Makine Öğrenmesi ile Tahmin

Bu colab dosyasında bir zaman serisini tahmin etmek için bir makine öğrenmesi modeli geliştirelim. İlk olarak basit bir regresyon modeli oluşturacağız sonrasında ise iki katmanlı bir sinir ağı oluşturarak devam edeceğiz. Daha önceden kullandığımız işlevleri ve paketleri içeri aktararak başlayalım.

Gerekli Tanımlamalar ve Kurulumlar

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

keras = tf.keras
```

In [2]:

```
def plot_series(time, series, format="-", start=0, end=None, label=None):
    plt.plot(time[start:end], series[start:end], format, label=label)
    plt.xlabel("Time")
    plt.ylabel("Value")
    if label:
        plt.legend(fontsize=14)
    plt.grid(True)

def trend(time, slope=0):
    return slope * time


def seasonal_pattern(season_time):
    """Just an arbitrary pattern, you can change it if you wish"""
    return np.where(season_time < 0.4,
                    np.cos(season_time * 2 * np.pi),
                    1 / np.exp(3 * season_time))


def seasonality(time, period, amplitude=1, phase=0):
    """Repeats the same pattern at each period"""
    season_time = ((time + phase) % period) / period
    return amplitude * seasonal_pattern(season_time)


def white_noise(time, noise_level=1, seed=None):
    rnd = np.random.RandomState(seed)
    return rnd.randn(len(time)) * noise_level
```

In [3]:

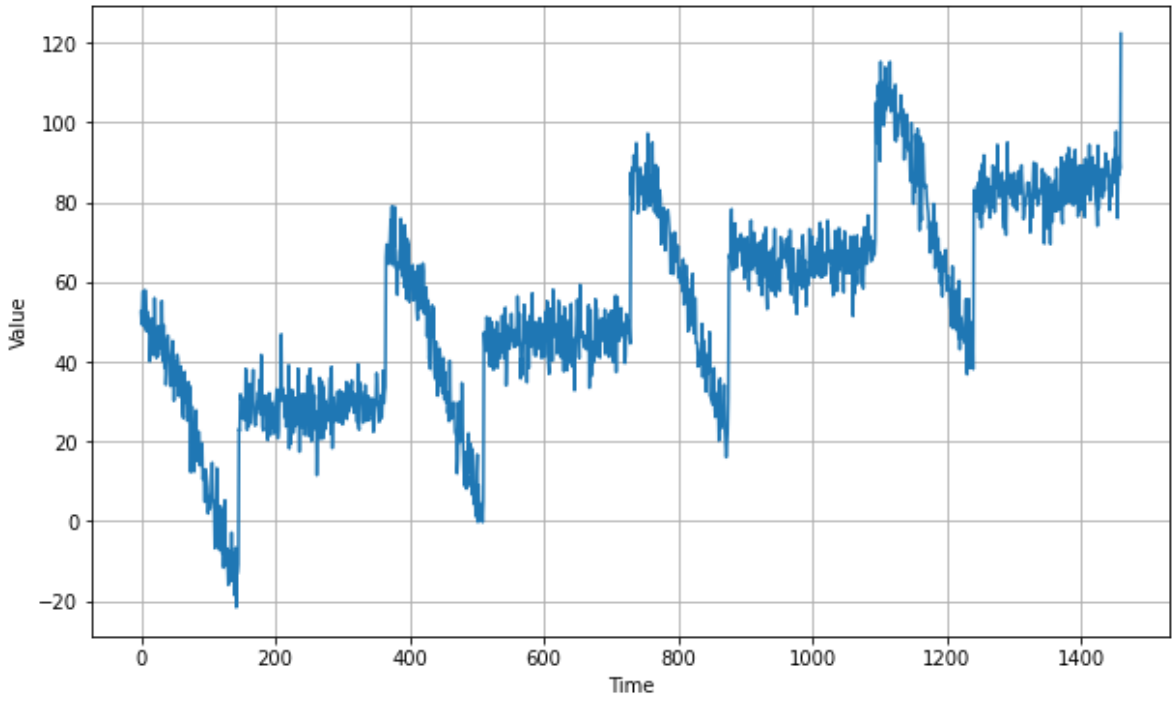
```
time = np.arange(4 * 365 + 1)

slope = 0.05
baseline = 10
amplitude = 40
series = baseline + trend(time, slope) + seasonality(time, period=365, amplitude=amplitude)

noise_level = 5
noise = white_noise(time, noise_level, seed=42)

series += noise

plt.figure(figsize=(10, 6))
plot_series(time, series)
plt.show()
```



Pencere Veri Seti Oluşturan İşlevin Tanımlanması

Önceki colav dosyasında oluşturduğumuz gibi bir `window_dataset` işlevi oluşturalım. Bu işlev serileri alır ve makine öğrenmesine uygun girdi setlerine dönüştürür.

İlk olarak, önceki 30 adımdaki verilere bakarak bir sonraki adımı tahmin etmek için bir model eğiteceğiz. Bu nedenle eğitim için 30 adımlık pencerelerden oluşan bir veri seti oluşturmamız gerekiyor.

In [4]:

```
def window_dataset(series, window_size, batch_size=32,
                   shuffle_buffer=1000):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.shuffle(shuffle_buffer)
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset
```

In [5]:

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]
```

In [ ]:

```

```

Doğrusal Model

İlk olarak bir doğrusal model oluşturalım.

- `keras.backend.clear_session()` : Kerasın arka uç oturumlarını temizler.
- `tf.random.set_seed(42)` ve `np.random.seed(42)` : Kodun her çalıştığında aynı çıktıyı vermesini sağlar = Tekrarlanabilirlik sağlar.

In [6]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

window_size = 30
train_set = window_dataset(x_train, window_size)
valid_set = window_dataset(x_valid, window_size)

model = keras.models.Sequential([
  keras.layers.Dense(1, input_shape=[window_size])
])
optimizer = keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
model.fit(train_set, epochs=100, validation_data=valid_set)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/
optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `lear
ning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")

Epoch 1/100
31/31 [==============================] - 3s 9ms/step - loss: 46.6813 - mae:
47.1784 - val_loss: 23.1475 - val_mae: 23.6473
Epoch 2/100
31/31 [==============================] - 0s 4ms/step - loss: 11.9123 - mae:
12.4017 - val_loss: 10.6108 - val_mae: 11.0934
Epoch 3/100
31/31 [==============================] - 0s 5ms/step - loss: 10.2063 - mae:
10.6947 - val_loss: 9.1115 - val_mae: 9.6013
Epoch 4/100
31/31 [==============================] - 0s 5ms/step - loss: 9.8053 - mae: 1
0.2898 - val_loss: 9.0103 - val_mae: 9.4965
Epoch 5/100
31/31 [==============================] - 0s 4ms/step - loss: 9.7068 - mae: 1
0.1940 - val_loss: 8.9696 - val_mae: 9.4569
Epoch 6/100
31/31 [==============================] - 0s 5ms/step - loss: 9.5858 - mae: 1
0.0706 - val_loss: 9.2814 - val_mae: 9.7695
Epoch 7/100
31/31 [==============================] - 0s 5ms/step - loss: 9.5716 - mae: 1
0.0561 - val_loss: 8.7506 - val_mae: 9.2409
Epoch 8/100
31/31 [==============================] - 0s 4ms/step - loss: 9.4335 - mae:
9.9208 - val_loss: 8.9409 - val_mae: 9.4259
Epoch 9/100
31/31 [==============================] - 0s 5ms/step - loss: 9.2821 - mae:
9.7655 - val_loss: 8.7019 - val_mae: 9.1887
Epoch 10/100
31/31 [==============================] - 0s 5ms/step - loss: 9.1588 - mae:
9.6455 - val_loss: 9.1858 - val_mae: 9.6757
Epoch 11/100
31/31 [==============================] - 0s 5ms/step - loss: 9.1080 - mae:
9.5909 - val_loss: 8.3841 - val_mae: 8.8717
Epoch 12/100
31/31 [==============================] - 0s 5ms/step - loss: 9.0054 - mae:
9.4901 - val_loss: 8.3203 - val_mae: 8.8105
Epoch 13/100
```

```
31/31 [==============================] - 0s 5ms/step - loss: 8.8572 - mae:
9.3423 - val_loss: 8.2304 - val_mae: 8.7186
Epoch 14/100
31/31 [==============================] - 0s 5ms/step - loss: 8.7560 - mae:
9.2400 - val_loss: 8.2015 - val_mae: 8.6929
Epoch 15/100
31/31 [==============================] - 0s 5ms/step - loss: 8.7060 - mae:
9.1891 - val_loss: 8.0562 - val_mae: 8.5446
Epoch 16/100
31/31 [==============================] - 0s 5ms/step - loss: 8.6244 - mae:
9.1106 - val_loss: 7.9800 - val_mae: 8.4679
Epoch 17/100
31/31 [==============================] - 0s 5ms/step - loss: 8.4941 - mae:
8.9788 - val_loss: 7.8962 - val_mae: 8.3846
Epoch 18/100
31/31 [==============================] - 0s 5ms/step - loss: 8.3914 - mae:
8.8778 - val_loss: 7.8248 - val_mae: 8.3136
Epoch 19/100
31/31 [==============================] - 0s 5ms/step - loss: 8.3789 - mae:
8.8672 - val_loss: 7.9163 - val_mae: 8.4018
Epoch 20/100
31/31 [==============================] - 0s 5ms/step - loss: 8.2872 - mae:
8.7752 - val_loss: 7.9781 - val_mae: 8.4660
Epoch 21/100
31/31 [==============================] - 0s 5ms/step - loss: 8.1183 - mae:
8.6040 - val_loss: 7.6067 - val_mae: 8.0948
Epoch 22/100
31/31 [==============================] - 0s 5ms/step - loss: 8.1174 - mae:
8.6013 - val_loss: 7.6044 - val_mae: 8.0939
Epoch 23/100
31/31 [==============================] - 0s 5ms/step - loss: 7.9768 - mae:
8.4606 - val_loss: 7.5181 - val_mae: 8.0027
Epoch 24/100
31/31 [==============================] - 0s 5ms/step - loss: 7.9488 - mae:
8.4330 - val_loss: 7.4165 - val_mae: 7.9037
Epoch 25/100
31/31 [==============================] - 0s 5ms/step - loss: 7.8248 - mae:
8.3076 - val_loss: 7.4864 - val_mae: 7.9701
Epoch 26/100
31/31 [==============================] - 0s 5ms/step - loss: 7.7746 - mae:
8.2588 - val_loss: 7.3600 - val_mae: 7.8449
Epoch 27/100
31/31 [==============================] - 0s 5ms/step - loss: 7.6489 - mae:
8.1347 - val_loss: 7.4656 - val_mae: 7.9534
Epoch 28/100
31/31 [==============================] - 0s 5ms/step - loss: 7.5881 - mae:
8.0748 - val_loss: 7.2633 - val_mae: 7.7468
Epoch 29/100
31/31 [==============================] - 0s 5ms/step - loss: 7.5281 - mae:
8.0121 - val_loss: 7.2226 - val_mae: 7.7046
Epoch 30/100
31/31 [==============================] - 0s 5ms/step - loss: 7.4287 - mae:
7.9117 - val_loss: 7.4429 - val_mae: 7.9314
Epoch 31/100
31/31 [==============================] - 0s 5ms/step - loss: 7.4079 - mae:
7.8897 - val_loss: 7.0365 - val_mae: 7.5220
Epoch 32/100
31/31 [==============================] - 0s 5ms/step - loss: 7.2735 - mae:
7.7602 - val_loss: 6.9825 - val_mae: 7.4673
Epoch 33/100
31/31 [==============================] - 0s 6ms/step - loss: 7.2739 - mae:
```

```
7.7580 - val_loss: 6.9299 - val_mae: 7.4151
Epoch 34/100
31/31 [==============================] - 0s 5ms/step - loss: 7.1570 - mae:
7.6430 - val_loss: 7.1487 - val_mae: 7.6338
Epoch 35/100
31/31 [==============================] - 0s 5ms/step - loss: 7.1307 - mae:
7.6135 - val_loss: 6.8096 - val_mae: 7.2905
Epoch 36/100
31/31 [==============================] - 0s 5ms/step - loss: 7.0562 - mae:
7.5384 - val_loss: 6.8589 - val_mae: 7.3451
Epoch 37/100
31/31 [==============================] - 0s 5ms/step - loss: 6.9709 - mae:
7.4550 - val_loss: 6.6912 - val_mae: 7.1721
Epoch 38/100
31/31 [==============================] - 0s 5ms/step - loss: 6.9274 - mae:
7.4110 - val_loss: 6.7854 - val_mae: 7.2720
Epoch 39/100
31/31 [==============================] - 0s 5ms/step - loss: 6.8287 - mae:
7.3123 - val_loss: 7.0975 - val_mae: 7.5836
Epoch 40/100
31/31 [==============================] - 0s 5ms/step - loss: 6.7947 - mae:
7.2743 - val_loss: 6.6359 - val_mae: 7.1221
Epoch 41/100
31/31 [==============================] - 0s 8ms/step - loss: 6.7593 - mae:
7.2442 - val_loss: 6.6222 - val_mae: 7.1082
Epoch 42/100
31/31 [==============================] - 0s 5ms/step - loss: 6.6816 - mae:
7.1641 - val_loss: 6.4618 - val_mae: 6.9384
Epoch 43/100
31/31 [==============================] - 0s 5ms/step - loss: 6.5978 - mae:
7.0800 - val_loss: 6.6013 - val_mae: 7.0877
Epoch 44/100
31/31 [==============================] - 0s 5ms/step - loss: 6.6256 - mae:
7.1066 - val_loss: 6.4921 - val_mae: 6.9772
Epoch 45/100
31/31 [==============================] - 0s 5ms/step - loss: 6.5119 - mae:
6.9944 - val_loss: 6.4523 - val_mae: 6.9408
Epoch 46/100
31/31 [==============================] - 0s 6ms/step - loss: 6.5113 - mae:
6.9923 - val_loss: 6.3676 - val_mae: 6.8539
Epoch 47/100
31/31 [==============================] - 0s 5ms/step - loss: 6.4703 - mae:
6.9551 - val_loss: 6.4135 - val_mae: 6.9027
Epoch 48/100
31/31 [==============================] - 0s 5ms/step - loss: 6.3575 - mae:
6.8423 - val_loss: 6.2646 - val_mae: 6.7469
Epoch 49/100
31/31 [==============================] - 0s 5ms/step - loss: 6.3132 - mae:
6.7936 - val_loss: 6.5807 - val_mae: 7.0648
Epoch 50/100
31/31 [==============================] - 0s 5ms/step - loss: 6.3524 - mae:
6.8329 - val_loss: 6.1738 - val_mae: 6.6523
Epoch 51/100
31/31 [==============================] - 0s 5ms/step - loss: 6.2175 - mae:
6.7013 - val_loss: 6.5609 - val_mae: 7.0448
Epoch 52/100
31/31 [==============================] - 0s 5ms/step - loss: 6.2483 - mae:
6.7297 - val_loss: 6.3995 - val_mae: 6.8871
Epoch 53/100
31/31 [==============================] - 0s 8ms/step - loss: 6.2216 - mae:
6.7043 - val_loss: 6.0807 - val_mae: 6.5592
```

```
Epoch 54/100
31/31 [==============================] - 0s 5ms/step - loss: 6.0713 - mae:
6.5524 - val_loss: 6.0608 - val_mae: 6.5410
Epoch 55/100
31/31 [==============================] - 0s 5ms/step - loss: 6.0764 - mae:
6.5575 - val_loss: 6.0446 - val_mae: 6.5258
Epoch 56/100
31/31 [==============================] - 0s 5ms/step - loss: 6.0693 - mae:
6.5438 - val_loss: 5.9976 - val_mae: 6.4784
Epoch 57/100
31/31 [==============================] - 0s 5ms/step - loss: 5.9879 - mae:
6.4677 - val_loss: 6.5870 - val_mae: 7.0729
Epoch 58/100
31/31 [==============================] - 0s 5ms/step - loss: 5.9670 - mae:
6.4472 - val_loss: 6.2659 - val_mae: 6.7540
Epoch 59/100
31/31 [==============================] - 0s 5ms/step - loss: 5.9528 - mae:
6.4379 - val_loss: 5.9457 - val_mae: 6.4270
Epoch 60/100
31/31 [==============================] - 0s 5ms/step - loss: 5.8707 - mae:
6.3552 - val_loss: 5.9625 - val_mae: 6.4462
Epoch 61/100
31/31 [==============================] - 0s 5ms/step - loss: 5.8656 - mae:
6.3480 - val_loss: 6.0507 - val_mae: 6.5383
Epoch 62/100
31/31 [==============================] - 0s 5ms/step - loss: 5.8503 - mae:
6.3305 - val_loss: 5.9375 - val_mae: 6.4194
Epoch 63/100
31/31 [==============================] - 0s 5ms/step - loss: 5.7668 - mae:
6.2473 - val_loss: 5.8891 - val_mae: 6.3675
Epoch 64/100
31/31 [==============================] - 0s 5ms/step - loss: 5.7735 - mae:
6.2540 - val_loss: 5.7918 - val_mae: 6.2716
Epoch 65/100
31/31 [==============================] - 0s 5ms/step - loss: 5.7196 - mae:
6.1992 - val_loss: 5.7822 - val_mae: 6.2677
Epoch 66/100
31/31 [==============================] - 0s 5ms/step - loss: 5.7195 - mae:
6.2001 - val_loss: 5.8370 - val_mae: 6.3169
Epoch 67/100
31/31 [==============================] - 0s 5ms/step - loss: 5.7116 - mae:
6.1953 - val_loss: 5.8026 - val_mae: 6.2866
Epoch 68/100
31/31 [==============================] - 0s 5ms/step - loss: 5.6455 - mae:
6.1273 - val_loss: 5.8970 - val_mae: 6.3825
Epoch 69/100
31/31 [==============================] - 0s 5ms/step - loss: 5.5995 - mae:
6.0752 - val_loss: 6.1008 - val_mae: 6.5871
Epoch 70/100
31/31 [==============================] - 0s 6ms/step - loss: 5.6110 - mae:
6.0918 - val_loss: 5.9595 - val_mae: 6.4444
Epoch 71/100
31/31 [==============================] - 0s 5ms/step - loss: 5.5976 - mae:
6.0817 - val_loss: 5.6474 - val_mae: 6.1311
Epoch 72/100
31/31 [==============================] - 0s 5ms/step - loss: 5.5355 - mae:
6.0157 - val_loss: 5.6318 - val_mae: 6.1172
Epoch 73/100
31/31 [==============================] - 0s 5ms/step - loss: 5.5398 - mae:
6.0224 - val_loss: 5.6279 - val_mae: 6.1140
Epoch 74/100
```

```
31/31 [==============================] - 0s 5ms/step - loss: 5.5468 - mae:
6.0275 - val_loss: 5.5921 - val_mae: 6.0779
Epoch 75/100
31/31 [==============================] - 0s 5ms/step - loss: 5.4855 - mae:
5.9659 - val_loss: 6.0816 - val_mae: 6.5651
Epoch 76/100
31/31 [==============================] - 0s 5ms/step - loss: 5.4834 - mae:
5.9646 - val_loss: 5.6142 - val_mae: 6.0898
Epoch 77/100
31/31 [==============================] - 0s 5ms/step - loss: 5.4842 - mae:
5.9646 - val_loss: 5.5331 - val_mae: 6.0186
Epoch 78/100
31/31 [==============================] - 0s 5ms/step - loss: 5.4070 - mae:
5.8878 - val_loss: 5.5209 - val_mae: 6.0072
Epoch 79/100
31/31 [==============================] - 0s 5ms/step - loss: 5.3568 - mae:
5.8389 - val_loss: 5.5470 - val_mae: 6.0356
Epoch 80/100
31/31 [==============================] - 0s 5ms/step - loss: 5.3579 - mae:
5.8375 - val_loss: 5.5790 - val_mae: 6.0657
Epoch 81/100
31/31 [==============================] - 0s 5ms/step - loss: 5.3304 - mae:
5.8088 - val_loss: 5.7429 - val_mae: 6.2243
Epoch 82/100
31/31 [==============================] - 0s 5ms/step - loss: 5.2761 - mae:
5.7589 - val_loss: 5.7831 - val_mae: 6.2640
Epoch 83/100
31/31 [==============================] - 0s 6ms/step - loss: 5.3007 - mae:
5.7804 - val_loss: 5.4424 - val_mae: 5.9216
Epoch 84/100
31/31 [==============================] - 0s 5ms/step - loss: 5.2705 - mae:
5.7502 - val_loss: 5.6147 - val_mae: 6.0958
Epoch 85/100
31/31 [==============================] - 0s 5ms/step - loss: 5.2405 - mae:
5.7215 - val_loss: 5.4449 - val_mae: 5.9350
Epoch 86/100
31/31 [==============================] - 0s 5ms/step - loss: 5.2553 - mae:
5.7318 - val_loss: 5.3910 - val_mae: 5.8785
Epoch 87/100
31/31 [==============================] - 0s 5ms/step - loss: 5.1897 - mae:
5.6698 - val_loss: 5.3836 - val_mae: 5.8726
Epoch 88/100
31/31 [==============================] - 0s 5ms/step - loss: 5.1770 - mae:
5.6583 - val_loss: 5.5470 - val_mae: 6.0273
Epoch 89/100
31/31 [==============================] - 0s 6ms/step - loss: 5.1712 - mae:
5.6501 - val_loss: 5.3334 - val_mae: 5.8172
Epoch 90/100
31/31 [==============================] - 0s 5ms/step - loss: 5.1521 - mae:
5.6322 - val_loss: 5.3244 - val_mae: 5.8025
Epoch 91/100
31/31 [==============================] - 0s 5ms/step - loss: 5.1463 - mae:
5.6261 - val_loss: 5.4182 - val_mae: 5.9050
Epoch 92/100
31/31 [==============================] - 0s 5ms/step - loss: 5.1003 - mae:
5.5802 - val_loss: 5.3004 - val_mae: 5.7758
Epoch 93/100
31/31 [==============================] - 0s 5ms/step - loss: 5.0947 - mae:
5.5750 - val_loss: 5.4387 - val_mae: 5.9227
Epoch 94/100
31/31 [==============================] - 0s 6ms/step - loss: 5.0597 - mae:
```

```
5.5381 - val_loss: 5.3146 - val_mae: 5.8038
Epoch 95/100
31/31 [==============================] - 0s 5ms/step - loss: 5.0638 - mae:
5.5395 - val_loss: 5.4179 - val_mae: 5.9012
Epoch 96/100
31/31 [==============================] - 0s 5ms/step - loss: 5.0408 - mae:
5.5199 - val_loss: 5.2369 - val_mae: 5.7242
Epoch 97/100
31/31 [==============================] - 0s 5ms/step - loss: 5.0147 - mae:
5.4941 - val_loss: 5.3345 - val_mae: 5.8204
Epoch 98/100
31/31 [==============================] - 0s 5ms/step - loss: 5.0047 - mae:
5.4842 - val_loss: 5.2042 - val_mae: 5.6894
Epoch 99/100
31/31 [==============================] - 0s 5ms/step - loss: 5.0098 - mae:
5.4891 - val_loss: 5.3813 - val_mae: 5.8612
Epoch 100/100
31/31 [==============================] - 0s 5ms/step - loss: 5.0298 - mae:
5.5070 - val_loss: 5.7582 - val_mae: 6.2393
```

Out[6]:

```
<tensorflow.python.keras.callbacks.History at 0x7fac9115d450>
```

Şimdi oluşturduğumuz modele bir adet geri arama (`callbacks`) tanımlayalım ve kerasın `LearningRateScheduler` işlevini kullanalım. Bu işlev sayesinde, eğitim defalarca çalıştırılır ve en iyi sonucu veren öğrenme değeri (`lr_schedule`) bulunur.

In [7]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

window_size = 30
train_set = window_dataset(x_train, window_size)

model = keras.models.Sequential([
  keras.layers.Dense(1, input_shape=[window_size])
])

lr_schedule = keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-6 * 10**(epoch / 30))
optimizer = keras.optimizers.SGD(lr=1e-6, momentum=0.9)
model.compile(loss=keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v
2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `
learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

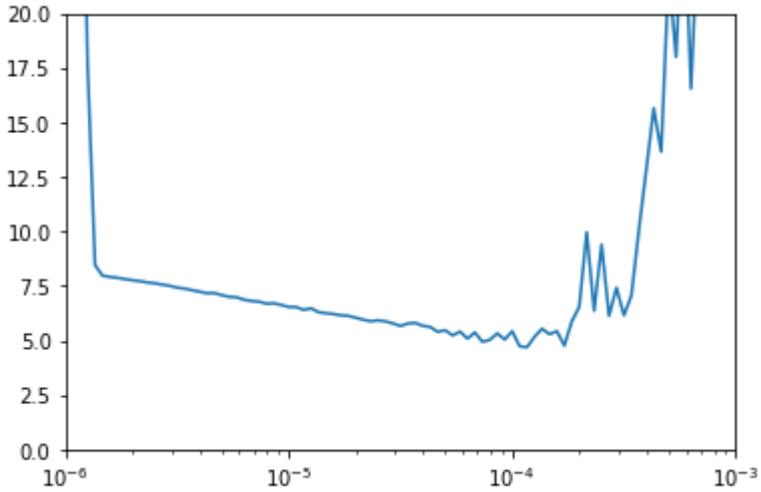
Öğrenme kaybımız (`loss`) başlangıçta hızlıca düşer sonrasında bir süre yavaş hızda düşüş yaşamaya devam eder. Bir noktadan sonra bir patlamak noktası ile tekar yüksek değerler almaya başlar. Grafikte daha kolay gözlemleyebiliriz:

In [8]:

```
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-6, 1e-3, 0, 20])
```

Out[8]:

```
(1e-06, 0.001, 0.0, 20.0)
```



Grafiği incelediğimizde en uygun değerin `1e-5` olacağını kabul edebiliriz. `1e-4` değerine ilerlerledikçe seçeceğimiz öğrenme puanı riskli olabilecektir. En uygun `lr` değerimizi bulduğumuza göre modelimizin optimize edici fonksiyonuna parametre olarak bunu verip modelimizi eğitebiliriz.

Modelimizi eğitirken erken durdurma `early_stopping` işlevi tanımlayabiliriz. Eğer modelimiz bir süre boyunca belirli bir ilerleme göstermiyorsa modelin aşırı uyuma geçmesine engel olmak için eğitimi durduruuz. Aşağıdaki kodda `patiance=10` argümanı 10 yinelemede (epochs) modelimiz öğrenme açısından ilerleme kaydetmiyorsa durmasını sağlayacaktır.

Şimdi modelimizi eğitebiliriz. Bulduğumuz `lr` değerini optimize edici fonksiyona verelim.

Bir doğrulama seti oluşturmamız gerektiğini unutmayalım: `valid_set` . Aynı zamanda `fit` içerisinde `callbacks` listesine tanımladığımız erken durdurma işlevini eklememiz gerekecektir.

Ve son olarak `epochs` değerini 500 olarak atıyoruz. Bu deüer oldukça büyük olabilir ancak belli bir epoch sayısından sonra model aşırı uyuma geçme riski oluşturacağı için erken durdurma işlevimiz modelin eğitimini epoch (yineleme) sayısına ulaşmadan bitirecektir.

In [9]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

window_size = 30
train_set = window_dataset(x_train, window_size)
valid_set = window_dataset(x_valid, window_size)

model = keras.models.Sequential([
  keras.layers.Dense(1, input_shape=[window_size])
])
optimizer = keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
early_stopping = keras.callbacks.EarlyStopping(patience=10)
model.fit(train_set, epochs=500,
          validation_data=valid_set,
          callbacks=[early_stopping])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v
2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `
learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Görüldüğü gibi erken durdurma işlevimiz 222. yinelemeden sonra modelimizin eğitimini durdurdu.

Şimdi tahminler yapmak için modelimizi kullanabiliriz. Bunun için zaman serisinin bir kısmını ve pencere boyutunu parametre olarak alan bir tahmin fonksiyonu (`model_forecast`) oluşturuyoruz.

In [10]:

```
def model_forecast(model, series, window_size):
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast
```

Şimdi rahatlıkla tahminlerde bulunabiliriz. Doğrulama verilerimizin bir kısmını parametre olarak vererek tahmnin

başarısını gözlemliyelim.

In [11]:

```
lin_forecast = model_forecast(model, series[split_time - window_size:-1], window_size)[:, 0
```

In [12]:

```
lin_forecast.shape
```

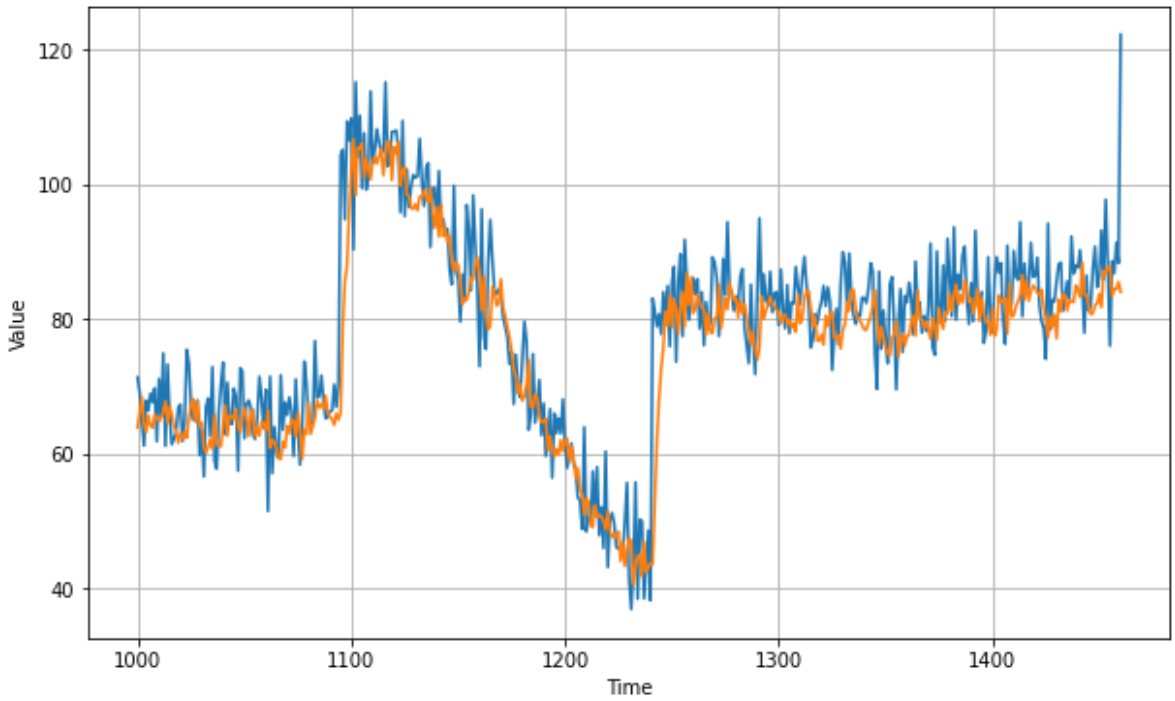
Out[12]:

```
(461,)
```

Şimdi tahminlerimizle gerçek değerlerimizi bir arada grafik üzerinde göserelim. Bunu yapmak için daha önceden tanımladığımız `plot_series` fonksiyonunu kullanabiliriz.

In [13]:

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, lin_forecast)
```



Tahminler gerçek değerlere çok iyi olmasa da eşleşiyor gibi görünüyor. Modelimizin performansını ölçelim ve ortalama mutlak hata (`mae`) değerimizi bulalım.

In [14]:

```
keras.metrics.mean_absolute_error(x_valid, lin_forecast).numpy()
```

Out[14]:

```
5.166268
```

Yoğun (Dense) Modeli ile Tahminlerde Bulunulması

Şimdi de iki katmanlı bir sinir ağı oluşturalım. Doğrusal modelde yaptığımız gibi oluşturduğumuz modele bir adet geri arama (`callbacks`) tanımlayalım ve kerasın `LearningRateScheduler` işlevini kullanalım. Bu işlev sayesinde, eğitim defalarca çalıştırılır ve en iyi sonucu veren öğrenme değeri (`lr_schedule`) bulunur.

In [15]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

window_size = 30
train_set = window_dataset(x_train, window_size)

model = keras.models.Sequential([
  keras.layers.Dense(10, activation="relu", input_shape=[window_size]),
  keras.layers.Dense(10, activation="relu"),
  keras.layers.Dense(1)
])

lr_schedule = keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-7 * 10**(epoch / 20))
optimizer = keras.optimizers.SGD(lr=1e-7, momentum=0.9)
model.compile(loss=keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v
2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `
learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

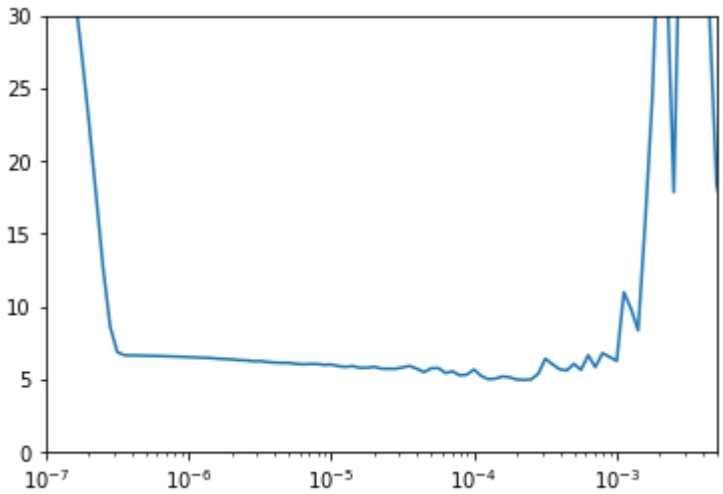
En iyi öğrenme değerinin ne olduğuna bakmak için grafiğini çizelim.

In [16]:

```
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-7, 5e-3, 0, 30])
```

Out[16]:

```
(1e-07, 0.005, 0.0, 30.0)
```



Grafiği incelediğimizde en uygun değerin `1e-5` olacağını kabul edebiliriz. `1e-4` değerine ilerlerledikçe seçeceğimiz öğrenme puanı riskli olabilecektir. En uygun `lr` değerimizi bulduğumuza göre modelimizin optimize edici fonksiyonuna parametre olarak bunu verip modelimizi eğitebiliriz.

Modelimizi eğitirken erken durdurma `early_stopping` işlevi tanımlayabiliriz. Eğer modelimiz bir süre boyunca belirli bir ilerleme göstermiyorsa modelin aşırı uyuma geçmesine engel olmak için eğitimi durduruuz. Aşağıdaki kodda `patiance=10` argümanı 10 yinelemede (epochs) modelimiz öğrenme açısından ilerleme kaydetmiyorsa durmasını sağlayacaktır.

Şimdi modelimizi eğitebiliriz. Bulduğumuz `lr` değerini optimize edici fonksiyona verelim.

Bir doğrulama seti oluşturmamız gerektiğini unutmayalım: `valid_set` . Aynı zamanda `fit` içerisinde `callbacks` listesine tanımladığımız erken durdurma işlevini eklememiz gerekecektir.

Ve son olarak `epochs` değerini 500 olarak atıyoruz. Bu deüer oldukça büyük olabilir ancak belli bir epoch sayısından sonra model aşırı uyuma geçme riski oluşturacağı için erken durdurma işlevimiz modelin eğitimini epoch (yineleme) sayısına ulaşmadan bitirecektir.

In [17]:

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

window_size = 30
train_set = window_dataset(x_train, window_size)
valid_set = window_dataset(x_valid, window_size)

model = keras.models.Sequential([
  keras.layers.Dense(10, activation="relu", input_shape=[window_size]),
  keras.layers.Dense(10, activation="relu"),
  keras.layers.Dense(1)
])

optimizer = keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
early_stopping = keras.callbacks.EarlyStopping(patience=10)
model.fit(train_set, epochs=500,
          validation_data=valid_set,
          callbacks=[early_stopping])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v
2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `
learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Görüldüğü gibi erken durdurma işlevimiz 172. yinelemeden sonra modelimizin eğitimini durdurdu.

Şimdi tahminler yapmak için modelimizi kullanabiliriz. Bunun için zaman serisinin bir kısmını ve pencere boyutunu parametre olarak alan bir tahmin fonksiyonu (`model_forecast`) oluşturmuştuk. `dense_forecast` adında bir değişken oluşturalım ve fonksiyonu çağıralım.

In [18]:

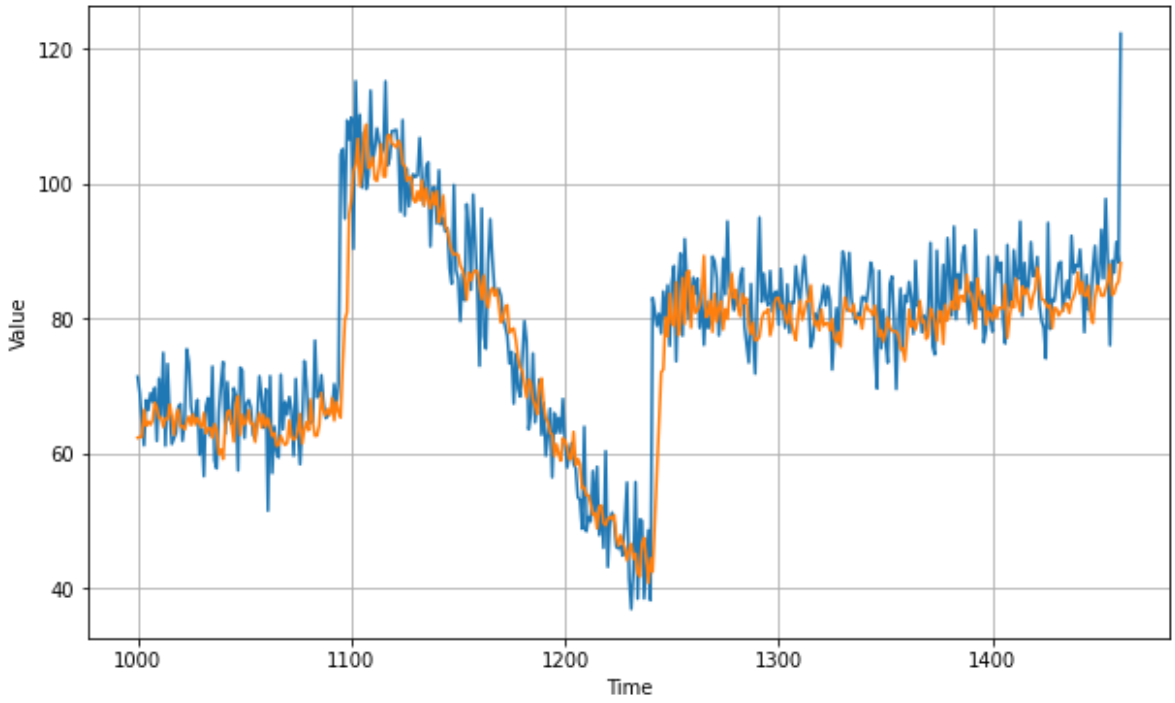
```
dense_forecast = model_forecast(
    model,
    series[split_time - window_size:-1],
    window_size)[:, 0]
```

Şimdi tahminlerimizle gerçek değerlerimizi bir arada grafik üzerinde göserelim. Bunu yapmak için daha önceden

tanımladığımız `plot_series` fonksiyonunu kullanabiliriz.

In [19]:

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, dense_forecast)
```



Tahminler gerçek değerlere çok iyi olmasa da eşleşiyor gibi görünüyor. Bakalım oluşturduğumuz doğrusal model mi yoksa sinir ağı mı daha iyi performan gösteriyor. Modelimizin performansını ölçelim ve ortalama mutlak hata (`mae`) değerimizi bulalım.

In [20]:

```
keras.metrics.mean_absolute_error(x_valid, dense_forecast).numpy()
```

Out[20]:

```
5.202555
```