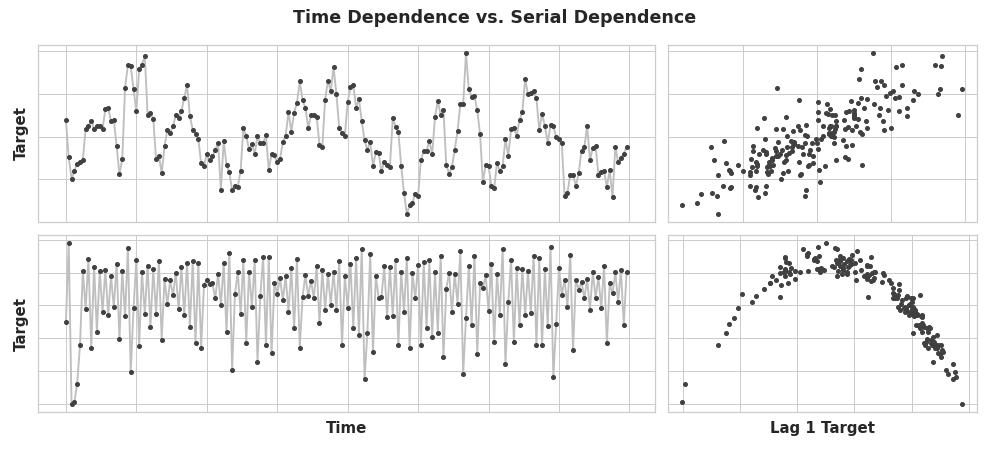
What is Serial Dependence?

Önceki derslerde, **zaman bağımlı** özellikler olarak en kolay şekilde modellenebilen zaman serisi özelliklerini inceledik; yani, doğrudan zaman indeksinden türetebileceğimiz özelliklerle. Ancak, bazı zaman serisi özellikleri yalnızca **ardışık bağımlı** özellikler olarak modellenebilir; yani, hedef serinin geçmiş değerlerini özellik olarak kullanarak. Bu zaman serilerinin yapısı, zaman içinde çizilen bir grafikten belirgin olmayabilir; ancak, geçmiş değerlere karşı çizildiğinde, yapı netleşir - tıpkı aşağıdaki şekilde gördüğümüz gibi.

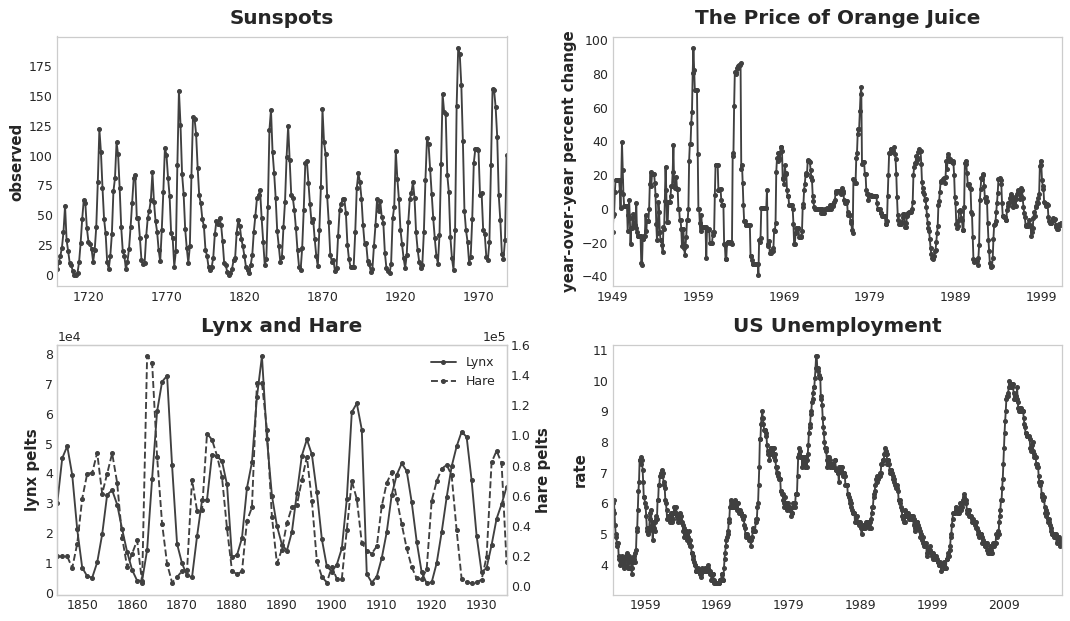


Bu iki serinin seri bağımlılığı vardır, ancak zaman bağımlılığı yoktur. Sağdaki noktaların koordinatları vardır (t-1 anındaki değer, t anındaki değer).

Trend ve mevsimsellik ile, yukarıdaki şekilde soldaki grafikler gibi çizimlere eğriler uydurmak için modeller eğittik; yani modeller **zaman bağımlılığını** öğreniyordu. Bu dersteki amacımız, sağdaki grafikler gibi çizimlere eğriler uydurmak için modeller eğitmektir; yani onların **ardışık bağımlılığı** öğrenmesini istiyoruz.,

### **Cycles[¶](https://www.kaggle.com/code/ryanholbrook/time-series-as-features" \l "Cycles" \t "_self)**

Ardışık bağımlılığın ortaya çıkmasının özellikle yaygın bir yolu **döngülerdir**. Döngüler, bir serideki bir zamandaki değerin, mutlaka zaman adımının kendisine değil, önceki zamanlardaki değerlere nasıl bağlı olduğuyla ilişkili, bir zaman serisindeki büyüme ve çöküş kalıplarıdır. Döngüsel davranış; kendini etkileyebilen veya tepkileri zaman içinde devam eden sistemlerin bir özelliğidir. Ekonomiler, salgın hastalıklar, hayvan popülasyonları, volkanik patlamalar ve benzeri doğal olaylar genellikle döngüsel davranışlar sergiler.



Four time series with cyclic behavior.

Döngüsel davranışı mevsimsellikten ayıran şey, döngülerin, mevsimlerin aksine, mutlaka zamana bağımlı olmamasıdır. Bir döngüde olanlar, belirli bir meydana gelme tarihinden ziyade, yakın geçmişte ne olduğuna daha çok bağlıdır. Zamandan (en azından göreceli olarak) bağımsız olması, döngüsel davranışın mevsimsellikten çok daha düzensiz olabileceği anlamına gelir.

# Lagged Series and Lag Plots[¶](https://www.kaggle.com/code/ryanholbrook/time-series-as-features" \l "Lagged-Series-and-Lag-Plots" \t "_self)

Bir zaman serisinde olası ardışık bağımlılığı (döngüler gibi) araştırmak için, serinin "gecikmeli" kopyalarını oluşturmamız gerekir. Bir zaman serisini **gecikme (lag)**, değerlerini bir veya daha fazla zaman adımı ileriye kaydırmak veya eşdeğer olarak, indeksindeki zamanları bir veya daha fazla adım geriye kaydırmak anlamına gelir. Her iki durumda da, gecikmeli serideki gözlemler, zaman içinde daha sonra gerçekleşmiş gibi görünür.

Bu, ABD'deki aylık işsizlik oranını (y) ve onun birinci ve ikinci gecikmeli serilerini (y\_lag\_1 ve y\_lag\_2) birlikte göstermektedir. Gecikmeli serilerin değerlerinin zaman içinde nasıl ileriye kaydırıldığına dikkat edin.

import pandas as pd

*# Federal Reserve dataset: https://www.kaggle.com/federalreserve/interest-rates*

reserve = pd.read\_csv(

"../input/ts-course-data/reserve.csv",

parse\_dates={'Date': ['Year', 'Month', 'Day']},

index\_col='Date',

)

y = reserve.loc[:, 'Unemployment Rate'].dropna().to\_period('M')

df = pd.DataFrame({

'y': y,

'y\_lag\_1': y.shift(1),

'y\_lag\_2': y.shift(2),

})

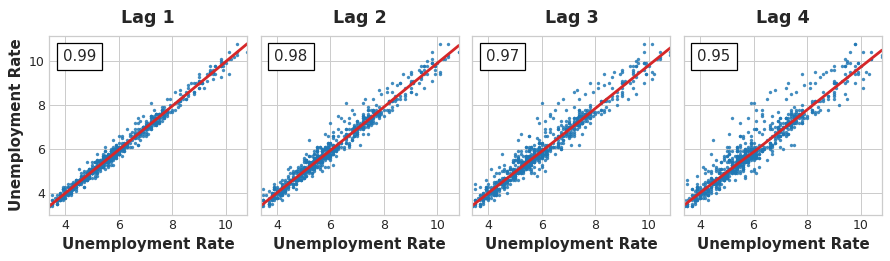
df.head()

| y | y\_lag\_1 | y\_lag\_2 |
| --- | --- | --- |
| Date |  |  |  |
| 1954-07 | 5.8 | NaN | NaN |
| 1954-08 | 6.0 | 5.8 | NaN |
| 1954-09 | 6.1 | 6.0 | 5.8 |
| 1954-10 | 5.7 | 6.1 | 6.0 |
| 1954-11 | 5.3 | 5.7 | 6.1 |

Bir zaman serisini geciktirerek, geçmiş değerlerini tahmin etmeye çalıştığımız değerlerle eş zamanlı hale getirebiliriz (başka bir deyişle, aynı satırda görünmelerini sağlarız). Bu durum, gecikmeli serileri ardışık bağımlılığı modellemek için özellik olarak kullanışlı hale getirir. ABD işsizlik oranı serisini tahmin etmek için, y\_lag\_1 ve y\_lag\_2 özelliklerini, hedef y değerini tahmin etmek üzere kullanabiliriz. Bu, gelecekteki işsizlik oranını, önceki iki ayın işsizlik oranının bir fonksiyonu olarak tahmin edecektir.

### **Lag plots[¶](https://www.kaggle.com/code/ryanholbrook/time-series-as-features" \l "Lag-plots" \t "_self)**

Bir zaman serisinin **gecikme grafiği (lag plot)**, serinin değerlerini gecikme değerlerine karşı gösterir. Bir zaman serisindeki ardışık bağımlılık, bir gecikme grafiğine bakarak genellikle belirgin hale gelir. **ABD İşsizlik Oranı**'nın bu gecikme grafiğinden, mevcut işsizlik oranı ile geçmiş oranlar arasında güçlü ve görünüşe göre doğrusal bir ilişki olduğunu görebiliriz.



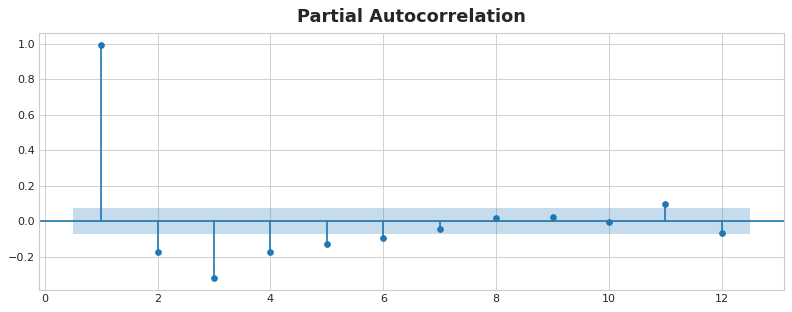
Otokorelasyonların gösterildiği ABD İşsizliğinin gecikme grafiği.

Ardışık bağımlılığın en sık kullanılan ölçüsü **otokorelasyon** olarak bilinir. Bu, bir zaman serisinin kendi gecikmelerinden biriyle olan korelasyonudur. **ABD İşsizlik Oranı**'nın, 1 gecikmesinde 0.99, 2 gecikmesinde 0.98 ve bu şekilde devam eden bir otokorelasyonu vardır.

### **Choosing lags[¶](https://www.kaggle.com/code/ryanholbrook/time-series-as-features" \l "Choosing-lags" \t "_self)**

Özellik olarak kullanılacak gecikmeleri seçerken, genellikle yüksek otokorelasyona sahip **her** gecikmeyi dahil etmek faydalı olmayacaktır. Örneğin, **ABD İşsizlik Oranı**'nda, 2. gecikmedeki otokorelasyon, tamamen 1. gecikmedeki "bozulmuş" bilgilerden kaynaklanabilir; yani bir önceki adımdan taşınan bir korelasyon olabilir. Eğer 2. gecikme yeni bir bilgi içermiyorsa, 1. gecikme zaten elimizdeyken onu dahil etmek için bir sebep yoktur.

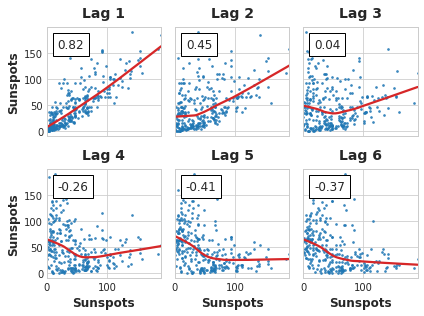
**Kısmi otokorelasyon**, size tüm önceki gecikmeleri hesaba katarak bir gecikmenin korelasyonunu, yani o gecikmenin kattığı "yeni" korelasyon miktarını söyler. Kısmi otokorelasyonu grafiğe dökmek, hangi gecikme özelliklerini kullanacağınızı seçmenize yardımcı olabilir. Aşağıdaki şekilde, 1'den 6'ya kadar olan gecikmeler "korelasyon yok" aralıklarının (mavi renkte) dışında kalmaktadır, bu yüzden **ABD İşsizlik Oranı** için 1'den 6'ya kadar olan gecikmeleri özellik olarak seçebiliriz. (11. gecikmenin yanlış pozitif olması muhtemeldir.)



ABD İşsizliğinin 12. gecikmeye kadar kısmi otokorelasyonları, %95 güven aralıklarında korelasyon yok.

Yukarıdaki gibi bir grafiğe **korelogram** denir. Korelogram, gecikme (lag) özellikleri için, esasen periyodogramın Fourier özellikleri için olduğu şeydir.

Son olarak, otokorelasyon ve kısmi otokorelasyonun **doğrusal** bağımlılık ölçüleri olduğunu unutmamız gerekir. Gerçek dünya zaman serileri genellikle önemli doğrusal olmayan bağımlılıklara sahip olduğundan, gecikme özelliklerini seçerken bir gecikme grafiğine bakmak (veya **karşılıklı bilgi** gibi daha genel bir bağımlılık ölçüsü kullanmak) en iyisidir. **Güneş Lekeleri (Sunspots)** serisi, otokorelasyon ile gözden kaçırabileceğimiz doğrusal olmayan bağımlılığa sahip gecikmelere sahiptir.



Lag plot of the *Sunspots* series.

Bu gibi doğrusal olmayan ilişkiler ya doğrusal hale dönüştürülebilir ya da uygun bir algoritma ile öğrenilebilir.

# Example - Flu Trends[¶](https://www.kaggle.com/code/ryanholbrook/time-series-as-features" \l "Example---Flu-Trends" \t "_self)

**Flu Trends** veri seti, 2009 ve 2016 yılları arasında grip için yapılan doktor ziyaretlerinin kayıtlarını içerir. Amacımız, önümüzdeki haftalar için grip vakalarının sayısını tahmin etmektir.

İki farklı yaklaşım izleyeceğiz. İlk yaklaşımda, doktor ziyaretlerini gecikme (lag) özelliklerini kullanarak tahmin edeceğiz. İkinci yaklaşımımız ise, başka bir zaman serisi grubunun gecikmelerini kullanarak doktor ziyaretlerini tahmin etmek olacak: Google Trends tarafından yakalanan, gribe ilişkin arama terimlerini kullanacağız.

from pathlib import Path

from warnings import simplefilter

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

from scipy.signal import periodogram

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from statsmodels.graphics.tsaplots import plot\_pacf

simplefilter("ignore")

*# Set Matplotlib defaults*

plt.style.use("seaborn-whitegrid")

plt.rc("figure", autolayout=True, figsize=(11, 4))

plt.rc(

"axes",

labelweight="bold",

labelsize="large",

titleweight="bold",

titlesize=16,

titlepad=10,

)

plot\_params = dict(

color="0.75",

style=".-",

markeredgecolor="0.25",

markerfacecolor="0.25",

)

%config InlineBackend.figure\_format = 'retina'

def lagplot(x, y=None, lag=1, standardize=False, ax=None, \*\*kwargs):

from matplotlib.offsetbox import AnchoredText

x\_ = x.shift(lag)

if standardize:

x\_ = (x\_ - x\_.mean()) / x\_.std()

if y **is** **not** None:

y\_ = (y - y.mean()) / y.std() if standardize else y

else:

y\_ = x

corr = y\_.corr(x\_)

if ax **is** None:

fig, ax = plt.subplots()

scatter\_kws = dict(

alpha=0.75,

s=3,

)

line\_kws = dict(color='C3', )

ax = sns.regplot(x=x\_,

y=y\_,

scatter\_kws=scatter\_kws,

line\_kws=line\_kws,

lowess=True,

ax=ax,

\*\*kwargs)

at = AnchoredText(

f"**{**corr**:**.2f**}**",

prop=dict(size="large"),

frameon=True,

loc="upper left",

)

at.patch.set\_boxstyle("square, pad=0.0")

ax.add\_artist(at)

ax.set(title=f"Lag **{**lag**}**", xlabel=x\_.name, ylabel=y\_.name)

return ax

def plot\_lags(x, y=None, lags=6, nrows=1, lagplot\_kwargs={}, \*\*kwargs):

import math

kwargs.setdefault('nrows', nrows)

kwargs.setdefault('ncols', math.ceil(lags / nrows))

kwargs.setdefault('figsize', (kwargs['ncols'] \* 2, nrows \* 2 + 0.5))

fig, axs = plt.subplots(sharex=True, sharey=True, squeeze=False, \*\*kwargs)

for ax, k **in** zip(fig.get\_axes(), range(kwargs['nrows'] \* kwargs['ncols'])):

if k + 1 <= lags:

ax = lagplot(x, y, lag=k + 1, ax=ax, \*\*lagplot\_kwargs)

ax.set\_title(f"Lag **{**k + 1**}**", fontdict=dict(fontsize=14))

ax.set(xlabel="", ylabel="")

else:

ax.axis('off')

plt.setp(axs[-1, :], xlabel=x.name)

plt.setp(axs[:, 0], ylabel=y.name if y **is** **not** None else x.name)

fig.tight\_layout(w\_pad=0.1, h\_pad=0.1)

return fig

data\_dir = Path("../input/ts-course-data")

flu\_trends = pd.read\_csv(data\_dir / "flu-trends.csv")

flu\_trends.set\_index(

pd.PeriodIndex(flu\_trends.Week, freq="W"),

inplace=True,

)

flu\_trends.drop("Week", axis=1, inplace=True)

ax = flu\_trends.FluVisits.plot(title='Flu Trends', \*\*plot\_params)

\_ = ax.set(ylabel="Office Visits")

**Flu Trends** verilerimiz, düzenli bir mevsimsellik yerine düzensiz döngüler gösteriyor: Zirve genellikle yılbaşı civarında gerçekleşiyor, ancak bazen daha erken veya daha geç, bazen daha büyük veya daha küçük olabiliyor. Bu döngüleri gecikme (lag) özellikleriyle modellemek, tahmincimizin mevsimsel özelliklerdeki gibi belirli tarih ve saatlere bağlı kalmak yerine, değişen koşullara dinamik olarak tepki vermesini sağlayacaktır.

Öncelikle gecikme ve otokorelasyon grafiklerine bir göz atalım:

\_ = plot\_lags(flu\_trends.FluVisits, lags=12, nrows=2)

\_ = plot\_pacf(flu\_trends.FluVisits, lags=12)

Gecikme grafikleri, **FluVisits**'in kendi gecikmeleriyle olan ilişkisinin çoğunlukla doğrusal olduğunu gösterirken, kısmi otokorelasyonlar bu bağımlılığın 1, 2, 3 ve 4. gecikmeler kullanılarak yakalanabileceğini düşündürmektedir. Bir zaman serisini Pandas'ta **shift** metoduyla geciktirebiliriz. Bu problem için, geciktirmenin oluşturduğu eksik değerleri **0.0** ile dolduracağız.

def make\_lags(ts, lags):

return pd.concat(

{

f'y\_lag\_**{**i**}**': ts.shift(i)

for i **in** range(1, lags + 1)

},

axis=1)

X = make\_lags(flu\_trends.FluVisits, lags=4)

X = X.fillna(0.0)

Önceki derslerde, eğitim verilerinin ötesinde istediğimiz kadar adım için tahminler yapabiliyorduk. Ancak, gecikme (lag) özellikleri kullanırken, yalnızca gecikmeli değerleri mevcut olan zaman adımları için tahmin yapmakla sınırlıyız. Pazartesi günü 1. gecikme özelliğini kullanarak, Çarşamba için bir tahmin yapamayız, çünkü gerekli olan 1. gecikme değeri, henüz gerçekleşmemiş olan Salı günüdür.

Bu sorunu çözmeye yönelik stratejileri 6. Derste göreceğiz. Bu örnek için, sadece bir test setindeki değerleri kullanacağız.

*# Create target series and data splits*

y = flu\_trends.FluVisits.copy()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=60, shuffle=False)

*# Fit and predict*

model = LinearRegression() *# `fit\_intercept=True` since we didn't use DeterministicProcess*

model.fit(X\_train, y\_train)

y\_pred = pd.Series(model.predict(X\_train), index=y\_train.index)

y\_fore = pd.Series(model.predict(X\_test), index=y\_test.index)

ax = y\_train.plot(\*\*plot\_params)

ax = y\_test.plot(\*\*plot\_params)

ax = y\_pred.plot(ax=ax)

\_ = y\_fore.plot(ax=ax, color='C3')

Sadece tahmin değerlerine baktığımızda, modelimizin hedef serideki ani değişikliklere tepki vermek için bir zaman aralığına ihtiyaç duyduğunu görebiliriz. Bu, yalnızca hedef serinin gecikmelerini özellik olarak kullanan modellerin yaygın bir sınırlamasıdır.

ax = y\_test.plot(\*\*plot\_params)

\_ = y\_fore.plot(ax=ax, color='C3')

Tahmini iyileştirmek için, grip vakalarındaki değişiklikler için bir "erken uyarı" sağlayabilecek **öncü göstergeler** bulmayı deneyebiliriz. Bu nedenle, ikinci yaklaşımımızda, Google Trends tarafından ölçülen, gribe ilişkin bazı arama terimlerinin popülaritesini eğitim verimize ekleyeceğiz.

**'FluCough'** arama terimini, hedefimiz olan **'FluVisits'**'e karşı çizmek, bu tür arama terimlerinin öncü göstergeler olarak faydalı olabileceğini düşündürmektedir: gribe ilişkin aramalar, doktor ziyaretlerinden önceki haftalarda daha popüler hale gelme eğilimindedir.

ax = flu\_trends.plot(

y=["FluCough", "FluVisits"],

secondary\_y="FluCough",

)

Veri setinde 129 tane bu tür terim var, ancak biz sadece birkaçını kullanacağız.

search\_terms = ["FluContagious", "FluCough", "FluFever", "InfluenzaA", "TreatFlu", "IHaveTheFlu", "OverTheCounterFlu", "HowLongFlu"]

*# Create three lags for each search term*

X0 = make\_lags(flu\_trends[search\_terms], lags=3)

X0.columns = [' '.join(col).strip() for col **in** X0.columns.values]

*# Create four lags for the target, as before*

X1 = make\_lags(flu\_trends['FluVisits'], lags=4)

*# Combine to create the training data*

X = pd.concat([X0, X1], axis=1).fillna(0.0)

Tahminlerimiz biraz daha kabataslak, ancak modelimizin grip ziyaretlerindeki ani artışları tahmin etmede daha iyi olduğu görülüyor; bu da arama popülaritesinin birkaç zaman serisinin öncü göstergeler olarak gerçekten etkili olduğunu gösteriyor.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=60, shuffle=False)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = pd.Series(model.predict(X\_train), index=y\_train.index)

y\_fore = pd.Series(model.predict(X\_test), index=y\_test.index)

ax = y\_test.plot(\*\*plot\_params)

\_ = y\_fore.plot(ax=ax, color='C3')

Bu derste gösterilen zaman serileri, "saf döngüsel" olarak adlandırılabilecek türden serilerdir: belirgin bir trend veya mevsimsellikleri yoktur. Ancak, bir zaman serisinin trend, mevsimsellik ve döngülerin hepsini aynı anda içermesi nadir görülen bir durum değildir. Bu tür serileri, her bir bileşen için uygun özellikleri ekleyerek doğrusal regresyonla modelleyebilirsiniz. Hatta her bir bileşeni ayrı ayrı öğrenmek için eğitilmiş modelleri birleştirebilirsiniz; bunun nasıl yapılacağını bir sonraki derste **tahmin hibritleri** ile öğreneceğiz.