Introduction[¶](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Introduction)

Ders 2 ve 3'te, tahmin yapmayı, tüm özelliklerimizi tek bir girdiden, yani **zaman indeksinden** türeterek basit bir regresyon problemi olarak ele aldık. İstediğimiz eğilim ve mevsimsellik özelliklerini oluşturarak gelecekteki herhangi bir zaman için kolayca tahminler yapabiliyorduk.

Ancak, Ders 4'te **gecikme özelliklerini (lag features)** eklediğimizde, problemin doğası değişti. Gecikme özellikleri, tahmin edilen zamanda gecikmeli hedef değerinin bilinmesini gerektirir. Örneğin, bir gecikme-1 özelliği, zaman serisini bir adım ileri kaydırır, bu da geleceğe sadece 1 adım için tahmin yapabileceğiniz anlamına gelir, 2 adım için değil.

Ders 4'te, tahmin etmek istediğimiz döneme kadar her zaman gecikme değerlerini oluşturabileceğimizi varsaymıştık (yani, her tahmin yalnızca bir adım ileriye yönelikti). Gerçek dünyadaki tahminler genellikle bundan daha fazlasını gerektirir. Bu nedenle, bu derste çeşitli durumlar için nasıl tahmin yapılacağını öğreneceğiz.

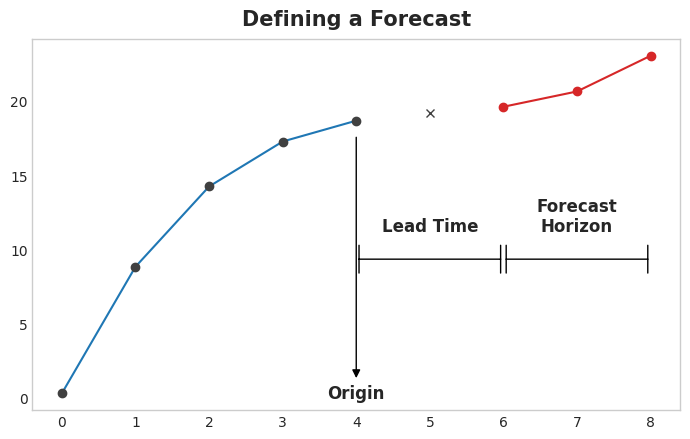
# Defining the Forecasting Task[¶](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Defining-the-Forecasting-Task)

Bir tahmin modeli tasarlamadan önce belirlenmesi gereken iki şey vardır:

* Bir tahmin yapıldığı anda hangi bilgilerin mevcut olduğu (**özellikler**)
* Tahmin edilen değerleri gerektirdiğiniz zaman aralığı (**hedef**)

**Tahmin başlangıcı (forecast origin)**, tahmin yaptığınız andır. Pratik olarak, tahmin başlangıcını, tahmin edilecek zaman için eğitim verilerinizin olduğu son zaman olarak düşünebilirsiniz. Bu başlangıca kadar olan her şey, özellikler oluşturmak için kullanılabilir.

**Tahmin ufku (forecast horizon)**, tahmin yaptığınız zamandır. Tahmini genellikle ufkundaki zaman adımı sayısıyla tanımlarız; örneğin, "1 adımlı" veya "5 adımlı" bir tahmin gibi. Tahmin ufku, hedefi açıklar.



Dört gecikme özelliği kullanılarak, iki adımlı bir ön süreye sahip üç adımlı bir tahmin ufku. Şekil, tek bir eğitim verisi satırını (başka bir deyişle, tek bir tahmin için veri) temsil ediyor.

Orijin ve ufuk arasındaki zamana **öncül zaman (lead time)** veya bazen **gecikme (latency)** denir. Bir tahminin öncül zamanı, orijinden ufka kadar olan adım sayısıyla tanımlanır: örneğin, "1 adım ilerisi" veya "3 adım ilerisi" tahmini gibi. Uygulamada, veri elde etme veya işleme süreçlerindeki gecikmeler nedeniyle bir tahminin orijinin birkaç adım ilerisinden başlaması gerekebilir.

# Preparing Data for Forecasting[¶](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Preparing-Data-for-Forecasting)

Makine öğrenimi algoritmalarıyla zaman serisi tahmini yapmak için, seriyi bu algoritmalarla kullanabileceğimiz bir veri çerçevesine dönüştürmemiz gerekir. (Elbette, sadece trend ve mevsimsellik gibi **deterministik** özellikler kullanmıyorsanız bu geçerlidir.)

Bu sürecin ilk yarısını, gecikmelerden (lags) bir özellik seti oluşturduğumuz Ders 4'te görmüştük. İkinci yarısı ise **hedefi** hazırlamaktır. Bunu nasıl yapacağımız, tahmin görevine bağlıdır.

Bir veri çerçevesindeki her bir satır, tek bir tahmini temsil eder. Satırın zaman indeksi, tahmin ufkunun ilk zamanıdır, ancak tüm ufuk için değerleri aynı satırda düzenleriz. Çok adımlı tahminler için bu, bir modelin her bir adım için bir tane olmak üzere birden fazla çıktı üretmesini gerektirdiğimiz anlamına gelir.

import numpy as np

import pandas as pd

N = 20

ts = pd.Series(

np.arange(N),

index=pd.period\_range(start='2010', freq='A', periods=N, name='Year'),

dtype=pd.Int8Dtype,

)

*# Lag features*

X = pd.DataFrame({

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

'y\_lag\_3': ts.shift(3),

'y\_lag\_4': ts.shift(4),

'y\_lag\_5': ts.shift(5),

'y\_lag\_6': ts.shift(6),

})

*# Multistep targets*

y = pd.DataFrame({

'y\_step\_3': ts.shift(-2),

'y\_step\_2': ts.shift(-1),

'y\_step\_1': ts,

})

data = pd.concat({'Targets': y, 'Features': X}, axis=1)

data.head(10).style.set\_properties(['Targets'], \*\*{'background-color': 'LavenderBlush'}) \

.set\_properties(['Features'], \*\*{'background-color': 'Lavender'})

| Targets | | | Features | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | y\_step\_3 | y\_step\_2 | y\_step\_1 | y\_lag\_2 | y\_lag\_3 | y\_lag\_4 | y\_lag\_5 | y\_lag\_6 |
| Year |  |  |  |  |  |  |  |  |
| 2010 | 2 | 1 | 0 | nan | nan | nan | nan | nan |
| 2011 | 3 | 2 | 1 | nan | nan | nan | nan | nan |
| 2012 | 4 | 3 | 2 | 0 | nan | nan | nan | nan |
| 2013 | 5 | 4 | 3 | 1 | 0 | nan | nan | nan |
| 2014 | 6 | 5 | 4 | 2 | 1 | 0 | nan | nan |
| 2015 | 7 | 6 | 5 | 3 | 2 | 1 | 0 | nan |
| 2016 | 8 | 7 | 6 | 4 | 3 | 2 | 1 | 0 |
| 2017 | 9 | 8 | 7 | 5 | 4 | 3 | 2 | 1 |
| 2018 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 |
| 2019 | 11 | 10 | 9 | 7 | 6 | 5 | 4 | 3 |

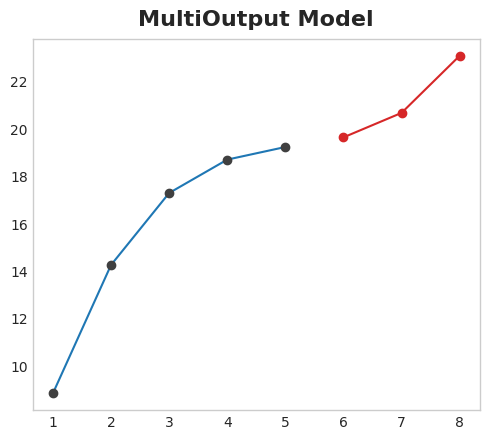
Yukarıdaki açıklama, bir veri setinin **'Tahmini Tanımlama'** figürüne benzer şekilde nasıl hazırlanacağını gösteriyor: beş gecikme özelliği kullanılarak, iki adımlık bir öncül zamanı olan üç adımlı bir tahmin görevi. Orijinal zaman serisi **y\_step\_1**'dir. Eksik değerleri ise ya doldurabilir ya da düşürebiliriz.

# Multistep Forecasting Strategies[¶](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Multistep-Forecasting-Strategies)

Bir tahmin için gereken birden fazla hedef adımı üretmek için çeşitli stratejiler mevcuttur. Her birinin güçlü ve zayıf yönleri olan dört yaygın stratejiyi özetleyeceğiz.

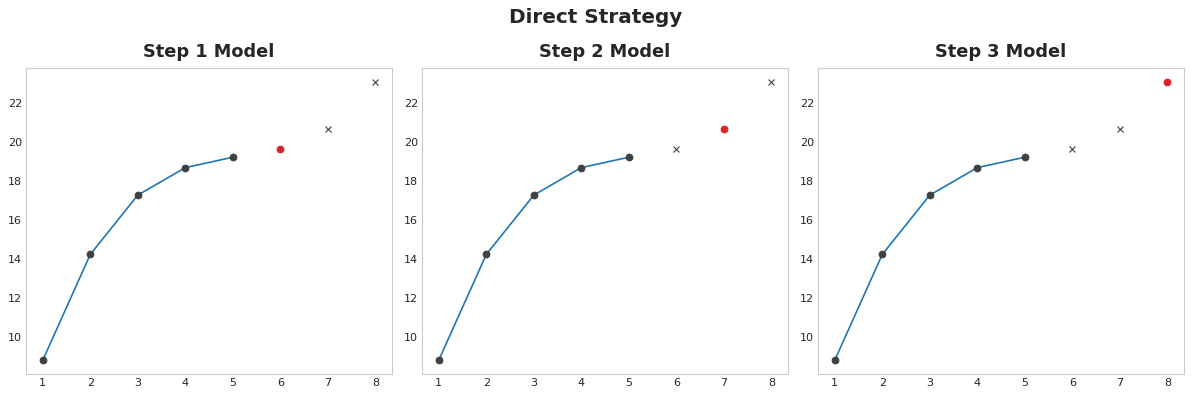
### **Multioutput model**[**¶**](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Multioutput-model)

Doğal olarak çoklu çıktı üreten bir model kullanın. Hem doğrusal regresyon hem de sinir ağları çoklu çıktı üretebilir. Bu strateji basit ve etkilidir, ancak kullanmak isteyeceğiniz her algoritma için mümkün değildir. Örneğin, **XGBoost** bunu yapamaz.



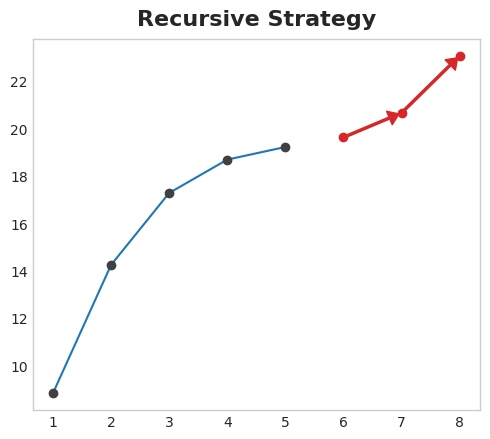
### **Direct strategy**[**¶**](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Direct-strategy)

Ufuktaki her adım için ayrı bir model eğitin: bir model 1 adım ilerisini, bir diğeri 2 adım ilerisini ve bu şekilde devam ederek her adım için farklı bir model tahmin yapar. 1 adım ilerisini tahmin etmek, 2 adım ilerisini tahmin etmekten farklı bir problem olduğundan, her adım için farklı bir modelin tahmin yapması faydalı olabilir. Ancak, bu yaklaşımın dezavantajı, çok sayıda model eğitmenin işlem gücü açısından maliyetli olabilmesidir.



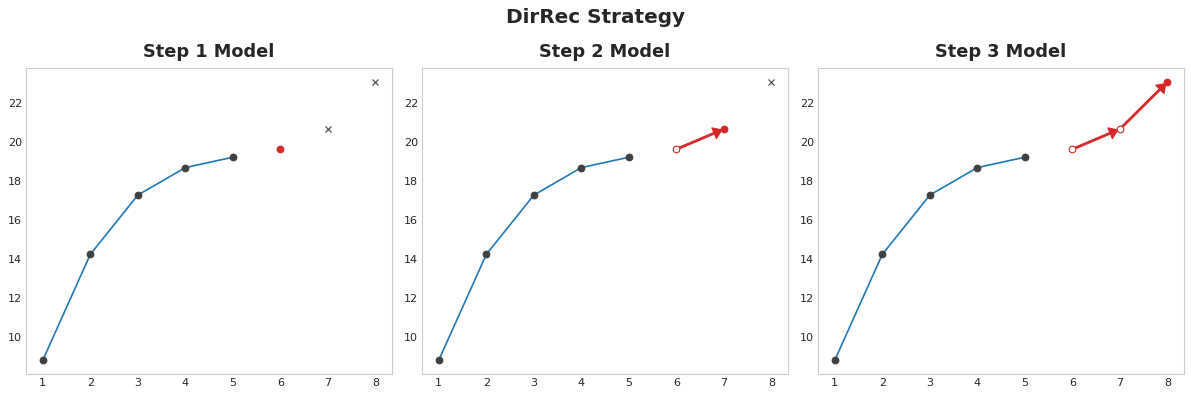
### **Recursive strategy**[**¶**](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Recursive-strategy)

Tek bir bir-adımlı model eğitin ve tahminlerini bir sonraki adım için gecikme özelliklerini (lag features) güncellemekte kullanın. **Özyinelemeli (recursive) yöntemle**, bir modelin bir-adımlı tahminini, bir sonraki tahmin adımı için gecikme özelliği olarak kullanmak üzere aynı modele geri besleriz. Sadece bir model eğitmemiz gerekir, ancak hatalar adımdan adıma yayılabileceği için, uzun ufuklar için tahminler yanlış olabilir.



### **DirRec strategy**[**¶**](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#DirRec-strategy)

Hem doğrudan (direct) hem de özyinelemeli (recursive) stratejilerin birleşimi: her adım için bir model eğitilir ve önceki adımların tahminleri **yeni gecikme özellikleri (new lag features)** olarak kullanılır. Adım adım, her model ek bir gecikme girdisi alır. Her modelin her zaman güncel bir gecikme özellikleri setine sahip olması sayesinde, **DirRec stratejisi**, serisel bağımlılığı doğrudan yöntemden daha iyi yakalayabilir, ancak özyinelemeli yöntem gibi hata yayılımından da etkilenebilir.



# Example - Flu Trends[¶](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Example---Flu-Trends)

Bu örnekte, **Çoklu Çıktı (MultiOutput)** ve **Doğrudan (Direct)** stratejilerini, Ders 4'teki **Grip Eğilimleri (Flu Trends)** verilerine uygulayacağız. Bu sefer, eğitim döneminin ötesine geçerek birden fazla hafta için gerçek tahminler yapacağız.

Tahmin görevimizi, **bir haftalık öncül zaman (lead time)** ile **sekiz haftalık bir ufuk (horizon)** olarak tanımlayacağız. Başka bir deyişle, bir sonraki haftadan başlayarak sekiz haftalık grip vakalarını tahmin edeceğiz.

Gizli hücre, bu örneği kurar ve bir yardımcı fonksiyon olan **plot\_multistep**'i tanımlar.

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 sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

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

from xgboost import XGBRegressor

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 plot\_multistep(y, every=1, ax=None, palette\_kwargs=None):

palette\_kwargs\_ = dict(palette='husl', n\_colors=16, desat=None)

if palette\_kwargs **is** **not** None:

palette\_kwargs\_.update(palette\_kwargs)

palette = sns.color\_palette(\*\*palette\_kwargs\_)

if ax **is** None:

fig, ax = plt.subplots()

ax.set\_prop\_cycle(plt.cycler('color', palette))

for date, preds **in** y[::every].iterrows():

preds.index = pd.period\_range(start=date, periods=len(preds))

preds.plot(ax=ax)

return ax

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)

Öncelikle, çok adımlı tahmin için hedef serimizi (grip için haftalık muayene ziyaretleri) hazırlayacağız. Bu tamamlandığında, eğitim ve tahmin çok kolaylaşacak.

def make\_lags(ts, lags, lead\_time=1):

return pd.concat(

{

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

for i **in** range(lead\_time, lags + lead\_time)

},

axis=1)

*# Four weeks of lag features*

y = flu\_trends.FluVisits.copy()

X = make\_lags(y, lags=4).fillna(0.0)

def make\_multistep\_target(ts, steps):

return pd.concat(

{f'y\_step\_**{**i + 1**}**': ts.shift(-i)

for i **in** range(steps)},

axis=1)

*# Eight-week forecast*

y = make\_multistep\_target(y, steps=8).dropna()

*# Shifting has created indexes that don't match. Only keep times for*

*# which we have both targets and features.*

y, X = y.align(X, join='inner', axis=0)

### **Multioutput model**[**¶**](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Multioutput-model)

Çoklu Çıktı stratejisi olarak doğrusal regresyon kullanacağız. Verilerimizi birden fazla çıktı için hazırladıktan sonra, eğitim ve tahmin her zamanki gibi devam edecek.

*# Create splits*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, shuffle=False)

model = LinearRegression()

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

y\_fit = pd.DataFrame(model.predict(X\_train), index=X\_train.index, columns=y.columns)

y\_pred = pd.DataFrame(model.predict(X\_test), index=X\_test.index, columns=y.columns)

Çok adımlı bir modelin, girdi olarak kullanılan her örnek için eksiksiz bir tahmin üreteceğini unutmayın. Eğitim setinde 269 hafta, test setinde ise 90 hafta bulunuyor ve artık bu haftaların her biri için 8 adımlı bir tahminimiz var.

train\_rmse = mean\_squared\_error(y\_train, y\_fit, squared=False)

test\_rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

print((f"Train RMSE: **{**train\_rmse**:**.2f**}\n**" f"Test RMSE: **{**test\_rmse**:**.2f**}**"))

palette = dict(palette='husl', n\_colors=64)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(11, 6))

ax1 = flu\_trends.FluVisits[y\_fit.index].plot(\*\*plot\_params, ax=ax1)

ax1 = plot\_multistep(y\_fit, ax=ax1, palette\_kwargs=palette)

\_ = ax1.legend(['FluVisits (train)', 'Forecast'])

ax2 = flu\_trends.FluVisits[y\_pred.index].plot(\*\*plot\_params, ax=ax2)

ax2 = plot\_multistep(y\_pred, ax=ax2, palette\_kwargs=palette)

\_ = ax2.legend(['FluVisits (test)', 'Forecast'])

Train RMSE: 389.12

Test RMSE: 582.33

### **Direct strategy**[**¶**](https://www.kaggle.com/code/ryanholbrook/forecasting-with-machine-learning#Direct-strategy)

XGBoost, regresyon görevleri için birden fazla çıktı üretemez. Ancak, **Doğrudan (Direct) indirgeme stratejisini** uygulayarak, yine de çok adımlı tahminler yapmak için onu kullanabiliriz. Bu, XGBoost'u scikit-learn'ün **MultiOutputRegressor**'ı ile sarmalamak kadar kolaydır.

from sklearn.multioutput import MultiOutputRegressor

model = MultiOutputRegressor(XGBRegressor())

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

y\_fit = pd.DataFrame(model.predict(X\_train), index=X\_train.index, columns=y.columns)

y\_pred = pd.DataFrame(model.predict(X\_test), index=X\_test.index, columns=y.columns)

Burada XGBoost, eğitim verisine açıkça aşırı uyum sağlıyor (overfitting). Ancak test verisi üzerinde, grip sezonunun bazı dinamiklerini doğrusal regresyon modelinden daha iyi yakalayabildiği görülüyor. Muhtemelen, hiperparametre ayarı yapıldığında daha da iyi performans gösterecektir.

train\_rmse = mean\_squared\_error(y\_train, y\_fit, squared=False)

test\_rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

print((f"Train RMSE: **{**train\_rmse**:**.2f**}\n**" f"Test RMSE: **{**test\_rmse**:**.2f**}**"))

palette = dict(palette='husl', n\_colors=64)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(11, 6))

ax1 = flu\_trends.FluVisits[y\_fit.index].plot(\*\*plot\_params, ax=ax1)

ax1 = plot\_multistep(y\_fit, ax=ax1, palette\_kwargs=palette)

\_ = ax1.legend(['FluVisits (train)', 'Forecast'])

ax2 = flu\_trends.FluVisits[y\_pred.index].plot(\*\*plot\_params, ax=ax2)

ax2 = plot\_multistep(y\_pred, ax=ax2, palette\_kwargs=palette)

\_ = ax2.legend(['FluVisits (test)', 'Forecast'])

Train RMSE: 1.22

Test RMSE: 526.45

DirRec stratejisini kullanmak için, MultiOutputRegressor'ı başka bir scikit-learn sarmalayıcısı olan RegressorChain ile değiştirmeniz yeterli olacaktır. Özyinelemeli stratejiyi kendimiz kodlamamız gerekecektir.