesinin ilk adımı, verileri tanıtmaktır. Bunun için Pandas kütüphanesini kullanacaksınız. **Pandas**, bilim insanların verileri keşfetmek ve işlemek için kullandığı temel araçtır. Çoğu kişi **Pandas** kodlarında **pd** olarak kısaltılır. Bunu şu komutla yapıyoruz:

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

Pandas kütüphanesinin en önemli kısmı DataFrame'dir. Bir DataFrame, tablo olarak düşünebileceğiniz veri türünü tutar. Bu, Excel'deki bir sayfaya veya SQL veritabanındaki bir tabloya benzer.

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

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

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

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

Verileri aşağıdaki komutlarla yükler ve inceleriz:

- # kolay erişim için dosya yolunu değişkene kaydet
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
- # verileri okuyun ve DataFrame'de melbourne _data başlıklı verileri depolayın
melbourne_data = pd.read_csv(melbourne_file_path)

melbourne_data.describe()

[14]:	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Latitude	Longitude	Propertycount
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.000000	8205.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.416127	151.967650	1964.684217	-37.809203	144.995216	7454.417378
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990.669241	541.014538	37.273762	0.079260	0.103916	4378.581772
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000	-38.182550	144.431810	249.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000000	1940.000000	-37.856822	144.929600	4380.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000000	-37.802355	145.000100	6555.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.000000	1999.000000	-37.756400	145.058305	10331.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000	-37.408530	145.526350	21650.000000

Interpreting Data Description (Veri Açıklamalarını Yorumlama):

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

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

Eksik değerler birçok nedenden dolayı ortaya çıkar. Örneğin, 1 yatak odalı bir ev araştırılırken 2. yatak odasının boyutu toplanmaz. Eksik veriler konusuna geri döneceğiz.

İkinci değer, ortalama olan **mean**'dır.

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

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

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

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

Excercise: Explore Your Data

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

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

Kurs örnekleri Melbourne'den gelen verileri kullanır. Bu teknikleri kendi başınıza uygulayabilmeniz için, bunları yeni bir veri kümeye (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.

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

```
[10]:  
    # What is the average lot size (rounded to nearest integer)?  
    avg_lot_size = 10517  
  
    # As of today, how old is the newest home (current year - the date in which it was built)  
    newest_home_age = 10  
  
    # Checks your answers  
    step_2.check()
```

Correct

Verilerinizi Düşünün

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

- 1- Bu verilerin toplandığı yeni evler inşa etmediler.
- 2- Veriler uzun zaman önce toplanmıştır. Veri yayımından sonra inşa edilen evler görünmezdi.

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

Hangi açıklamanın daha mantıklı olduğunu görmek için verileri nasıl inceleyebilirsiniz?

Your First Machine Learning Model:

Selecting Data for Modeling (Modelleme için Veri Seçmek):

Veri kümenizin, kafanızda canlanması veya güzelce ekrana yazdırma için çok fazla değişkeni vardı. Bu başa çıkmaz veri miktarını anlayabileceğiniz bir şeye nasıl ayıracaksınız?

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.)

```
[1]: import pandas as pd  
melbourne_file_path='melb_data.csv'  
melbourne_data=pd.read_csv(melbourne_file_path)  
melbourne_data.columns  
  
[1]: Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',  
          'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',  
          'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Latitude',  
          'Longitude', 'Regionname', 'Propertycount'],  
          dtype='object')
```

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

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

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

```
# Daha sonraki bir derste eksik değerleri ele almayı öğreneceğiz.
```

```
# Lowa verileriniz, kullandığınız sütunlarda eksik değerlere sahip değildi.
```

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

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

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

Column'ların içinde kaçar tane eksik veri var ona baktık.

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

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

Verilerinizin bir alt kümelerini 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 **Seri**'de depolanır.

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

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

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

```
y=melbourne_data.Price  
y  
  
0      1480000.0  
1      1035000.0  
2      1465000.0  
3      850000.0  
4      1600000.0  
...  
13575    1245000.0  
13576    1031000.0  
13577    1170000.0  
13578    2500000.0  
13579    1285000.0  
Name: Price, Length: 13580, dtype: float64
```

Choosing "Features" (Özellik Seçimi):

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

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

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

Şimdilik, sadece birkaç özelliğe sahip bir model oluşturacağız.

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

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

Here is an example:

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

Kural olarak, bu verilere X denir.

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

```
[19]:
```

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

13580 rows × 5 columns

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

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

```
[20]:
```

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

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

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

Verilerinizi bu komutlarla görsel olarak kontrol etmek, bir veri bilim insanının işinin önemli bir parçasıdır. Veri kümesinde 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 ağacı mı? Başka bir model mi? Model tipinin diğer bazı parametreleri de belirtilir.
- **fit** : Sağlanan verilerden pattern(desen) yakalayın. Bu modellemenin kalbidir.
- **predict** : Tahmin
- **evaluate** : Modelin tahminlerinin ne kadar doğru olduğu belirleyin.

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

- Modeli tanımlayın. Her çalıştırımda aynı sonuçları sağlamak için random_state için bir sayı belirtin

```
[23]: from sklearn.tree import DecisionTreeRegressor  
  
#Modeli tanımlayın. Her çalıştırımda aynı sonuçları sağlamak için random_state için bir sayı belirtin  
melbourne_model=DecisionTreeRegressor(random_state=1)  
#Fit model  
melbourne_model.fit(X,y)  
  
[23]: DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,  
                           max_features=None, max_leaf_nodes=None,  
                           min_impurity_decrease=0.0, min_impurity_split=None,  
                           min_samples_leaf=1, min_samples_split=2,  
                           min_weight_fraction_leaf=0.0, presort='deprecated',  
                           random_state=1, splitter='best')
```

random_state: Kodu her çalıştığımızda aynı çıktıyı 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 çalıştığımızda aynı şekilde bölmüş olur ve aynı sonucu vermiş olur. Farklı değerler verdiğinde farklı sonuçlar aldığıni 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 çalıştığında 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 çalışmada 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 olmayıacaktır.

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

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

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

Making predictions for the following 5 houses:

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

The predictions are
[1480000. 1035000. 1465000. 850000. 1600000.]

Exercises: Your First Machine Learning Model

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

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

tahmin hedefinin adını bulmak için veri kümelerindeki sütunların listesini yazdır

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

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



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

Aşağıdaki satırlar size bir ipucu veya çözüm gösterecektir.

```
# step_1.hint()  
  
# step_1.solution()
```

Step 2: Create X

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

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

Listede yalnızca aşağıdaki sütunları kullanacaksınız :

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

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

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

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

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

Review Data

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

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

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

	LotArea	YearBuilt	1stFlrSF	2ndFlrSF	FullBath	\	
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000		
mean	10516.828082	1971.267808	1162.626712	346.992466	1.565068		
std	9981.264932	30.202904	386.587738	436.528436	0.558916		
min	1300.000000	1872.000000	334.000000	0.000000	0.000000		
25%	7553.500000	1954.000000	882.000000	0.000000	1.000000		
50%	9478.500000	1973.000000	1087.000000	0.000000	2.000000		
75%	11601.500000	2000.000000	1391.250000	728.000000	2.000000		
max	215245.000000	2010.000000	4692.000000	2065.000000	3.000000		
	BedroomAbvGr	TotRmsAbvGrd					
count	1460.000000	1460.000000					
mean	2.866438	6.517808					
std	0.815778	1.625393					
min	0.000000	2.000000					
25%	2.000000	5.000000					
50%	3.000000	6.000000					
75%	3.000000	7.000000					
max	8.000000	14.000000					
	LotArea	YearBuilt	1stFlrSF	2ndFlrSF	FullBath	BedroomAbvGr	\
0	8450	2003	856	854	2	3	
1	9600	1976	1262	0	2	3	
2	11250	2001	920	866	2	3	
3	9550	1915	961	756	1	3	
4	14260	2000	1145	1053	2	4	
	TotRmsAbvGrd						
0	8						
1	6						
2	6						
3	7						
4	9						

Step 3: Specify and Fit Model:

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

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

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

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

Step 4: Make Predictions: (Tahmin Yapma)

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

Notlar:

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

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

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

Şimdi ise test datamızla modelimize tahmin üretirelim.

```
prediction = model.predict(X_test)  
print(prediction)  
  
[ 55870.35248488 124217.47107547  53061.56678938 114854.85209045  
 55870.35248488 115791.11398896  63360.44767289  92384.56652643  
 63360.44767289 102683.44740994]
```



Model Validation(Model geçerliliği):

Bir model oluşturduğunuz. Ama ne kadar iyi?

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

Model kalitesini ölçmek, modellerinizi tekrar tekrar geliştirmenin anahtarıdır.

What is Model Validation: (Model Validation Nedir)

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

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

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

Önce model kalitesini anlaşılabılır 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 metrik olarak özetlemeliyiz.

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

Bu metriği son kelimeyle başlayarak parçalayalım, hata.

Her ev için tahmin hatası:

```
error=actual-predicted
```

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

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

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

Bir modelimiz olduğunda, ortalama mutlak hatayı nasıl hesaplarız:

```
from sklearn.metrics import mean_absolute_error

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

```
434.71594577146544
```

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

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

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

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

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

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

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

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

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

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

Coding It

Scikit-learn Kütüphanesi, verileri iki parçaya bölmek için **train_test_split** işlevine sahiptir.

Bu verilerin bir kısmını modele uyacak şekilde eğitim verileri olarak kullanacağız ve diğer verileri **mean_absolute_error** hesaplamak için doğrulama verileri olarak kullanacağız.

Here is the code:

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

246802.63033873343

]:

Wow!

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

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

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

Exercises: Model Validation

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

Önceki alıştırmانın kaldığı kodlama ortamınızı ayarlamak için aşağıdaki hücreyi çalıştırın.

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

Step 1: Split Your Data (Verilerinizi Bölün)

Verilerinizi bölmek için train_test_split işlevini kullanın.

Random_state = 1 argümanını verin, böylece kontrol fonksiyonları kodunuza doğrularken ne bekleyeceğini bilir.

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

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

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

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

Correct

Step 2: Specify and Fit the Model

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

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

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

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

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

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

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

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

Doğrulama verilerinden tahminlerinizi ve gerçek değerlerinizi inceleyin.

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

```
29652.931506849316
   LotArea YearBuilt 1stFlrSF 2ndFlrSF FullBath BedroomAbvGr \
258    12435    2001     963     829       2        3
267     8400    1939    1052     720       2        4
288    9819    1967     900       0       1        3
649    1936    1978     630       0       1        1
1233   12160    1959    1188       0       1        3

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

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

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

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

Step 4: Calculate the Mean Absolute Error in Validation Data

+ Code

+ Markdown

```
[1]:  
from sklearn.metrics import mean_absolute_error  
val_mae = mean_absolute_error(val_y, val_predictions)  
  
# uncomment following line to see the validation_mae  
print(val_mae)  
  
# Check your answer  
step_4.check()
```

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

Underfitting and Overfitting

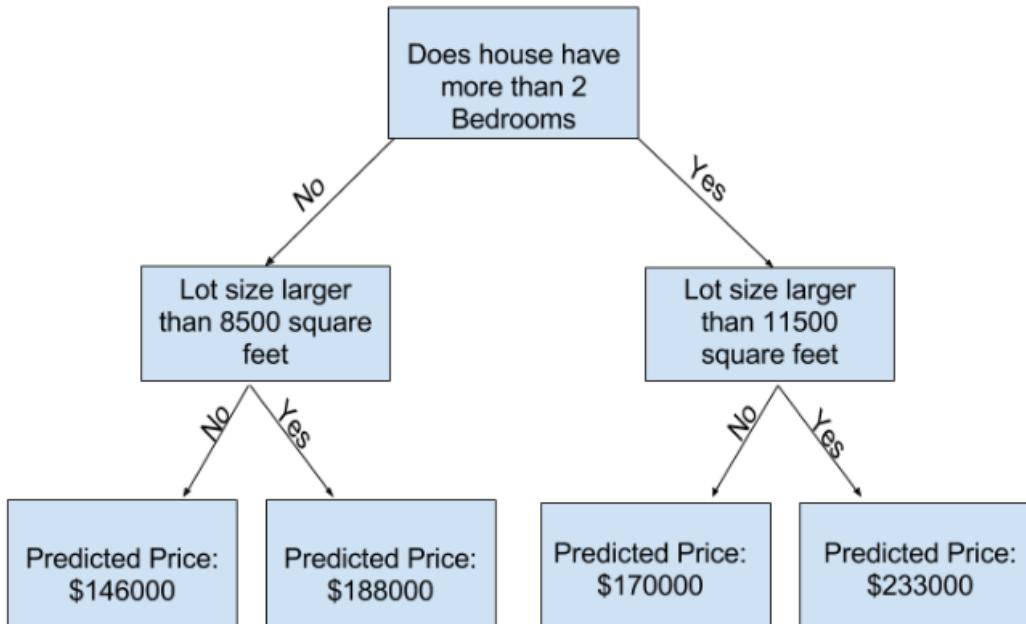
Bu adımın sonunda, uygun olmayan ve fazla uygunluk kavramlarını anlayacak ve modellerinizi daha doğru hale getirmek için bu fikirleri uygulayabileceksiniz.

Experimenting With Different Models

Artık model doğruluğunu ölçmenin güvenilir bir yoluna sahip olduğunuzda göre, alternatif modelleri deneyebilir ve hangisinin en iyi tahminleri verdienen görebilirsiniz. Peki modeller için hangi alternatifleriniz var?

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

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



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

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

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

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

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

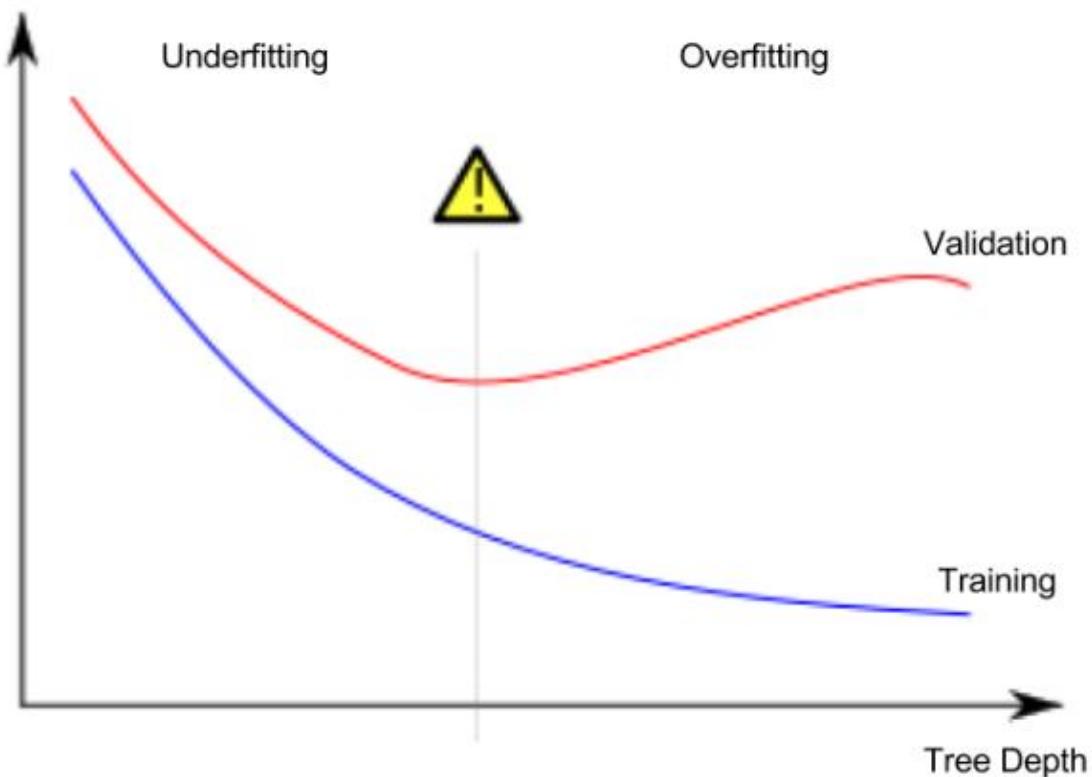
Aşırı derecede, eğer bir ağaç evleri sadece 2 veya 4'e bölerse, her grup hala çok çeşitli evlere sahiptir.

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

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

Görsel olarak, (kırmızı) doğrulama eğrisinin düşük noktasını istiyoruz

Mean Average Error



Examples:

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

Ancak **max_leaf_nodes** argümanı overfitting vs underfitting kontrol etmek için çok mantıklı bir yol sağlar.

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

Max_leaf_nodes için farklı değerlerden Mae puanlarını karşılaştırmaya yardımcı olmak için bir yardımcı program işlevi kullanabiliriz:

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

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

Veriler, daha önce gördüğünüz (ve daha önce yazdığınız) kodu kullanarak train_X, val_X, train_y ve val_y içine yüklenir.

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

mae'yi farklı max_leaf_nodes değerleriyle karşılaştırın

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

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

Listelenen seçeneklerden 500, en uygun yaprak sayısıdır.

Sonuç:

Modeller şunlardan herhangi birine sahip olabilir.

- Overfitting: gelecekte tekrarlamayacak sahte pattern(desen)leri yakalamak , daha az doğru tahminlere yol açmak veya
- Underfittin: 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 denememize ve en iyisini elde etmemizi sağlar.

Overfitting:

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

Underfitting:

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

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

Exercise: Underfitting and Overfitting

Tekrarlamak:

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

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

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'
home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

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

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
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

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

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

Step 1: Compare Different Tree Sizes

Bir dizi olası değerden max_leaf_nodes için aşağıdaki değerleri 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.

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

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

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

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

En iyi ağaç boyutunu 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 gerekmekz.

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

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

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

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

Random Forests:

Giriş:

Decision Tree sizi zor bir kararla baş başa bırakır. Çok sayıda yapraklı derin bir ağaç, her tahmin, yaprağındaki sadece birkaç evden gelen tarihsel verilerden geldiğinden fazla olacaktır. Ancak, az yapraklı sıg 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 ağaçının tahminlerini ortalayarak bir tahmin yapar.

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

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

Example

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

- train_X
- val_X
- train_y
- val_y

```
In [1]:  
import pandas as pd  
  
# Load data  
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'  
melbourne_data = pd.read_csv(melbourne_file_path)  
# Filter rows with missing values  
melbourne_data = melbourne_data.dropna(axis=0)  
# Choose target and features  
y = melbourne_data.Price  
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',  
                      'YearBuilt', 'Lattitude', 'Longtitude']  
X = melbourne_data[melbourne_features]  
  
from sklearn.model_selection import train_test_split  
  
# split data into training and validation data, for both features and target  
# The split is based on a random number generator. Supplying a numeric value to  
# the random_state argument guarantees we get the same split every time we  
# run this script.  
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)
```

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

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

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

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

```
202888.18157951365
```

Sonuç:

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

Single decision tree'nin maksimum derinliğini değiştirdiğimiz gibi Random Forest'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: 29,652
Validation MAE for best value of max_leaf_nodes: 27,283
Setup complete

Exercises

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

Step 1: Use a Random Forest

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

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

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

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

Validation MAE for Random Forest Model: 21857.15912981083

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

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 10/10 overfitted on the training data and it has bad performance on the test data. Which hyper-parameter could help us to get rid of this problem?

Note: You can use `sklearn.tree.DecisionTreeClassifier` documentation.<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier>*

criterion

max_depth ✓

random_state

splitter

AÇIKLAMA:

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

[https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier *](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)

➤ **max_depth-(integer or none)- Default=None**

Bu, ağaçlarınızı ne kadar derin yapmak istediğiniz sefer. Max_depth'inizi ayarlamamızı önerilir, çünkü overfitting baş etmek için önerilir.

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

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

```
In [2]: import pandas as pd  
  
data = pd.read_csv("/home/fatih/Desktop/melb_data.csv")  
  
data.describe()
```

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.000000	8205.000000	1
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	90.678964	0.965921	0.691712	0.962634	3990.669241	541.014538	37.273762
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000000	1940.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.000000	1999.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000

- I, II
- II, III
- II, III, IV
- I, II, III



AÇIKLAMA

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

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

IV. IV. veri kümesinde 9 odalı bir ev yoktur

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

10/10

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

AÇIKLAMA

Overfitting ve underfitting için hangisi yanlıştır?

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

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

10/10

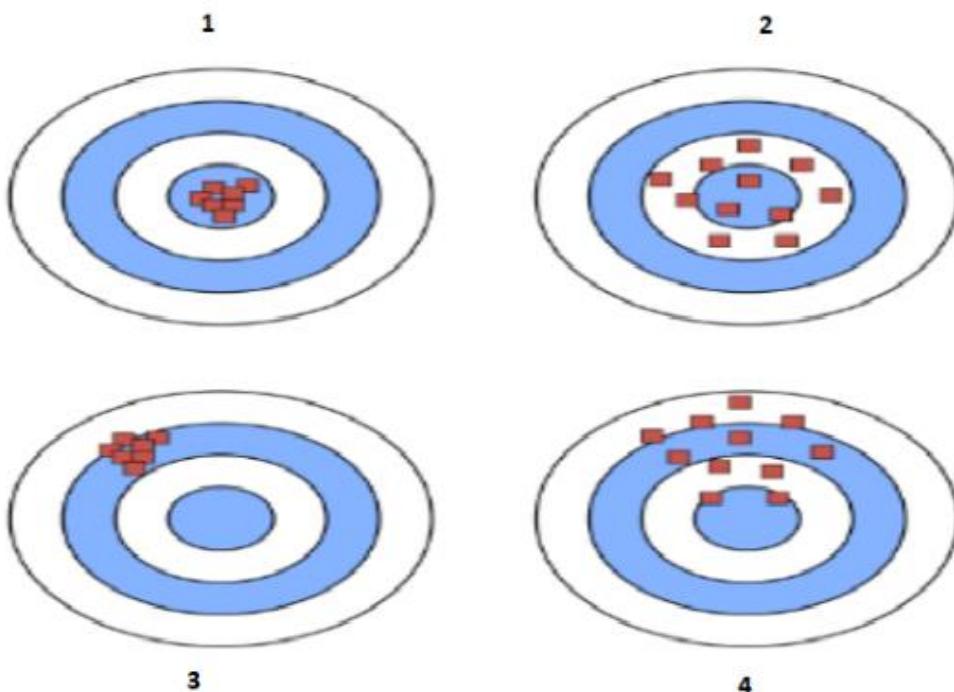
- DataFrame.head(x) shows x samples in the DataFrame from the beginning.
- DataFrame.describe() shows summary of the data.
- model.predict() determines how accurate the model's predictions are. ✓
- DataFrame.dropna(axis=0) drops missing values.

AÇIKLAMA

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

- DataFrame.head (x), DataFrame x örneklerini baştan gösterir.
- DataFrame.describe () verilerin özetini gösterir.
- model.predict (), modelin tahminlerinin ne kadar doğru olduğunu belirler.
- DataFrame.dropna (axis=0) eksik değerleri düşürür.
- **predict**: Regresyon, sınıflandırma, kümeleme gibi yöntemler kullanarak yapacağınız çalışmalarda tahmin edilen etiket bilgisini **predict** fonksiyonuyla elde edebilirsiniz.

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



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

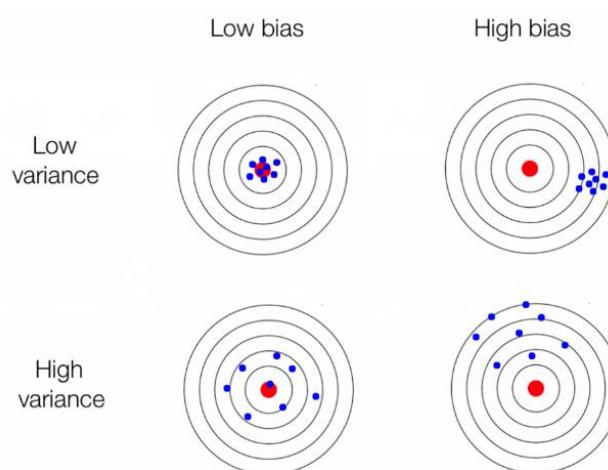
- 1:High Bias- Low Variance 2: High Bias-High Variance 3:Low Bias-Low Variance
4:Low Bias-High Variance
- 1:High Bias- High Variance 2:High Bias-Low Variance 3:Low Bias-High Variance
4:Low Bias-Low Variance

AÇIKLAMA

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

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

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



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

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

I - It is an algorithm that aims to increase the classification value by producing multiple decision trees.

II - It was created by combining Bagging and Random Subspace methods.

III - While creating the tree, it is made performance evaluation with 2/3 of the data set.

I, III

II, III

I, II



I, II, III

AÇIKLAMA

Random forests algoritmasına göre, aşağıdaki ifadelerden hangisi doğrudur?

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

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

10/10

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

- They generate different random number so the train_X differs from each other.
- They generate different same number and the train_X is equal to each other.
- They generate different random number so the train_X is equal to each other.
- They generate different random number ,but the train_X is equal to each other. ✓

AÇIKLAMA

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

- Farklı rasgele sayı üretirler, böylece train_X birbirinden farklıdır.
- Farklı aynı sayı üretirler ve train_X birbirine eşittir.
- Farklı rasgele sayı üretirler, böylece train_X birbirine eşittir.
- Farklı rasgele sayı üretirler, ancak train_X birbirine eşittir.

shuffle : bool, default = Doğru

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

random_state : int veya RandomState örneği, default = Yok

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

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

Misal:

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

`shuffle` True olarak ayarladığınızda `random_state`, rastgele sayı üretici için tohum olarak kullanılır. Sonuç olarak, veri kümeniz rastgele olarak tren ve test kümelerine ayrıılır.

Random_state = 42 ile örnek:

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

Random_state = 44 ile örnek:

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

✓ Q8- Trees have their length and we call that the depth of the tree. 10/10
RandomForestRegressor, in scikit-learn library, has a maximum leaf (`max_depth`) parameter which is `None` as default which means nodes are expanded until all leaves are pure. What can be said if we change the number of maximum leaf nodes of a random forest? *

- Length of a tree does not affect any of the results.
- Model may overfit for large depth values. ✓
- The longer tree is the better tree.
- Short trees more precise than long trees.

AÇIKLAMA

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

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

✓ Q9- Let assume, we have a data set called home_data with 3 features 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()

AÇIKLAMA

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

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

```
from sklearn.metrics import mean_absolute_error

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

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

AÇIKLAMA

Aşağıdaki kod parçalarının amacı nedir?

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

Intermediate Machine Learning (Orta Düzey Makine Öğrenimi)

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

Introduction (Giriş)

Bu mikro kurs için neye ihtiyacınız olduğunu inceleyin.

Kaggle Learn'in Intermediate Machine Learning mikro kursuna hoş geldiniz!

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

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

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

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

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

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

The screenshot shows the 'InClass Prediction Competition' page on Kaggle. The title is 'Housing Prices Competition for Kaggle Learn Users'. Below it, a sub-header says 'Apply what you learned in the Machine Learning course on Kaggle Learn alongside others in the course.' There are tabs for Overview, Data, Kernels, Discussion, Leaderboard, Rules, Team, My Submissions, and Submit Predictions. The Overview tab is selected. On the left, there's a sidebar with links for Description, Evaluation, Frequently Asked Questions, Tutorials, and a blue '+ Add Page' button. The main content area has a section titled 'Start here if...' with a brief introduction about the competition's purpose. It also includes a 'Competition Description' section with text about the dataset and its characteristics.

Prerequisites (Önkoşullar)

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

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

Your Turn(Sıra Sende)

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

Exercise: Introduction

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

Setup

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

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

Setup Complete

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



Y_train ve y_valid'deki tahmin hedefleriyle(prediction targets) birlikte x_train ve X_valid'deki training (eğitim) ve validation(doğrulama) features yüklemek için değişiklik yapmadan bir sonraki kod hücresinin çalıştırın.

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

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

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

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

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

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

```
[ ]:
```

Step 1: birkaç modeli değerlendirin

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

Kod hüresini değişiklik yapmadan çalıştırın.

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

Buradaki parametrelere göz atalım;

➤ **n_estimators-(integer)- Default=10**

Tahminleri için, bir Rasgele Orman içinde inşa etmek istediğiniz ağaç sayısını. Sayı ne kadar yüksekse, kodunuzun çalışması için daha uzun süreoeğini bilmek önemlidir. Bilgisayarınızın işlem hızıyla ilgili önceki bilgilere dayanarak, bu hızla orantılı bir **n_estimator** oluşturmak gereklidir.

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

➤ **min_samples_split-(integer, float)-Default=2**

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

➤ **max_depth-(integer or none)- Default=None**

Bu, ağaçlarınızı ne kadar derin yapmak istediğinizizi seçer. `Max_depth'inizi ayarlamınızı önerilir`, çünkü overfitting baş etmek için önerilir.

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

Beşten en iyi modeli seçmek için, aşağıda bir `score_model ()` islevi tanımlarız.

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

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

Kod hücresini değişiklik yapmadan çalıştırın.

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

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

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

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

Step 2: Generate test predictions (Test tahminleri oluştur)

Harika. Doğru bir modeli neyin yaptığına nasıl değerlendireceğinizi biliyorsunuz.

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

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

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

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

Missing Values (Eksik Değerler)

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

Introduction (Giriş)

Verilerin eksik değerlerle sonuçlanması birçok yolu vardır.

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

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

Yani aşağıdaki stratejilerden birini seçmeniz gereklidir.

Three Approaches

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

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

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

Bath
1.0
1.0
2.0
2.0

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

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

Bu yaklaşım sütunu tamamen drop edecektir.

- 2) A Better Option: Imputation (Daha iyi bir seçenek)

İmputation, eksik değerleri bazı sayılarla doldurur.

Örneğin, her sütun boyunca ortalama değeri doldurabiliriz.

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
2.0	2.0

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

3) An Extension To Imputation

Imputation standart yaklaşımdır ve genellikle iyi çalışır. Bununla birlikte, imputed edilen değerler sistematik olarak gerçek değerlerinin (veri kümesinde toplanmamış) üstünde veya altında olabilir. Veya eksik değerlere sahip satırlar başka bir şekilde benzersiz olabilir. Bu durumda, modeliniz başlangıçta hangi değerlerin eksik olduğunu göz önünde bulundurarak daha iyi tahminlerde bulunur.

Bed	Bath	Bed	Bath	Bed_was_missing
1.0	1.0	1.0	1.0	FALSE
2.0	1.0	2.0	1.0	FALSE
3.0	2.0	3.0	2.0	FALSE
NaN	2.0	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.

Example(Örnek)

Örnekte, [Melbourne Housing dataset](#) ile çalışacağız.

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

Veri yüklemeye adımına odaklanmayacağız. Bunun yerine, `x_train`, `X_valid`, `y_train` ve `y_valid`'de training ve validation verilerine zaten sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.

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

Eksik değerlerle başa çıkmak için farklı yaklaşımları karşılaştırmak için bir işlev `score_dataset()` tanımlıyoruz. Bu işlev, bir random forest modelinden ortalama mutlak hata (MAE) bildirir.

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

Score from Approach 1 (Drop Columns with Missing Values)

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

```

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

```

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

183550.22137772635

Score from Approach 2 (Imputation)

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

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

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

```

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

```

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

Eksik değerleri doldurmak için aşağıdaki satırı kullanabiliriz:

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

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

MAE from Approach 2 (Imputation):

178166.46269899711

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

Score from Approach 3 (An Extension to Imputation)

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

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

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

MAE from Approach 3 (An Extension to Imputation):

178927.503183954

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

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

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

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

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

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

```
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    4387  
dtype: int64
```

Conclusion

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

Exercises

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

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

```
import pandas as pd  
from sklearn.model_selection import train_test_split  
  
# Verileri oku  
X_full = pd.read_csv('../input/train.csv', index_col='Id')  
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')  
  
# Eksik hedefi olan satırları kaldırın, hedefi belirleyicilerden ayıri  
X_full.dropna(axis=0, subset=['SalePrice'], inplace=True)  
y = X_full.SalePrice  
X_full.drop(['SalePrice'], axis=1, inplace=True)  
  
# İşleri basit tutmak için sadece sayısal tahmincileri kullanacağız  
X = X_full.select_dtypes(exclude=['object'])  
X_test = X_test_full.select_dtypes(exclude=['object'])  
  
# Break off validation set from training data  
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,  
                                                    random_state=0)
```

Notlar:

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

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

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

```
▶ X_train.head()  
  
Out[2]:  
MSSubClass LotFrontage LotArea OverallQual OverallCond YearBuilt YearRemodAdd MasVnrArea BsmtFinSF1 BsmtFinSF2 ... GarageArea Wc  
Id  
619 20 90.0 11694 9 5 2007 2007 452.0 48 0 ... 774  
871 20 60.0 6600 5 5 1962 1962 0.0 0 0 ... 308  
93 30 80.0 13360 5 7 1921 2006 0.0 713 0 ... 432  
818 20 NaN 13265 8 5 2002 2002 148.0 1218 0 ... 857  
303 20 118.0 13704 7 5 2001 2002 150.0 0 0 ... 843  
5 rows x 36 columns
```

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

Step 1: Preliminary investigation (Ön Soruşturma)

Aşağıdaki kod hücresini herhangi bir değişiklik yapmadan çalıştırın.

```
▶ # Shape of training data (num_rows, num_columns)  
print(X_train.shape)  
  
# Eğitim verilerinin her sütununda eksik değerlerin sayısı  
missing_val_count_by_column = (X_train.isnull().sum())  
print(missing_val_count_by_column[missing_val_count_by_column > 0])  
  
+ Örnek + Mark et  
  
(1168, 36)  
LotFrontage 212  
MasVnrArea 6  
GarageYrBlt 58  
dtype: int64
```

Part A

```
▶ # Aşağıdaki satırı doldurun: eğitim verilerinde kaç satır var?  
num_rows = 1168  
  
# Aşağıdaki satırı doldurun: eğitim verisinde kaç sütun var  
# eksik değerler var mı ?  
num_cols_with_missing = 3  
  
# Aşağıdaki satırı doldurun: kaç eksik giriş bulunur  
# tüm eğitim verileri?  
tot_missing = 276  
  
# Check your answers  
step_1.a.check()
```

Part B

Yukarıdaki cevaplarınızı göz önünde bulundurarak, eksik değerlerle başa çıkmadan 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.

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

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

+ Code

+ Markdown



Step 2: Drop columns with missing values

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

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

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

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



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

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

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



Step 3: Imputation

Part A

Her sütundaki eksik değerleri, ortalama değerler ile doldurmak için kod parçasını yazın. Önceden işlenmiş DataFrames değerini imputed_X_train ve imputed_X_valid olarak ayarlayın. Sütun adlarının X_train ve X_valid ile aynı olduğundan emin olun.

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

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

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

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



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

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

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

Part B

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

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

Çözüm: Veri kümesinde çok az eksik değer olduğu düşünüldüğünde, imputasyonun sütunları tamamen düşürmekten daha iyi performans göstermesini bekleriz. Ancak bu durumda, sütunları düşürmenin biraz daha iyi performans gösterdiğini görüyoruz! Bu muhtemelen kısmen veri kümesindeki gürültüye atfedilebilirken, başka bir potansiyel açıklama, imputasyon yönteminin bu veri kümesine mükemmel bir uyumunun olmadığıdır. Yani, ortalama değer ile doldurmak yerine, her eksik değeri 0 değerine ayarlamak, en sık karşılaşılan değeri doldurmak veya başka bir yöntem kullanmak daha mantıklıdır. Örneğin, garajın inşa edildiği yılı gösteren GarageYrBlt sütununu düşünün. Bazı durumlarda, eksik bir 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.

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

Correct

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

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

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

MAE (Your approach):
17837.82570776256

Part B

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

- önceden işlenmiş test veri çerçevesinin eksik değerleri yoktur ve
- final_X_test, x_test ile aynı sayıda satırı sahiptir.

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

step_4.b.check()
```

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

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

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 karşılaşıracağınız.

Üç Yaklaşım

1) Drop Categorical Variables

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

2) Label Encoding

Label Encoding her benzersiz değeri farklı bir tamsayıya atar.

Breakfast
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 varsayımda 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 çalışacağımız.

The diagram illustrates the process of one-hot encoding. On the left, there is a table with a single column labeled 'Color' containing five rows: 'Red', 'Red', 'Yellow', 'Green', and 'Yellow'. An arrow points from this table to a larger table on the right. The right table has three columns labeled 'Red', 'Yellow', and 'Green'. It contains five rows corresponding to the entries in the original table. The values are binary: 'Red' is represented by a 1 in the first column and 0s in the others; 'Yellow' is represented by 0s in the first two columns and a 1 in the third; 'Green' is represented by 0s in the first two columns and a 1 in the third; and 'Yellow' again is represented by 0s in the first two columns and a 1 in the third.

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 koyma; orijinal değer "Sarı" ise, "Sarı" sütununa 1 koyma 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 alırsa 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 yüklemeye adımına odaklanmayacağız. Bunun yerine, zaten X_train, X_valid, y_train ve y_valid'de eğitim ve doğrulama verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.

```

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

```

[7]: X_train.head()

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

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

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

```

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

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

```

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)

```

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

```

**MAE from Approach 2 (Label Encoding):
165936.40548390493**

Yukarıdaki kod hücresında, 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

Ignore= yoksaymak anlamında kullanılır.

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

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

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

```
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 yaklaşımı gelince, geri dönen MAE puanları çok yakın olduğundan, birinin diğerine karşı anlamlı bir faydası görünmemektedir.

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

Sonuç

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

Exercises

Kategorik değişkenleri encode ederek 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.

```
[1]:  
import pandas as pd  
from sklearn.model_selection import train_test_split  
  
# Verileri oku  
X = pd.read_csv('../input/train.csv', index_col='Id')  
X_test = pd.read_csv('../input/test.csv', index_col='Id')  
  
# Eksik hedefi olan satırları kaldırın, hedefi belirleyicilerden ayıran  
X.dropna(axis=0, subset=['SalePrice'], inplace=True)  
y = X.SalePrice  
X.drop(['SalePrice'], axis=1, inplace=True)  
  
# İşleri basit tutmak için, eksik değerlere sahip sütunları drop edeceğiz.  
cols_with_missing = [col for col in X.columns if X[col].isnull().any()]  
X.drop(cols_with_missing, axis=1, inplace=True)  
X_test.drop(cols_with_missing, axis=1, inplace=True)  
  
# Break off validation set from training data  
X_train, X_valid, y_train, y_valid = train_test_split(X, y,  
                                                    train_size=0.8, test_size=0.2,  
                                                    random_state=0)
```



X_train.head()

Out[4]:

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

5 rows × 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

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

Step 1: Drop columns with categorical data

En basit 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.

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

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

Bu yaklaşım için MAE hesaplayalım.

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

Step 2: Label Encoding

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

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

Unique values in 'Condition2' column in training data: ['Norm' 'PosA' 'Feedr' 'PosN' 'Artery' 'RR Ae']
Unique values in 'Condition2' column in validation data: ['Norm' 'RR An' 'RR Nn' '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 'RRAn' 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.

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

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

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

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

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

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

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.

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

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

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

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

Correct

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

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

Step 3: Investigating Cardinality (Kardinalite Araştırması)

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

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

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

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

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

Zip():

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

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

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

Aşağıdaki soruları cevaplamak için yukarıdaki çıktıyı kullanın.

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

# Aşağıdaki satırı doldurun: Tek bir etkin kodlama için kaç sütun gereklidir?
# Eğitim verilerinde 'Neighborhood' değişkeni?
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 uygulayacağız. 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.

```
3]: # Aşağıdaki satırı doldurun: Veri kümese kaç giriş eklendi:  
# replacing the column with a one-hot encoding?(# sütunu tek yeni kodlamayla değiştirmek mi?)  
OH_entries_added = 1e4*100-1e4  
  
# Aşağıdaki satırı doldurun: Veri kümese kaç giriş eklendi:  
# sütunu bir etiket kodlaması ile değiştirme?  
label_entries_added = 0  
  
# Check your answers  
step_3.b.check()
```

Correct

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

One Hot Encoding her unique kolon değeri için yeni kolon oluşturulması anlamına geliyordu. Buradaki 10e4 aslında 10000 anlamına gelir. (e=exponential yani 10 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.

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

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.

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

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

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

Categorical columns that will be one-hot encoded: ['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'Landslope', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'ExterQual', 'ExterCond', 'Foundation', 'Heating', 'HeatingQC', 'CentralAir', 'KitchenQual', 'Functional', 'PavedDrive', 'SaleType', 'SaleCondition']

Categorical columns that will be dropped from the dataset: ['Exterior2nd', 'Neighborhood', 'Exterior1st']
```

X_train ve X_valid içindeki verilere one-hot encoding yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla OH_X_train ve OH_X_valid olarak ayarlayın.

- Veri kümesindeki kategorik sütunların tam listesi Python listesi object_cols içinde bulunabilir.
- yalnızca Low_cardinality_cols içindeki kategorik sütunlara one-hot encoding uygulanmalıdır. 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
```

Not

Concat fonksiyonu iki seriyi birleştirmek için kullanılır.

Step 5: Generate test predictions and submit your results

4. Adım'ı tamamladıktan sonra, sonuçlarınızı skor tablosuna göndermek için öğrendiklerinizi kullanmak isterseniz, tahminler oluşturmadan önce test verilerini önceden işlemeniz 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 adımdaki 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 ölçüekte konuşlandırılabilir bir şeye geçirmek şaşırtıcı derecede zor olabilir. Burada birçok ilgili kaygıya girmeyeceğiz, ancak pipeline yardımcı olabilir.
- **Model Validation için Daha Fazla Seçenek:** Bir sonraki öğreticide cross validation'u kapsayan bir örnek göreceksiniz.

Example

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

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

```
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')

# 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 ['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()
```

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

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

Out[2]:

Out[2]:

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

Out[2]:

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: Önisleme Adımlarını Tanımlayın

Bir pipeline'nın ön işleme ve modelleme adımlarını nasıl bir araya getirdiğine benzer şekilde, farklı önisleme 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.

In [3]:

```
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)
    ])
```

Constant: "constant" , eksik değerleri fill_value ile değiştirin. Dizeler veya sayısal verilerle kullanılabilir.

- "Most_frequent" ise, her sütun boyunca en sık kullanılan değeri kullanarak eksik olanı değiştirin. Dizelerle veya sayısal verilerle kullanılabilir.

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 kullanız. 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 sofistik 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.

```

▶ import pandas as pd
from sklearn.model_selection import train_test_split

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

# Eksik hedefi olan satırları kaldırın, hedefi belirleyicilerden ayıran
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", bir sütundaki benzersiz değerlerin sayısı anlamına gelir
# Nispeten düşük cardinality ile kategorik sütunları seçin ( uygun ama keyfi )
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()

```

[4]: X_train.head()

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Condition1	Condition2	...	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea
Id																		
619	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Norm	Norm	...	774	0	108	0	0	260	0
871	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	PosN	Norm	...	308	0	0	0	0	0	0
93	RL	Pave	Grvl	IR1	HLS	AllPub	Inside	Gtl	Norm	Norm	...	432	0	0	44	0	0	0
818	RL	Pave	NaN	IR1	Lvl	AllPub	CulDeSac	Gtl	Norm	Norm	...	857	150	59	0	0	0	0
303	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Norm	Norm	...	843	468	81	0	0	0	0

5 rows × 18 columns

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

```

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

# Sayısal veriler için ön işleme
numerical_transformer = SimpleImputer(strategy='constant')

# Kategorik veriler için ön işleme
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Sayısal ve kategorik veriler için paket ön işleme
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Modeli tanımla
model = RandomForestRegressor(n_estimators=100, random_state=0)

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

# Eğitim verilerini ön işleme, FIT modeli
clf.fit(X_train, y_train)

# Doğrulama verilerinin ön işlenmesi, tahminler alın
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 önisleme adımlarınızı ve Random Forest modelinizi tanımlayın. Aşağıdaki değişkenler için değerleri girin:

- numerical_transformer
- categorical_transformer
- model

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

```

1]: # Sayısal veriler için ön işleme
numerical_transformer = SimpleImputer(strategy="median")# Your code here

# Kategorik veriler için ön işleme
categorical_transformer=Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))]) # Your code here

# Sayısal ve kategorik veriler için paket ön işleme
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.)

```
[12]: # Bundle preprocessing and modeling code in a pipeline(# Bir pipeline'da ön işleme ve modelleme kodunu paketleyin)
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                             ('model', model)
                            ])

# Eğitim verilerinin ön işleme, FIT modeli
my_pipeline.fit(X_train, y_train)

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

# Modeli değerlendirin
score = mean_absolute_error(y_valid, preds)
print('MAE:', score)

# Check your answer
step_1.b.check()

MAE: 17487.872363813696
```

İpucu: Daha iyi performans elde etmek için ön işleme 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.

```
[13]: # Test verilerinin ön işleme, uygun 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)
```

Cross-Validation

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

Introduction

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

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

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

Ancak bu yaklaşımın bazı dezavantajları vardır.

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

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

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

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

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

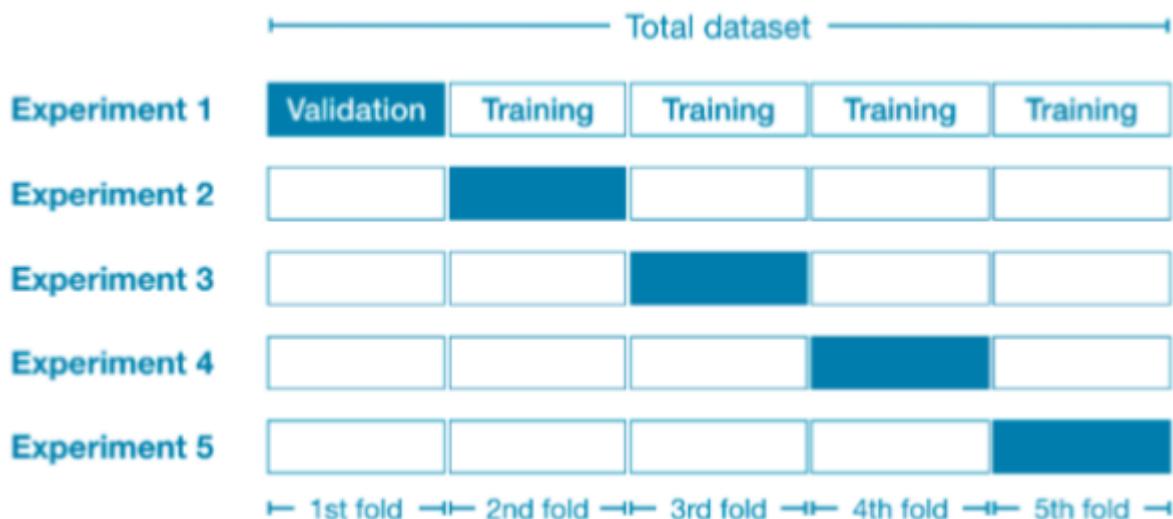
Genel olarak, validation seti ne kadar büyük olursa, model kalitesi ölçümüzde o kadar az rastgelelik ("gürültü") olur ve o kadar güvenilir olur.

Ne yazık ki, yalnızca training verilerimizdeki satırları kaldırarak büyük bir validation kümeli alabiliyoruz 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 kümelerinin % 20'si olan 5 parçaya bölgerek başlayabiliyoruz. 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 ödünləşmələr 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 küməsini oluşturan şey için basit bir eşik yoktur. Ancak modelinizin çalışması birkaç dakika veya daha az sürüyorsa, muhtemelen cross-validation'a geçmeye değer.

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

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

Example

Önceki derslerdeki verilerle çalışacağız. Input verilerini X'e, Output verilerini y'ye 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) seçtik. (https://scikit-learn.org/stable/modules/model_evaluation.html)

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

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

```
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.

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

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

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

# Sadece sayısal sütunları seçin
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()

Out[1]:
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	Sr
1	60	65.0	8450	7	5	2003	2003	196.0	706	0 ...	548	0	61	0	0	
2	20	80.0	9600	6	8	1978	1978	0.0	978	0 ...	480	298	0	0	0	
3	60	68.0	11250	7	5	2001	2002	162.0	486	0 ...	608	0	42	0	0	
4	70	60.0	9550	7	5	1915	1970	0.0	216	0 ...	642	0	35	272	0	
5	60	84.0	14200	8	5	2000	2000	350.0	655	0 ...	838	192	84	0	0	

5 rows × 36 columns

```
+ Code + Markdown
```

```
[3]: X.head(4)
```

	AllQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold
7	5	2003	2003	196.0	706	0 ...	548	0	61	0	0	0	0	0	0	2	2008
6	8	1978	1978	0.0	978	0 ...	480	298	0	0	0	0	0	0	0	5	2007
7	5	2001	2002	162.0	486	0 ...	608	0	42	0	0	0	0	0	0	9	2008
7	5	1915	1970	0.0	216	0 ...	642	0	35	272	0	0	0	0	0	2	2008
8	5	2000	2000	350.0	655	0 ...	838	192	84	0	0	0	0	0	0	12	2008

+ Code + Markdown

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.

```
[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))])
1)
```

+ Code + Markdown

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.



```
▶ from sklearn.model_selection import cross_val_score
# Sklearn hesapladığı için -1 ile çarpin * negatif * 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
```

+ Code + Markdown

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()` ögesine sağlanan `n_estimators` parametresi, Random Forest modelindeki ağaç sayısı ayarlanırken kullanılır.



```
[31]: 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()
```

Correct

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()
```

▶ | results

```
Out[11]: {50: 18353.8393511688,
100: 18395.2151688032,
150: 18288.730828956387,
200: 18248.345889881505,
250: 18255.2692247291,
300: 18275.241922621914,
350: 18270.2918388843,
400: 18270.197974402367}
```

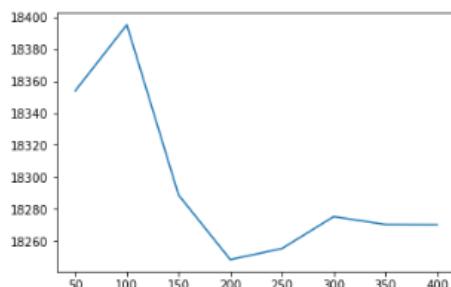
+ Code + Markdown

Step 3: Find the Best Parameter Value

▶ |

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(list(results.keys()), list(results.values()))
plt.show()
```



+ Code + Markdown

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.

[13]:

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

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

Correct

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.

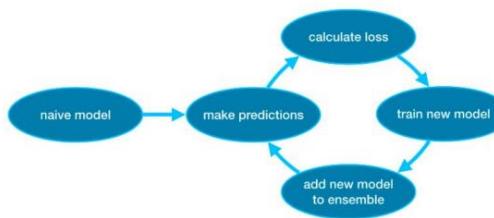
Gradient Boosting

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 çılginca 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, tıpkı 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: Series.base is de  
precated 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 is now deprec  
ated in favor of reg:squarederror.  
  
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 ettiğimiz 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 ayarlamak ve ardından yinelemeyi durdurmak için en uygun zamanı bulmak için early_stopping_rounds kullanmak akıllıcadır.

İşi şansa bırakmanın bazen validation puanlarının iyileşmediği tek bir round'a denk geldiğinde döngüyü durdurmaması 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 çalıştırığınızda 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) çarpabiliyoruz.

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 emin olmak isterseniz, 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 çalışacağımız.



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.

```

import pandas as pd
from sklearn.model_selection import train_test_split

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

# Hedefi eksik olan satırları kaldırın, hedefi öngörülerden ayırin
X.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X.SalePrice
X.drop(['SalePrice'], axis=1, inplace=True)

# Eğitim verilerinden doğrulama setini kırın
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" bir sütundaki benzersiz değerlerin sayısı anlamına gelir
# Nispeten düşük Cardinality ile kategorik sütunları seçin ( uygun ama keyfi)
low_cardinality_cols = [cname for cname in X_train_full.columns if X_train_full[cname].nunique() < 10 and
                           X_train_full[cname].dtype == "object"]

# Numeric sütunları seç
numeric_cols = [cname for cname in X_train_full.columns if X_train_full[cname].dtype in ['int64', 'float64']]

# Sadece seçilen sütunları tut
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)(# One hot verileri kodlamak (kodu kısaltmak için pandas kullanıyoruz))
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)

```

Notlar:

Get_dummies: Sütunlardan birinin kategorik bir değişken içerdiği bir dizi veri çerçevesi var. Bunu birkaç kukla değişkene dönüştürmek istiyorum, bu durumda normalde `get_dummies` kullanırım.

Olan şudur ki, `get_dummies`, kaç tane kategori olduğunu bulmak için her veri çerçevesindeki verilere bakar ve böylece uygun sayıda yapay değişkenler yaratır.

One Hot Encoding yöntemini **Sci-kit** yerine pandasın `get_dummies` fonksiyonu ile çok daha hızlı ve rahat bir şekilde kullanabilirsiniz.

Align: Test verilerinin, `align` komutuyla egzersiz verileriyle aynı şekilde kodlandığından emin olun. `Align` komutu, sütunların her iki veri kümesinde de aynı sırada görünmesini sağlar (her veri kümesinde hangi sütunların sıraya girdiğini belirlemek için sütun adlarını kullanır.) `Join='left'` argümanı, SQL'İN left join eşdeğerini yapacağımızı belirtir. Bu, diğerinde değil, bir veri kümesinde görünen sütunlar varsa, sütunları tam olarak eğitim verilerimizden saklayacağımızı anlamına gelir. `Join='inner'` argümanı, SQL veritabanlarının iç birleştirme olarak adlandırdığı şeyi yapar ve yalnızca her iki veri kümesinde de görünen sütunları tutar. Bu da mantıklı bir seçim.

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.

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

[7]:

```
# Hesapla 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()
```

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.

Step 2: Modelinizi İyileştirin

Artık varsayılan bir modeli temel olarak eğittiğinize göre, daha iyi performans elde edip edemeyeğinizi 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 öğrendiklerinizi 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.

```
[9]: # Modeli tanımlayın  
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  
  
# Tahmin alın  
predictions_2 = my_model_2.predict(X_valid) # Your code here  
  
# MAE'yi hesapla  
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()
```

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!

```
[10]: # 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()
```

Correct

Data Leakage (Veri Sızıntısı)

Bu bölümde, **Data Leakage(Veri Sızıntısı)**'nın ne olduğunu ve nasıl önleneceğini öğreneceksiniz.

Bunu nasıl önleyeceğinizi bilmiyorsanız, sizıntı sık sık ortaya çıkacak ve modellerinizi ince ve tehlikeli yollarla mahvedecekтир.

Bu, veri bilimcilerin uygulamaları için en önemli kavramlardan biridir.

Introduction

Data Leakage (verisızıntısı), 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.

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

İki ana sizıntı 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:

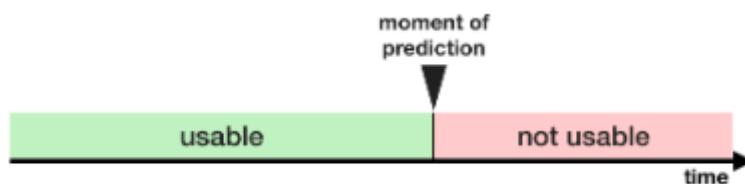
got_pneumonia	age	weight	male	took_antibiotic_medicine	...
False	65	100	False	False	...
False	72	130	True	False	...
True	58	100	False	True	...

İnsanlar pnömoni olduktan sonra iyileşmek için antibiyotik ilaçlar alırlar. Ham veriler, bu sütunlar arasında güclü 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 kullanıma geçtiğinde çok büyük yanlışlar yapacaktır. Çünkü pnömoni olan hastalar tedaviye başlamadan önce antibiyotik almamış olacaktır.

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



Train-Test Contamination

Training verilerini, validation verilerinden ayırmaya dikkat etmediğinizde farklı bir sizıntı 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 önişleme 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, preprocesing'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 uygulaması hakkındaki 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	majorcards	active
0	0	37.66667	4.5200	0.033270	124.983300	True	False	3	54	1	12
1	0	33.25000	2.4200	0.005217	9.854167	False	False	3	34	1	13
2	0	33.66667	4.5000	0.004156	15.000000	True	False	4	58	1	5
3	0	30.50000	2.5400	0.065214	137.869200	False	False	0	25	1	7
4	0	32.16667	9.7867	0.067051	546.503300	True	False	2	64	1	5

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 + bakiye 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:

```
[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 açıklır, ancak açıklamaya ilgili görünüyorum.

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

Target Leakage olmayan bir modeli şu şekilde çalıştırırız:

```
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 (crossvalidation'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

Çoğu insan uzun süre düşünene kadar hedef sizıntıyı çok zor buluyor.

Dolayısıyla, konut fiyatıörneğindeki sizıntıyı düşünmeye çalışmadan önce, diğer uygulamalarda birkaç örnek vereceğiz. Ev fiyatları ile ilgili bir soruya geri döndüğünüzde işler daha tanındık gelecektir.

Setup

Aşağıdaki sorular cevaplarınız hakkında size geri bildirim verecektir. Geri bildirim sistemini kurmak için aşağıdaki hücreyi çalıştırın.

```
[1]: # Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.ml_intermediate.ex7 import *
print("Setup Complete")
```

Setup Complete

1. *The Data Science of Shoelaces(Ayakkabı Bağcıklarının veri Bilimi)*

Nike sizi ayakkabı malzemelerinden tasarruf etmelerine yardımcı olacak bir veri bilimi danışmanı olarak işe aldı. İlk göreviniz, çalışanlarının her ay kaç ayakkabı bağıçına ihtiyaç duyacağını tahmin etmek için oluşturdukları bir modeli gözden geçirmektir. Makine öğrenimi modeline giren özellikler şunları içerir:

- Geçerli ay (Ocak, Şubat vb.)
- Önceki ayın reklam harcamaları
- Cari ayın başından itibaren çeşitli makroekonomik özellikler (ıssızlık oranı gibi)
- İçinde bulundukları ayda kullandıkları deri miktarı

Sonuçlar, ne kadar deri kullandıklarıyla ilgili özelliği eklerseniz modelin neredeyse mükemmel olduğunu gösterir. Ancak bu özelliği dışında bırakırsanız yalnızca orta derecede doğrudur. Bunun, kullandıkları deri miktarının kaç ayakkabı üretiklerinin mükemmel bir göstergesi olduğunun farkındasınız, bu da size kaç ayakkabı bağına ihtiyaç duyduklarını söyler.

Kullanılan deri özelliğinin bir veri sizıntısı kaynağı olduğunu düşünüyorsunuz? Cevabınız "duruma bağlı" ise neye bağlıdır?

Cevabınızı düşündükten sonra, aşağıdaki çözüme karşı kontrol edin.

Solution:

Bu zordur ve verilerin nasıl toplandığına (sizıntıyı düşünürken yaygın olan) bağlıdır. Ayın başında o ay ne kadar deri kullanılacağına karar verir misiniz? Eğer öyleyse, bu tamam. Ancak bu ay boyunca belirlenirse, tahmin yaptığınızda buna erişemezsiniz. Ayın başında bir tahmininiz varsa ve daha sonra ay boyunca değiştirilirse, ay boyunca kullanılan gerçek miktar bir özellik olarak kullanılamaz (çünkü sizıntıya neden olur).

2. *Return of the Shoelaces(Ayakkabı Bağcıklarının dönüşü)*

Yeni bir fikrin var. Nike'nin sipariş ettiği deri miktarını (gerçekte kullandıkları miktardan ziyade) ayakkabı bağı modelinizde öngörücü olarak belirli bir aya kadar kullanabilirsiniz.

Bu, bir sizıntı problemi olup olmadığı hakkındaki cevabınızı değiştiriyor mu? Eğer "buna bağlıdır" diye cevap verirseniz, neye bağlıdır?

Solution:

Bu iyi olabilir, ancak önce ayakkabı bağcığı mı yoksa önce deri sipariş etmelerine bağlıdır. Önce ayakkabı bağcığı sipariş ederse, ayakkabı bağı ihtiyaçlarını tahmin ettiğinizde ne kadar deri sipariş ettiklerini bilemezsiniz. İlk önce deri sipariş ederse, ayakkabı bağı siparişinizi verdığınızda bu numaraya sahip olacaksınız ve iyi olmalısınız.

3. Getting Rich With Cryptocurrencies?(Kripto Para Birimleri İle Zengin Olmak Mi?)

Nike'ı o kadar çok para kazandırdın ki sana bir bonus verdiler. Tebrikler.

Aynı zamanda bir veri bilimcisi olan arkadaşınız, bonusunuzu milyonlarca dolara dönüştürmenize izin verecek bir model geliştirdiğini söylüyor. Özellikle, modeli tahmin anından bir gün önce yeni bir kripto para biriminin (Bitcoin gibi, ancak daha yeni gibi) fiyatını tahmin ediyor. Planı, para biriminin fiyatının (dolar cinsinden) yükselmek üzere olduğunu söylediğinde kripto para birimini satın almaktır.

Modelindeki en önemli özellikler:

- Geçerli para birimi fiyatı
- Son 24 saat içinde satılan para birimi miktarı
- Son 24 saat içinde para birimi fiyatındaki değişim
- Son 1 saat içinde para birimi fiyatındaki değişim
- Para biriminden bahseden son 24 saat içindeki yeni tweet sayısı

Dolar cryptocurrency değeri Aşağı Yukarı 100 üzerinden

1 in the last year, and yet this model's average error is less than tarafından dalgalanma vardır. Bunun, modelinin doğru olduğunun kanıtı olduğunu ve model yükselmek üzere olduğunu söylediğinde para birimini satın alarak onunla yatırım yapmanız gerektiğini söylüyor.

Haklı mı? Modeliyle ilgili bir sorun varsa, nedir?

Solution: burada sizıntı kaynağı yok. Bu özellikler, bir ön hazırlık yapmak istediğiniz anda mevcut olmalıdır ve tahmin hedefi belirlendikten sonra eğitim verilerinde değiştirilmesi olası değildir. Ancak, dikkatli değilseniz, doğruluğu nasıl tanımladığı yanıldıcı olabilir. Fiyat yavaş yavaş hareket ederse, bugünkü fiyatı yarının fiyatının doğru bir belirleyicisi olacaktır, ancak yatırım yapmak için iyi bir zaman olup olmadığını size söylemeyebilir. Örneğin, eğer öyleyse *100 today, a model predicting a price of 100* yılın, fiyatın mevcut fiyattan yukarı veya aşağı gidip gitmediğini söyleyemese bile doğru görünebilir. Daha iyi bir tahmin hedefi, ertesi gün fiyatındaki değişim olacaktır. Fiyatın yukarı veya aşağı gitmek üzere olup olmadığını (ve ne kadar) sürekli olarak tahmin edebiliyorsanız, kazanan bir yatırım fırsatına sahip olabilirsiniz.

4. Preventing Infections(Enfeksiyonları Önleme)

Sağlık hizmeti veren bir kurum, nadir bir cerrahiden hangi hastaların enfeksiyon riski altında olduğunu tahmin etmek ister, bu nedenle hemşireleri bu hastaları takip ederken özellikle dikkatli olmaları konusunda uyarabilir.

Bir model oluşturmak istiyorsun. Modelleme veri kümesindeki her satır, ameliyatı alan tek bir hasta olacak ve tahmin hedefi, bir enfeksiyon olup olmadığı olacaktır.

Bazı cerrahlar prosedürü enfeksiyon riskini artıracak veya azaltacak şekilde yapabilir. Ancak cerrah bilgilerini modele en iyi nasıl dahil edebilirsiniz?

Zekice bir fikrin var.

1. Her cerrahın tüm ameliyatlarını alın ve bu cerrahlar arasındaki enfeksiyon oranını hesaplayın.
2. Verilerdeki her hasta için, cerrahın kim olduğunu bulun ve o cerrahın ortalama enfeksiyon oranını bir özellik olarak takın.

Bu herhangi bir hedef sizıntı sorunu yaratıyor mu?

Herhangi bir train testi Kontaminasyon sorunu var mı?

Solution:

Bu, hem hedef sizıntı hem de train testi Kontaminasyon riski oluşturur (ancak dikkatli olursanız her ikisinden de kaçınabilirsiniz).

Belirli bir hastanın sonucu cerrahi için enfeksiyon oranına katkıda bulunursa, hedef sizıntıya sahip olursunuz, bu da o hastanın enfekte olup olmadığı için tahmin modeline geri takılır. Cerrahın enfeksiyon oranını sadece tahmin ettiğimiz hastadan önce yapılan ameliyatları kullanarak hesaplarsanız, hedef sizıntısını önleyebilirsiniz. Egzersiz verilerinizdeki her ameliyat için bunu hesaplamak biraz zor olabilir.

Ayrıca, test setinden olanlar da dahil olmak üzere bir cerrahın gerçekleştirdiği tüm ameliyatları kullanarak bunu hesaplarsanız, bir train testi kontaminasyon probleminiz vardır. Sonuç, model yerleştirildikten sonra yeni hastalara iyi genelleştirilmese bile, modelinin test setinde çok doğru görünebileceğidir. Bunun nedeni, cerrahın risk özelliğinin test kümesindeki verileri hesaba katmasıdır. Yeni verileri görüntülerken modelin nasıl çalışacağını tahmin etmek için Test setleri vardır. Bu nedenle, bu kirlilik test setinin amacını bozar.

5. *Housing Prices*

Konut fiyatlarını tahmin etmek için bir model oluşturacaksınız. Model, bir web sitesine bir açıklama ekendiğinde yeni bir evin fiyatını tahmin etmek için sürekli olarak dağıtılacaktır. İşte yordayıcı olarak kullanılabilecek dört özellik.

- Evin büyüklüğü (metrekare)
- Aynı mahalledeki evlerin ortalama satış fiyatı
- Evin enlem ve boylamı
- Evin bodrum katının olup olmadığı

Modeli eğitmek ve doğrulamak için geçmiş verileriniz var.

Özelliklerden hangisinin bir sizıntı kaynağı olması muhtemeldir?

```
[6]: # Aşağıdaki satırı 1, 2, 3 veya 4'ten biriyle doldurun.
potential_leakage_feature = 2

# Check your answer
q_5.check()
```

hedef 2 sizıntısının kaynağıdır. İşte her özellik için bir analiz:

1. Bir evin büyüklüğünün satıldıktan sonra değiştirilmesi olası değildir (teknik olarak mümkün olsa da). Ancak tipik olarak bu bir tahmin yapmamız gereğinde kullanılabilir ve veriler ev satıldıktan sonra değiştirilmez. Bu yüzden oldukça güvenlidir.

2. Bunun ne zaman güncellendiğini bilmiyoruz. Bir ev satıldıktan sonra ham verilerde alan güncellenir ve ortalamanın hesaplanması için evin satışı kullanılırsa, bu bir hedef sizıntısı vakasını oluşturur. Bir

ucta, mahallede sadece bir ev satılıyorsa ve tahmin etmeye çalıştığımız ev ise, o zaman ortalama tahmin etmeye çalıştığımız değere tam olarak eşit olacaktır. Genel olarak, az satış yapılan mahalleler için model, eğitim verileri üzerinde çok iyi performans gösterecektir. Ancak modeli uyguladığınızda, tahmin ettiğiniz ev henüz satılmayacaktır, bu nedenle bu özellik eğitim verilerinde olduğu gibi çalışmaz.

3.Bunlar değişmez ve bir tahmin yapmak istediğimiz zaman hazır olur. Yani burada hedef sizıntı riski yoktur.

4.Bu da değişmez ve bir tahmin yapmak istediğimiz anda kullanılabilir. Yani burada hedef sizıntı riski yoktur.

Conclusion

Sızıntı zor ve ince bir konudur. Bu örneklerde bu konuları ele alıysanız gurur duymalısınız.

Artık son derece hassas modeller yapmak için araçlara sahipsiniz ve gerçek sorunları çözmek için bu modelleri uygulamakla ortaya çıkan en zor pratik problemleri toplayın.

Bilgi ve deneyim oluşturmak için hala çok yer var. Bir Makine Öğrenimi Yarışması deneyin veya yeni becerilerinizi uygulamak için Veri Kümelerimize bakın.

Tekrar Tebrikler!

Quiz2:

S1-çapraz doğrulamanın amaçlanan kullanımını hakkında aşağıdaki ifadelerden hangisi doğrudur?

- I-model performansını ölçerken rastgeleliği azaltmak için.
 - II-model performansının daha iyi bir ölçüsünü elde etmek.
 - III-modelin eğitim performansını artırmak.
 - IV-Mae (ortalama mutlak hata) veya MSE (ortalama Kare hata) artırmak için.
-
- I, II, IV
 - II, III
 - I, II ✓
 - All of them

A1- Cross-validation kullanmadıkta amaç modelimizde kullandığımız metrikleri daha doğru bir şekilde gözlemlileyebilmektir. Dolayısı ile modelimizin hatasını düşürmesi veya modeli daha iyi eğitmemiz üzerinde doğrudan bir etkisi yoktur.

S2-Labellncoder ve OneHotEncoder hakkında aşağıdaki ifadelerden hangisi doğrudur?

- Ben kategorik değerler ile başa çıkmak için bize yardımcı olur.
- II-Label kodlaması, benzersiz olsun ya da olmasın, her değeri farklı bir tamsayıya atar.
- III - bir sıcak kodlama, orijinal verideki her olası değer için yeni bir sütun oluşturur.
- IV-çok sayıda kategorik değişken sayısı değeri için (15 farklı değer gibi) kullanmak iyi değildir
Genellikle bir sıcak kodlayıcı.

- I, II, IV
- I, III, IV ✓
- I, II, III
- All of them

A2- Label Encoder ve One Hot Encoding kategorik verilerin üstesinden gelmek için kullanılırlar.
Label Encoding her eşsiz (unique) değer için bir değer üretip atama yapar. One Hot Encoding ise her değer için yeni bir kolon oluşturur. Bu değerler eşsiz (unique) değerlerdir. Kolon sayısı arttıkça, genelde One Hot Encoding iyi bir performans sergilemez. Bu yüzden One Hot Encoding genelde fazla sayıdaki unique kolon içeren kategorik verilerle kullanıldığında iyi sonuç vermez.

S3-aşağıdaki ifadelerden hangisi boru hatlarıyla tutarsız?

Pipelines ile, bir ön işleme adımını unutmak için daha az olasılık vardır

Pipelines ile bir model üretmek zor.

Bir pipeline ile her adımda eğitim ve doğrulama verilerinizi manuel olarak izlemeniz gerekmekz.

Bir pipeline ile çapraz doğrulama tekniğini kolayca kullanabiliriz.

A3- Pipelineleri, modelimize input olarak verecek datanın her zaman aynı işlemlerden geçirilmesi, ön-işlemde meydana gelebilecek hata ve eksiklik risklerinin azaltılması ve cross-validation gibi model değerlendirmesi yaptığım işlevleri kolayca yapabilmek için kullanıyoruz. Modelleri pipelineler ile oluşturmak zor değil ve model deployment aşamasında hata yapmanızı büyük ölçüde engelleyeceğinden dolayı oldukça kullanışlılar.

Q4- print(df.head).method()

Üst 10 satırındaki eksik değerlerin konumlarını yazdırınızı varsayıyalım. Hangi yöntem bunun için uygun mu?

- dropna(how='any')
- isnan
- notnull
- isnull ✓

A4- Bir veri çerçevesinde NULL değerlerini denetlemek ve yönetmek için `isnull()` ve `notnull()` yöntemleri kullanılır. `isnull()` yöntemi, NaN değeri için True ve boş olmayan değer için False döndürür. `notnull()` bu durumun tam tersidir.

Q5-aşağıdakilerden hangisi Xgboost'un bir güçlendirici parametresi değildir?

- `min_child_weight`
- `objective` ✓
- `max_leaf_nodes`
- `colsample_bylevel`

A5- “`objective`” parametresi bir learning task parametresidir. Bunun gibi parametreler, her adımda hesaplanacak

metriğin optimizasyon hedefini tanımlamak için kullanılır.

S6-vurgulanan kod parçaları ne anlama geliyor?

```
X_train_plus = X_train.copy()
X_valid_plus = X_valid.copy()
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()
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))
imputed_X_train_plus.columns = X_train_plus.columns
imputed_X_valid_plus.columns = X_valid_plus.columns
```

Ne olacağını gösteren yeni sütunlar yapmak

imputation için

Orijinal verileri değiştirmekten kaçınmak için kopya yapmak için

Kaldırılan sütun adlarını geri koymak için

A6- Yukarıdaki code parçası kayıp verileri işlemeye kullanılan bir yöntem olan Imputation adımlarını ifade etmektedir. İşaretli satırlar da, imputing işlemi sırasında kayıp verileri çıkarılmış kolonları, temizlenmiş olarak geri almamızı sağlar.

S7-aşağıdaki değişkenlerden hangisi nominal değişken (ler) dir?

I-Cinsiyet

II-genotip

III-dini tercih

IV-IQ

V-Bir hafta içinde kazanılan gelir.

- I, II
- I, II, III ✓
- II, III, IV
- All of them

A7- Nominal değişkenler aralarında sıralama yapılamayan kategorik değişkenlerdir. Cinsiyet, genotip ve dini tercihler değerlerinin birbirlerine herhangi bir üstünlüğü bulunmayan değişkenlerdir. IQ ve haftalık kazanç kategorik değişkenler olmadığından nominal değişken olarak değerlendirilemezler.

Q8- Which of the following statements are true about “max_depth” hyperparameter in Random Forest?

- I- Lower is better parameter in case of same validation accuracy
 - II- Higher is better parameter in case of same validation accuracy
 - III- Increase the value of max_depth may overfit the data
 - IV- Increase the value of max_depth may underfit the data
-
- I, IV
 - II, IV
 - I, III ✓
 - II, III

A8- Çünkü maksimum derinliği gereğinden fazla artırmamız modelimizin veriyi ezberlemesine ve overfit olmasına yol açar. Farklı derinlikler ile oluşturduğumuz modellerden aynı skoru alırsak modelimiz karmaşıklığını azaltmak için düşük derinlikli olanı tercih etmemiz gereklidir.

S9-konut fiyatlarını tahmin etmek için bir model oluşturacaksınız. Modeli üzerinde dağıtıllacak

sürekli olarak, bir web sitesine bir açıklama eklendiğinde yeni bir evin fiyatını tahmin etmek.

İşte yordayıcı olarak kullanılabilecek dört özellik. Bu özelliklerden hangisi en olası bir sizıntı kaynağı olabilir mi?

- Evin büyülüğu (metrekare cinsinden)
- Aynı mahallede evlerin ortalama satış fiyatı
- Evin enlem ve boylamı
- Evin bir bodrum katı var mı

A9- Data leakage (veri sizıntısı), eğitim verileri hedef hakkında bilgi içerdiginde gerçekleşir, ancak model tahmini için kullanıldığından benzer veriler kullanılamaz. Bu, eğitim setinde (ve hatta muhtemelen doğrulama verilerinde) yüksek performansa yol açar, ancak model üretimde kötü performans gösterecektir.

Başa bir deyişle, karar verme mekanizması başlayana kadar o model çok doğru görünür fakat en sonunda modelin çok yanlış kurulduğu ortaya çıkar.

- 1- Bir evin büyülüğünün satıldıktan sonra değiştirilmesi olası değildir (teknik olarak mümkün olsa da). Ancak tipik olarak bu bir tahmin yapmamız gerekiğinde kullanılabilir ve veriler ev satıldıktan sonra değiştirilmez. Bu yüzden oldukça güvenlidir.
- 2- Bunun ne zaman güncellendiğini bilmiyoruz. Bir ev satıldıktan sonra ham verilerde alan güncellenirse ve ortalamanın hesaplanması için evin satışı kullanılırsa, bu veri sizıntısı anlamına gelir. Bir ucta, mahallede sadece bir ev satılıyorsa ve tahmin etmeye çalıştığımız ev ise, o zaman ortalama tahmin etmeye çalıştığımız değere tam olarak eşit olacaktır. Genel olarak, az satış yapılan mahalleler için model, eğitim verileri üzerinde çok iyi performans gösterecektir. Ancak modeli uyguladığınızda, tahmin ettiğiniz ev henüz satılmayacaktır, bu nedenle bu özellik eğitim verilerinde olduğu gibi çalışmaz.
- 3- Bunlar değişmez ve bir tahmin yapmak istediğimiz zaman hazır olur. Yani burada veri sizıntı riski yoktur.
- 4- Bu da değişmez ve bir tahmin yapmak istediğimiz anda kullanılabilir. Yani burada veri sizıntı riski yoktur.

Q10- How is the Gradient Boosting cycle proceed? Please choose the correct order from the mixed statements below.

- I- We add the new model to ensemble.
- II- We use the current ensemble to generate predictions for each observation in the dataset.
- III- We use the loss function to fit a new model that will be added to the ensemble.

- I-II-III
- I-III-II
- II-I-III
- II-III-I ✓

A10- Gradient boosting döngüsünde ilk olarak, veri grubundaki her bir gözlem için tahminler oluşturmak üzere mevcut topluluğu (ensemble) kullanıyoruz. Bir tahmin yapmak için, topluluktaki tüm modellerden tahminleri ekliyoruz. Bu tahminler bir kayıp fonksiyonunu (loss function) hesaplamak için kullanılır.

Daha sonra loss function i, topluluğa (ensemble) eklenecek yeni bir modele uyacak şekilde kullanıyoruz. Özellikle, model parametrelerini belirlemeye kullanıyoruz ki böylece bu yeni modeli topluluğa eklemekle olası zaman kayıplarını azaltıyoruz. Son olarak da topluluğa yeni modeli ekliyoruz.

KAYNAKLAR

- Kaggle – Intro to Machine Learnin Course
<https://www.kaggle.com/learn/intro-to-machine-learning>
- <https://medium.com/data-science-tr/overfitting-underfitting-cross-validation-b47dfda0cf4e>
- <http://www.veridefteri.com/2017/11/23/scikit-learn-ile-veri-analitigine-giris/>
- <https://www.slideshare.net/VolkanOBANMsc/python-rastgele-ormanrandom-forest-parametreleri>
- <https://medium.com/@ahmetkuzubasli/modeliniz-neden-hala-hatal%C4%B1-bias-ve-variance-6368f36de751>
- <https://stackoverflow.com/questions/53249603/random-state-and-shuffle-together>

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
- <https://www.slideshare.net/VolkanOBANMsc/python-rastgele-ormanrandom-forest-parametreleri>
- <https://thepythonguru.com/python-built-in-functions/zip/>
- <https://www.sharpsightlabs.com/blog/pandas-dropna/>
- <https://www.kaggle.com/dansbecker/using-categorical-data-with-one-hot-encoding>