Welcome to Time Series!

Tahminleme, belki de makine öğreniminin gerçek dünyadaki en yaygın uygulamasıdır. İşletmeler ürün talebini, hükümetler ekonomik ve nüfus artışını, meteorologlar ise havayı tahmin eder. Gelecekte olacakları anlamak, bilimde, yönetimde ve endüstride (kişisel hayatlarımızdan bahsetmiyorum bile!) acil bir ihtiyaçtır ve bu alanlardaki uzmanlar, bu ihtiyacı karşılamak için makine öğrenimini giderek daha fazla kullanmaktadır.

Zaman serisi tahminleme, uzun bir geçmişe sahip geniş bir alandır. Bu ders, en doğru tahminleri üretmek amacıyla modern makine öğrenimi yöntemlerinin zaman serisi verilerine uygulanmasına odaklanmaktadır. Bu dersteki dersler, geçmişteki Kaggle tahmin yarışmalarındaki kazanan çözümlerden ilham almıştır, ancak doğru tahminlerin öncelikli olduğu her durumda uygulanabilir olacaktır.

Bu dersi tamamladıktan sonra şunları yapabileceksiniz:

* Temel zaman serisi bileşenlerini (trendler, mevsimler ve döngüler) modellemek için öznitelik mühendisliği yapmak,
* Birçok farklı zaman serisi grafiğiyle zaman serisi verilerini görselleştirmek,
* Birbirini tamamlayan modellerin güçlü yönlerini birleştiren **hibrit** tahminleme modelleri oluşturmak, ve
* Makine öğrenimi yöntemlerini çeşitli tahminleme görevlerine uyarlamak.

Alıştırmaların bir parçası olarak, "Store Sales - Time Series Forecasting" (Mağaza Satışları - Zaman Serisi Tahminleme) başlangıç yarışmamıza katılma fırsatı bulacaksınız. Bu yarışmada, yaklaşık 1800 ürün kategorisindeki satışları, Ekvador merkezli büyük bir market zinciri olan Corporación Favorita için tahmin etmeniz isteniyor.

# What is a Time Series?[¶](https://www.kaggle.com/code/ryanholbrook/linear-regression-with-time-series#What-is-a-Time-Series?)

Tahminlemenin temel nesnesi, zaman içinde kaydedilen bir gözlem kümesi olan **zaman serisidir**. Tahminleme uygulamalarında, gözlemler genellikle günlük veya aylık gibi düzenli aralıklarla kaydedilir.

import pandas as pd

df = pd.read\_csv(

"../input/ts-course-data/book\_sales.csv",

index\_col='Date',

parse\_dates=['Date'],

).drop('Paperback', axis=1)

df.head()

| Hardcover |
| --- |
| Date |  |
| 2000-04-01 | 139 |
| 2000-04-02 | 128 |
| 2000-04-03 | 172 |
| 2000-04-04 | 139 |
| 2000-04-05 | 191 |

Bu seri, bir perakende mağazasında 30 gün boyunca satılan ciltli kitap sayısını kaydeder. **Hardcover** adlı tek bir gözlem sütunumuzun ve **Date** adlı bir zaman indeksimizin olduğuna dikkat edin.

# Linear Regression with Time Series[¶](https://www.kaggle.com/code/ryanholbrook/linear-regression-with-time-series#Linear-Regression-with-Time-Series)

Bu kursun ilk bölümü için, tahminleme modelleri oluşturmak amacıyla doğrusal regresyon algoritmasını kullanacağız. Doğrusal regresyon, pratikte yaygın olarak kullanılan bir yöntemdir ve karmaşık tahminleme görevlerine bile kolayca uyum sağlar. **Doğrusal regresyon** algoritması, girdi özelliklerinden ağırlıklı bir toplam oluşturmayı öğrenir. İki özellik için şöyle bir formülümüz olur:

target = weight\_1 \* feature\_1 + weight\_2 \* feature\_2 + bias

Eğitim sırasında, regresyon algoritması hedefi en iyi şekilde yansıtan ag˘​ırlık1​, ag˘​ırlık2​ ve **sapma (bias)** parametreleri için değerler öğrenir. (Bu algoritmaya genellikle **sıradan en küçük kareler** denir, çünkü hedef ile tahminler arasındaki karesel hatayı en aza indiren değerleri seçer.) Ağırlıklara aynı zamanda **regresyon katsayıları** denir ve **sapma (bias)**, bu fonksiyonun grafiğinin y-eksenini nerede kestiğini gösterdiği için **kesme noktası (intercept)** olarak da adlandırılır.

### **Time-step features[¶](https://www.kaggle.com/code/ryanholbrook/linear-regression-with-time-series" \l "Time-step-features" \t "_self)**

Zaman serisine özgü iki tür özellik vardır: zaman adımı özellikleri ve gecikme (lag) özellikleri.

Zaman adımı özellikleri, doğrudan zaman indeksinden türetebileceğimiz özelliklerdir. En temel zaman adımı özelliği, serideki zaman adımlarını baştan sona sayan **zaman kuklasıdır (time dummy)**.

import numpy as np

df['Time'] = np.arange(len(df.index))

df.head()

| Hardcover | Time |
| --- | --- |
| Date |  |  |
| 2000-04-01 | 139 | 0 |
| 2000-04-02 | 128 | 1 |
| 2000-04-03 | 172 | 2 |
| 2000-04-04 | 139 | 3 |
| 2000-04-05 | 191 | 4 |

Zaman kukla değişkeni ile doğrusal regresyon şu modeli üretir:

target = weight \* time + bias

Zaman kuklası, **Zaman**'ın x eksenini oluşturduğu bir **zaman grafiğinde (time plot)**, zaman serilerine eğriler uydurmamızı sağlar.

import matplotlib.pyplot as plt

import seaborn as sns

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

plt.rc(

"figure",

autolayout=True,

figsize=(11, 4),

titlesize=18,

titleweight='bold',

)

plt.rc(

"axes",

labelweight="bold",

labelsize="large",

titleweight="bold",

titlesize=16,

titlepad=10,

)

%config InlineBackend.figure\_format = 'retina'

fig, ax = plt.subplots()

ax.plot('Time', 'Hardcover', data=df, color='0.75')

ax = sns.regplot(x='Time', y='Hardcover', data=df, ci=None, scatter\_kws=dict(color='0.25'))

ax.set\_title('Time Plot of Hardcover Sales');

Zaman adımı özellikleri, **zaman bağımlılığını** modellemenizi sağlar. Bir seri, değerleri gerçekleştiği zamana göre tahmin edilebiliyorsa zaman bağımlıdır. **Ciltli Kitap Satışları** serisinde, ayın sonlarındaki satışların genellikle ayın başlarındakilerden daha yüksek olduğunu tahmin edebiliriz.

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

Bir **gecikme (lag) özelliği** oluşturmak için, hedef serinin gözlemlerini, zaman içinde daha sonra gerçekleşmiş gibi görünecek şekilde kaydırırız. Burada 1 adımlı bir gecikme özelliği oluşturduk, ancak birden fazla adımla kaydırma yapmak da mümkündür.

df['Lag\_1'] = df['Hardcover'].shift(1)

df = df.reindex(columns=['Hardcover', 'Lag\_1'])

df.head()

| Hardcover | Lag\_1 |
| --- | --- |
| Date |  |  |
| 2000-04-01 | 139 | NaN |
| 2000-04-02 | 128 | 139.0 |
| 2000-04-03 | 172 | 128.0 |
| 2000-04-04 | 139 | 172.0 |
| 2000-04-05 | 191 | 139.0 |

Gecikmeli doğrusal regresyon şu modeli üretir:

target = weight \* lag + bias

Bu nedenle gecikme özellikleri, bir serideki her gözlemin bir önceki gözleme göre çizildiği gecikme grafiklerine eğriler yerleştirmemize olanak tanır.

fig, ax = plt.subplots()

ax = sns.regplot(x='Lag\_1', y='Hardcover', data=df, ci=None, scatter\_kws=dict(color='0.25'))

ax.set\_aspect('equal')

ax.set\_title('Lag Plot of Hardcover Sales');

Gecikme (lag) grafiğinden de görebileceğiniz gibi, bir gündeki satışlar (Hardcover), bir önceki günün satışlarıyla (Lag\_1) ilişkilidir. Böyle bir ilişki gördüğünüzde, bir gecikme özelliğinin faydalı olacağını anlarsınız.

Daha genel olarak, gecikme özellikleri **ardışık bağımlılığı** modellemenizi sağlar. Bir zaman serisi, bir gözlemin önceki gözlemlerden tahmin edilebildiği durumlarda ardışık bağımlılığa sahiptir. **Ciltli Kitap Satışları**'nda, bir gün yüksek satışların genellikle bir sonraki gün de yüksek satışlar anlamına geleceğini tahmin edebiliriz.

Makine öğrenimi algoritmalarını zaman serisi problemlerine uyarlamak, büyük ölçüde zaman indeksi ve gecikmelerle **öznitelik mühendisliği** yapmakla ilgilidir. Kursun çoğu bölümünde, basitliğinden dolayı doğrusal regresyonu kullanıyoruz, ancak bu özellikler tahmin göreviniz için hangi algoritmayı seçerseniz seçin faydalı olacaktır.

# Example - Tunnel Traffic[¶](https://www.kaggle.com/code/ryanholbrook/linear-regression-with-time-series" \l "Example---Tunnel-Traffic" \t "_self)

"Tunnel Traffic", Kasım 2003'ten Kasım 2005'e kadar İsviçre'deki Baregg Tüneli'nden her gün geçen araç sayısını gösteren bir zaman serisidir. Bu örnekte, doğrusal regresyonu zaman adımı özelliklerine ve gecikme özelliklerine uygulamak için biraz pratik yapacağız.

Gizli hücre her şeyi ayarlıyor.

from pathlib import Path

from warnings import simplefilter

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

simplefilter("ignore") *# ignore warnings to clean up output cells*

*# 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=14,

titlepad=10,

)

plot\_params = dict(

color="0.75",

style=".-",

markeredgecolor="0.25",

markerfacecolor="0.25",

legend=False,

)

%config InlineBackend.figure\_format = 'retina'

*# Load Tunnel Traffic dataset*

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

tunnel = pd.read\_csv(data\_dir / "tunnel.csv", parse\_dates=["Day"])

*# Create a time series in Pandas by setting the index to a date*

*# column. We parsed "Day" as a date type by using `parse\_dates` when*

*# loading the data.*

tunnel = tunnel.set\_index("Day")

*# By default, Pandas creates a `DatetimeIndex` with dtype `Timestamp`*

*# (equivalent to `np.datetime64`, representing a time series as a*

*# sequence of measurements taken at single moments. A `PeriodIndex`,*

*# on the other hand, represents a time series as a sequence of*

*# quantities accumulated over periods of time. Periods are often*

*# easier to work with, so that's what we'll use in this course.*

tunnel = tunnel.to\_period()

tunnel.head()

| NumVehicles |
| --- |
| Day |  |
| 2003-11-01 | 103536 |
| 2003-11-02 | 92051 |
| 2003-11-03 | 100795 |
| 2003-11-04 | 102352 |
| 2003-11-05 | 106569 |

### **Time-step feature[¶](https://www.kaggle.com/code/ryanholbrook/linear-regression-with-time-series" \l "Time-step-feature" \t "_self)**

Zaman serisinde eksik tarih yoksa, serinin uzunluğunu sayarak bir zaman kuklası oluşturabiliriz.

df = tunnel.copy()

df['Time'] = np.arange(len(tunnel.index))

df.head()

| NumVehicles | Time |
| --- | --- |
| Day |  |  |
| 2003-11-01 | 103536 | 0 |
| 2003-11-02 | 92051 | 1 |
| 2003-11-03 | 100795 | 2 |
| 2003-11-04 | 102352 | 3 |
| 2003-11-05 | 106569 | 4 |

Doğrusal regresyon modelinin uygulanması prosedürü scikit-learn'deki standart adımları takip eder.

from sklearn.linear\_model import LinearRegression

*# Training data*

X = df.loc[:, ['Time']] *# features*

y = df.loc[:, 'NumVehicles'] *# target*

*# Train the model*

model = LinearRegression()

model.fit(X, y)

*# Store the fitted values as a time series with the same time index as*

*# the training data*

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

Oluşturulan model aslında (yaklaşık olarak): Vehicles=22.5∗Time+98176. Uyumlu değerlerin zamanla birlikte grafiğini çizmek, doğrusal regresyonun zaman kuklasına nasıl uydurulduğunu gösterir ve bu denklemle tanımlanan trend çizgisini oluşturur.

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

ax = y\_pred.plot(ax=ax, linewidth=3)

ax.set\_title('Time Plot of Tunnel Traffic');

### **Lag feature[¶](https://www.kaggle.com/code/ryanholbrook/linear-regression-with-time-series" \l "Lag-feature" \t "_self)**

Pandas bize bir diziyi geciktirmek için basit bir yöntem sunuyor: kaydırma yöntemi.

df['Lag\_1'] = df['NumVehicles'].shift(1)

df.head()

| NumVehicles | Time | Lag\_1 |
| --- | --- | --- |
| Day |  |  |  |
| 2003-11-01 | 103536 | 0 | NaN |
| 2003-11-02 | 92051 | 1 | 103536.0 |
| 2003-11-03 | 100795 | 2 | 92051.0 |
| 2003-11-04 | 102352 | 3 | 100795.0 |
| 2003-11-05 | 106569 | 4 | 102352.0 |

Gecikme özellikleri oluştururken, ortaya çıkan eksik değerlerle ne yapacağımıza karar vermemiz gerekir. Bunları 0.0 ile doldurmak veya ilk bilinen değerle "geriye doğru doldurma" yapmak bir seçenektir. Bunun yerine, biz sadece eksik değerleri atacağız ve hedefteki değerleri de ilgili tarihlerden atacağımızdan emin olacağız.

from sklearn.linear\_model import LinearRegression

X = df.loc[:, ['Lag\_1']]

X.dropna(inplace=True) *# drop missing values in the feature set*

y = df.loc[:, 'NumVehicles'] *# create the target*

y, X = y.align(X, join='inner') *# drop corresponding values in target*

model = LinearRegression()

model.fit(X, y)

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

Gecikme grafiği bize bir günkü araç sayısı ile bir önceki günkü araç sayısı arasındaki ilişkiyi ne kadar iyi uydurabildiğimizi gösteriyor.

fig, ax = plt.subplots()

ax.plot(X['Lag\_1'], y, '.', color='0.25')

ax.plot(X['Lag\_1'], y\_pred)

ax.set\_aspect('equal')

ax.set\_ylabel('NumVehicles')

ax.set\_xlabel('Lag\_1')

ax.set\_title('Lag Plot of Tunnel Traffic');

Gecikmeli bir özellikten gelen bu tahmin, seriyi zaman içinde ne kadar iyi tahmin edebileceğimiz konusunda ne anlama geliyor? Aşağıdaki zaman çizelgesi, tahminlerimizin serinin yakın geçmişteki davranışına nasıl tepki verdiğini gösteriyor.

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

ax = y\_pred.plot()

En iyi zaman serisi modelleri genellikle zaman adımı özelliklerinin ve gecikme özelliklerinin bir kombinasyonunu içerir. Sonraki birkaç derste, bu dersteki özellikleri başlangıç noktası olarak kullanarak zaman serilerindeki en yaygın kalıpları modelleyen özellikler geliştirmeyi öğreneceğiz.