In [1]:
```python
import os
import random
import numpy as np
from pylab import mpl, plt
plt.style.use('seaborn')
mpl.rcParams['savefig.dpi'] = 300
mpl.rcParams['font.family'] = 'serif'
os.environ['PYTHONHASHSEED'] = '0'
```

```
C:\Users\benwi\AppData\Local\Temp\ipykernel_31928\2447407336.py:5: MatplotlibDeprecationWarning:
The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond
to the styles shipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use('seaborn')
```

Generate an evenly spaced grid of floats for the x values between 0 and 10

In [2]:
```python
x = np.linspace(0, 10)
```

Then we want to fix the seed values for all relevant random number generators (so the output remains constant each time we run)

In [3]:
```python
def set_seeds(seed=100):
    random.seed(seed)
    np.random.seed(seed)
set_seeds()
```

In [4]:
```python
y = x + np.random.standard_normal(len(x))
```

Ordinary Least Squares of degree 1 (linear regression) on our data

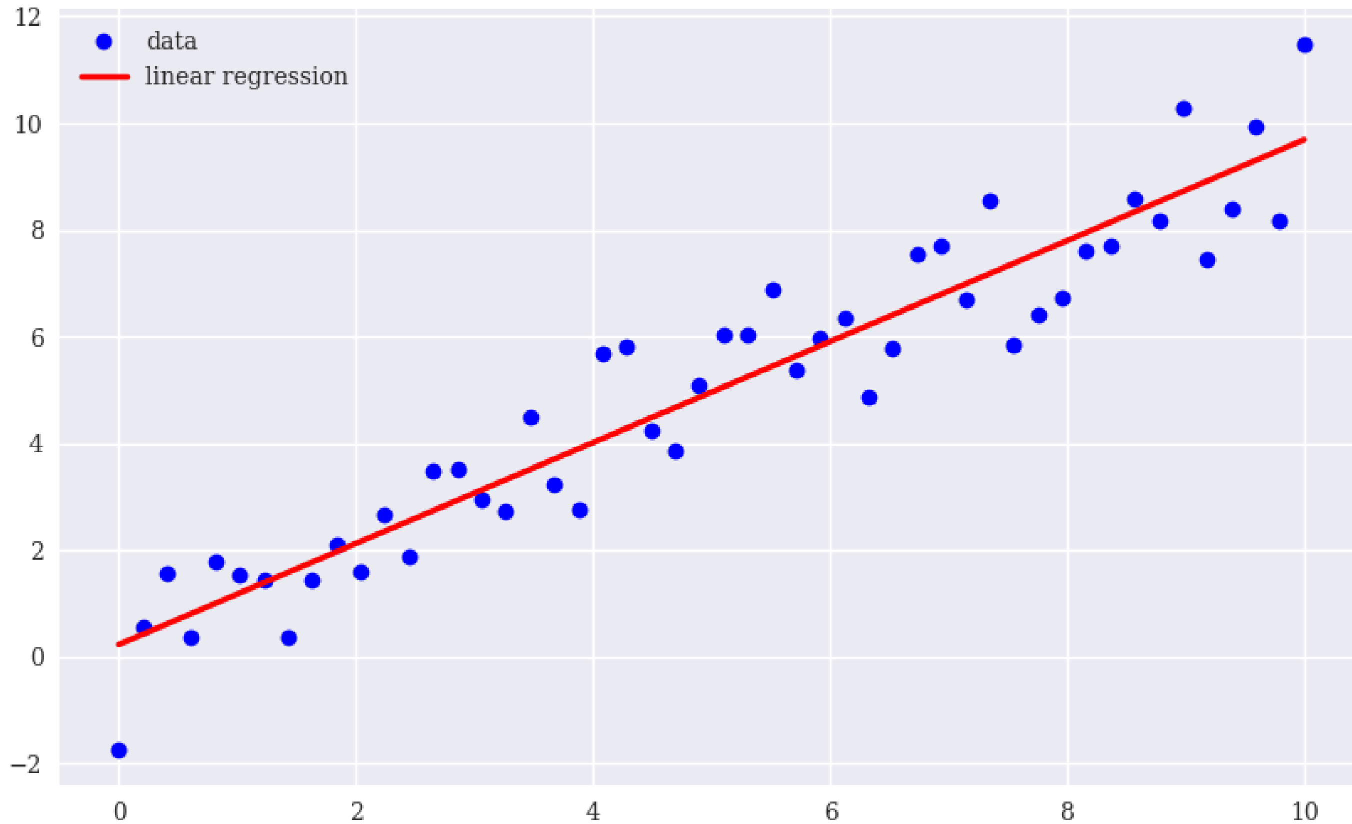
In [5]:
```python
reg = np.polyfit(x, y, deg=1)
```

In [6]:
```python
reg
```

Out[6]:
```
array([0.94612934, 0.22855261])
```

In [7]:
```python
plt.figure(figsize=(10,6))
plt.plot(x, y, 'bo', label='data')
plt.plot(x, np.polyval(reg, x), 'r', lw=2.5, label='linear regression')
plt.legend(loc=0)
```

Out[7]:
```
<matplotlib.legend.Legend at 0x14e0baab340>
```



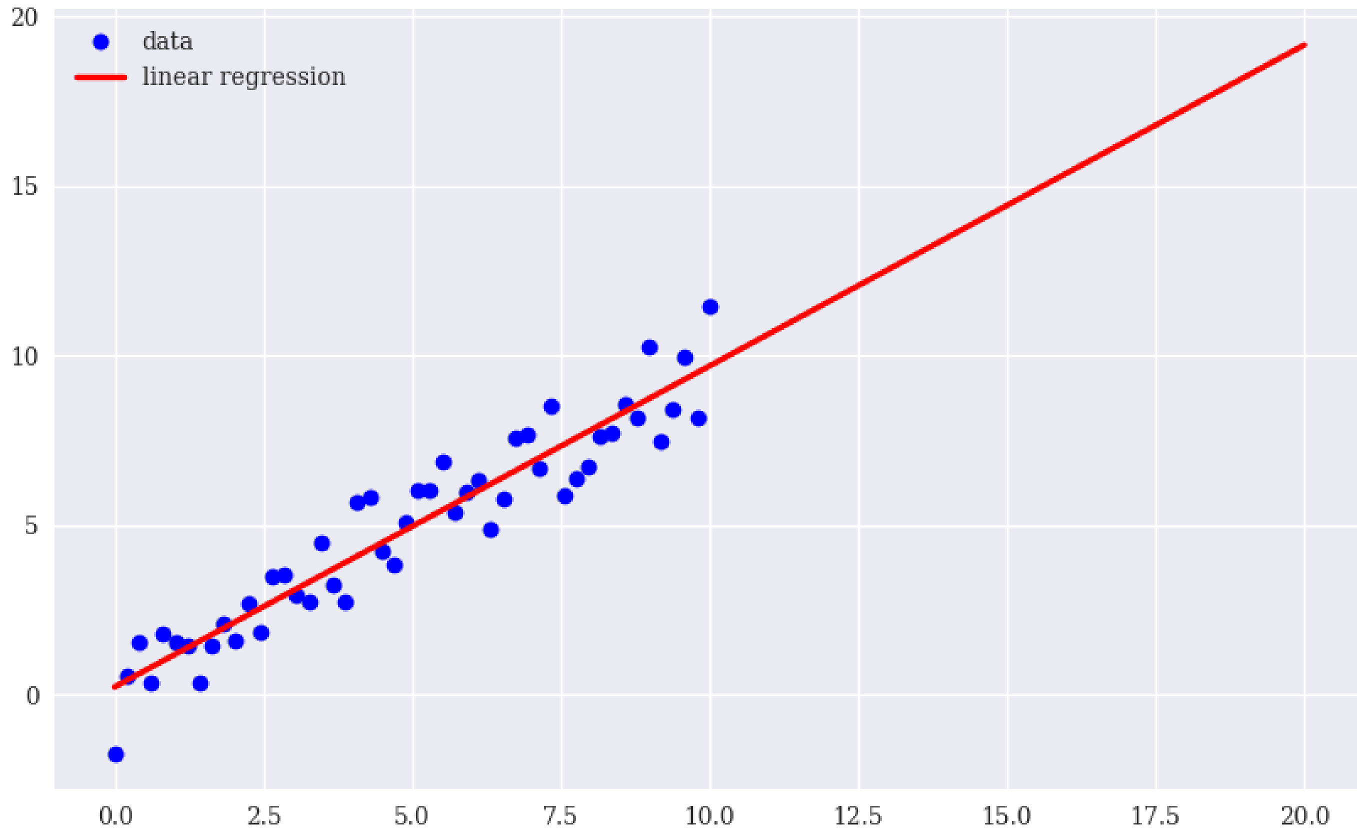
We can generate a larger domain for the x values, and extend our linear regression to 'predict' values for the dependent variable y beyond the domain of the original data set by extrapolation given the optimal regression parameters.

In [8]:

```
plt.figure(figsize=(10,6))
plt.plot(x, y, 'bo', label='data')
xn = np.linspace(0, 20)
plt.plot(xn, np.polyval(reg, xn), 'r', lw=2.5, label='linear regression')
plt.legend(loc=0)
```

Out[8]:

```
<matplotlib.legend.Legend at 0x14e0dbc18d0>
```



The Basic Idea for Price Prediction

In [9]:
```
x = np.arange(12)
x
```

Out[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

Assume three lags for the regression. This implies three independent variables for the regression and one dependent one. More concretely, 0, 1, and 2 are values of independent variables., while 3 would be the corresponding value for the dependent variable.

In [10]:
```
lags = 3
```

In [11]:
```
m = np.zeros((lags + 1, len(x) - lags))
```

In [12]:
```
m[lags] = x[lags:]
for i in range(lags):
    m[i] = x[i:i - lags]
```

In [13]:
```
m.T
```

Out[13]:
```
array([[ 0.,  1.,  2.,  3.],
       [ 1.,  2.,  3.,  4.],
       [ 2.,  3.,  4.,  5.],
       [ 3.,  4.,  5.,  6.],
       [ 4.,  5.,  6.,  7.],
       [ 5.,  6.,  7.,  8.],
       [ 6.,  7.,  8.,  9.],
       [ 7.,  8.,  9., 10.],
       [ 8.,  9., 10., 11.]])
```

Implement the linear Ordinary Least Squares regression

```
In [14]: reg = np.linalg.lstsq(m[:lags].T, m[lags], rcond=None)[0]
         reg
```

```
Out[14]: array([-0.66666667,  0.33333333,  1.33333333])
```

```
In [15]: np.dot(m[:lags].T, reg)
```

```
Out[15]: array([ 3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.])
```

We now want to translate this basic approach to time series data for a real financial instrument, like the EUR/USD exchange rate:

```
In [16]: import pandas as pd
```

```
In [19]: raw = pd.read_csv('http://hilpisch.com/pyalgo_eikon_eod_data.csv', index_col=0, parse_dates=True
         raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2516 entries, 2010-01-04 to 2019-12-31
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AAPL.O  2516 non-null   float64
 1   MSFT.O  2516 non-null   float64
 2   INTC.O  2516 non-null   float64
 3   AMZN.O  2516 non-null   float64
 4   GS.N    2516 non-null   float64
 5   SPY     2516 non-null   float64
 6   .SPX    2516 non-null   float64
 7   .VIX    2516 non-null   float64
 8   EUR=    2516 non-null   float64
 9   XAU=    2516 non-null   float64
 10  GDX     2516 non-null   float64
 11  GLD     2516 non-null   float64
dtypes: float64(12)
memory usage: 255.5 KB
```

```
In [20]: symbol = 'EUR='
```

```
In [21]: data = pd.DataFrame(raw[symbol])
```

```
In [22]: data.rename(columns={symbol: 'price'}, inplace=True)
```

```
In [23]: lags = 5
```

```
In [26]: cols = []
         for lag in range(1, lags + 1):
             col = f'lag_{lag}'
             data[col] =  data['price'].shift(lag)
             cols.append(col)
         data.dropna(inplace=True)
```

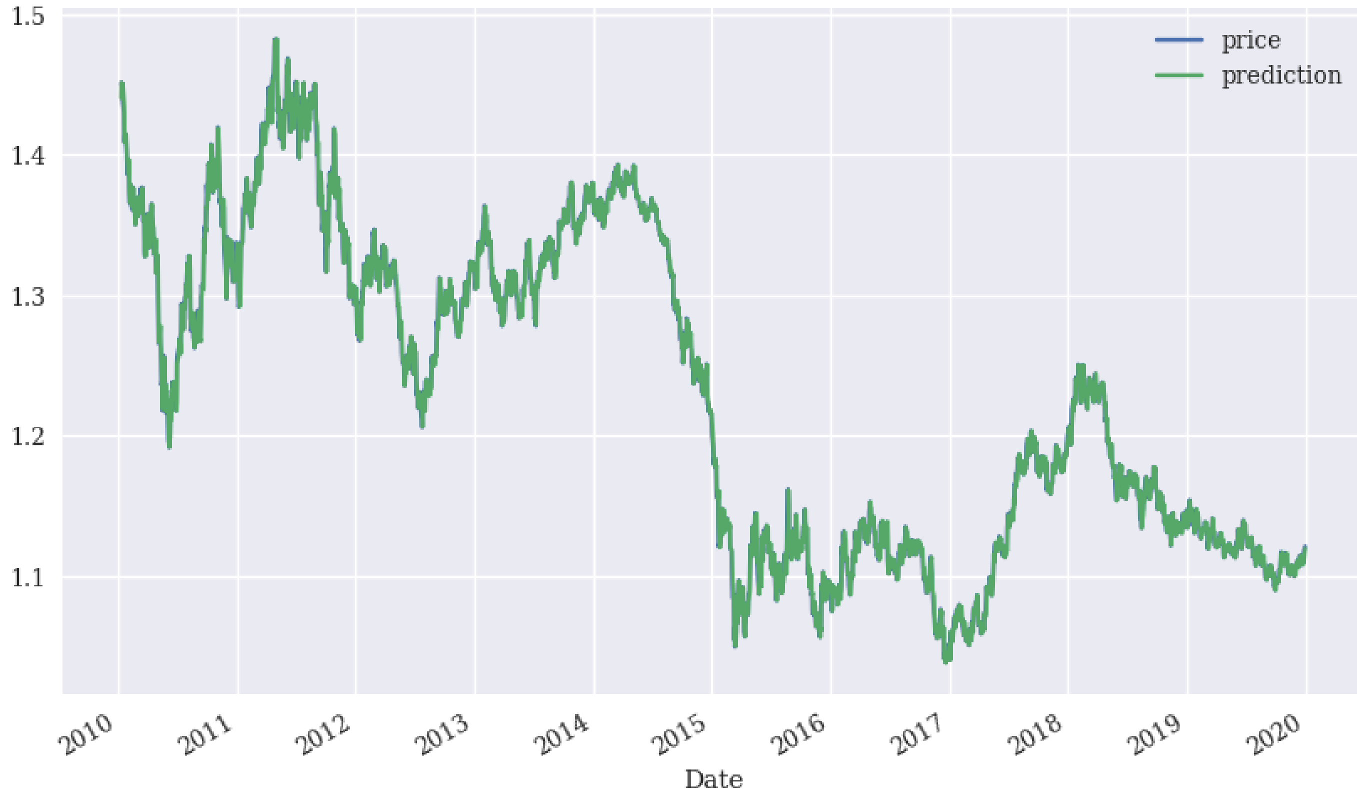
```
In [28]: reg = np.linalg.lstsq(data[cols], data['price'], rcond=None)[0]
         reg
```

```
Out[28]: array([ 0.98635864,  0.02292172, -0.04769849,  0.05037365, -0.01208135])
```

```
In [30]: data['prediction'] = np.dot(data[cols], reg)
```

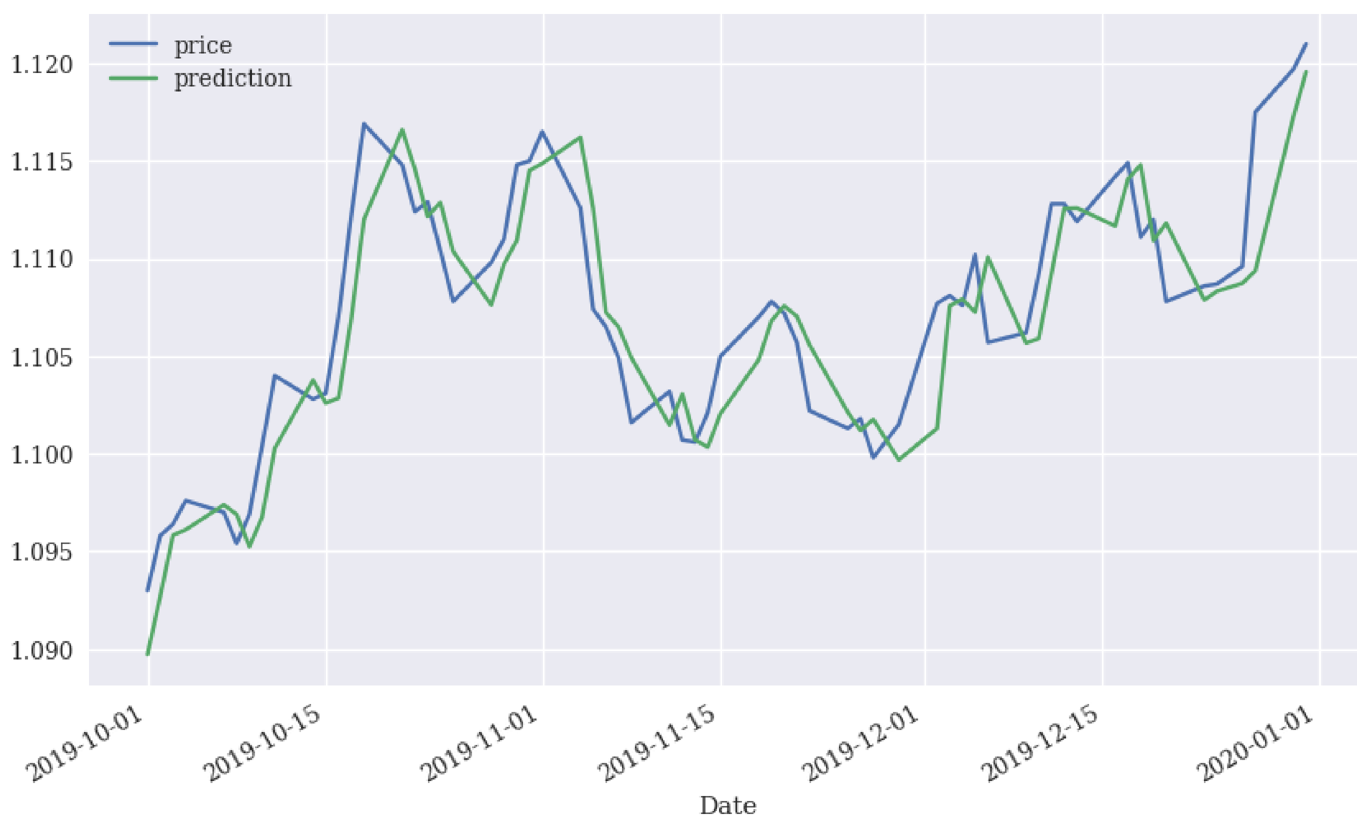
In [31]:
```
data[['price', 'prediction']].plot(figsize=(10, 6))
```

Out[31]: <AxesSubplot: xlabel='Date'>



In [32]:
```
data[['price', 'prediction']].loc['2019-10-1':].plot(figsize=(10,6))
```

Out[32]: <AxesSubplot: xlabel='Date'>



In [33]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2511 entries, 2010-01-11 to 2019-12-31
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   price       2511 non-null   float64
 1   lag_1       2511 non-null   float64
 2   lag_2       2511 non-null   float64
 3   lag_3       2511 non-null   float64
 4   lag_4       2511 non-null   float64
 5   lag_5       2511 non-null   float64
 6   prediction  2511 non-null   float64
dtypes: float64(7)
memory usage: 221.5 KB
```

In [ ]: