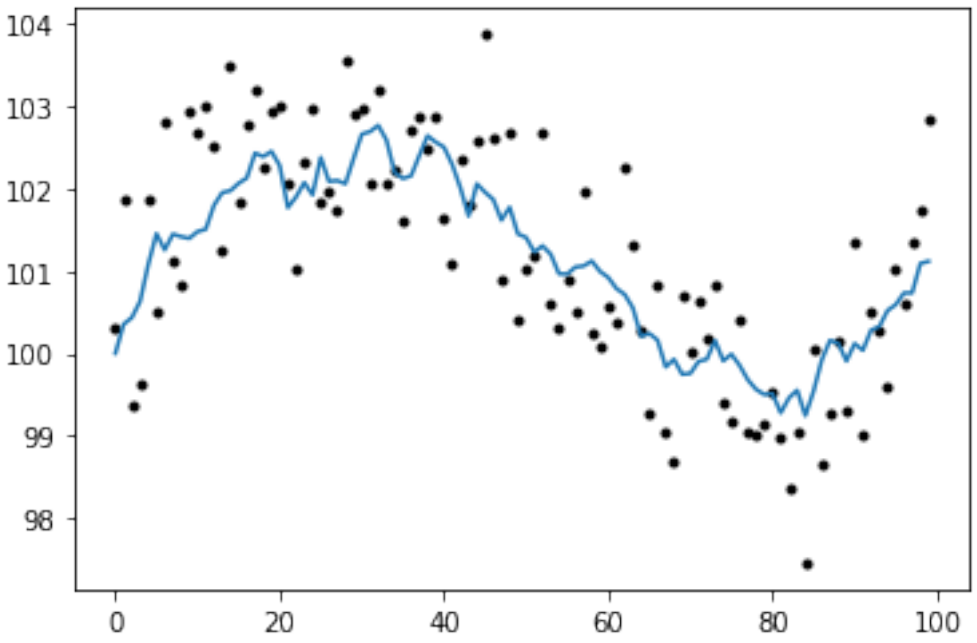
forecasting

April 14, 2022

[1]:
```
import numpy as np
from numpy.random import randn
from scipy.stats import norm     # normal cdf

import matplotlib.pyplot as plt
import pandas as pd

from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
import statsmodels.api as sm
from sklearn import datasets
```

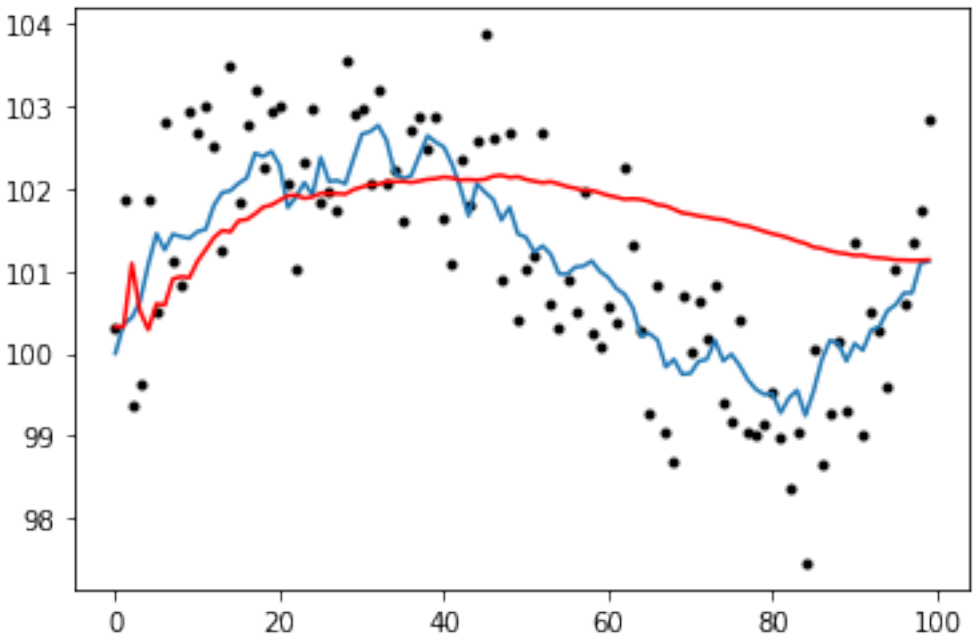
1 Used to generate Lecture 15 slides, section on "Simple Exponential Smoothing"

[2]:
```
# Generate synthetic data
np.random.seed(0)
n = 100
a  = np.zeros(n) # create an array of 0's of length n
s = .2           # controls how fast true mean wanders
a[0] = 100
for i in range(1,n):
    a[i] = a[i-1] + s*randn()
noise_std = .8
x = a + noise_std*randn(n)

plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
```



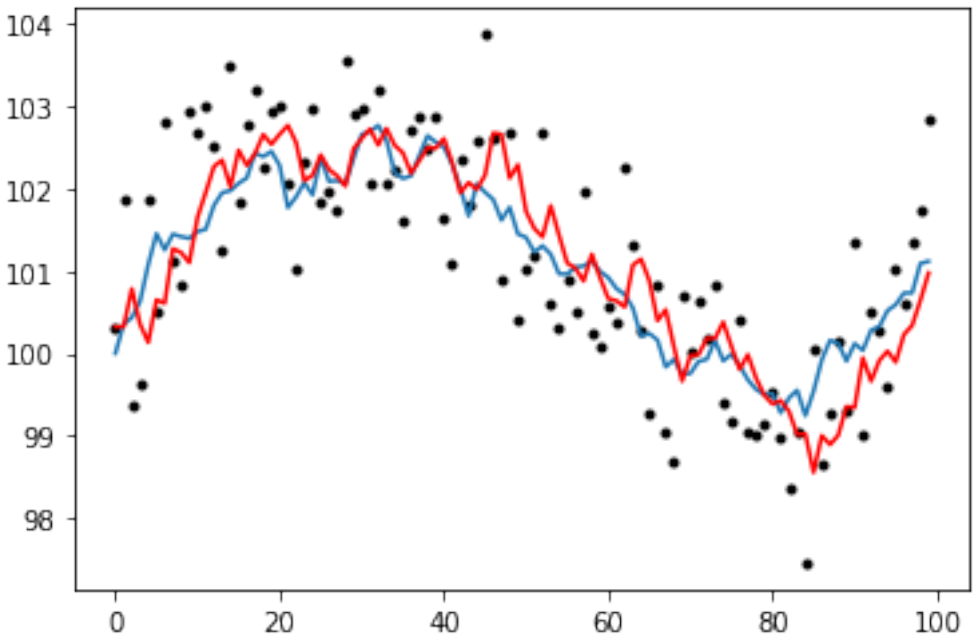
[3]:
```
# fit a constant mean model
ahat = np.zeros(len(x))
ahat[0] = x[0]
for i in range(1,n):
    ahat[i] = ahat[i-1] + 1/i*(x[i-1] - ahat[i-1])

plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), ahat, color='red')
```

[3]: [<matplotlib.lines.Line2D at 0x7fbfd860da50>]



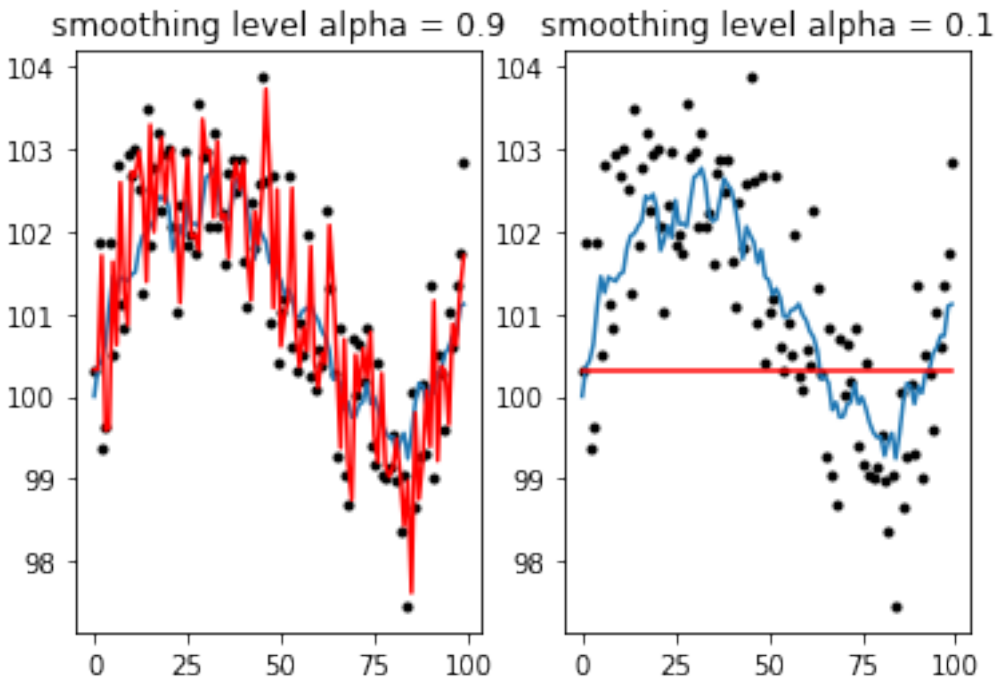
[4]:
```
fit = SimpleExpSmoothing(x).fit(smoothing_level=0.3, optimized=False) # choose␣
↪smoothing level explicitly
xhat = fit.fittedvalues
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), xhat, color='red')
```

[4]:
```
[<matplotlib.lines.Line2D at 0x7fbfca42e950>]
```



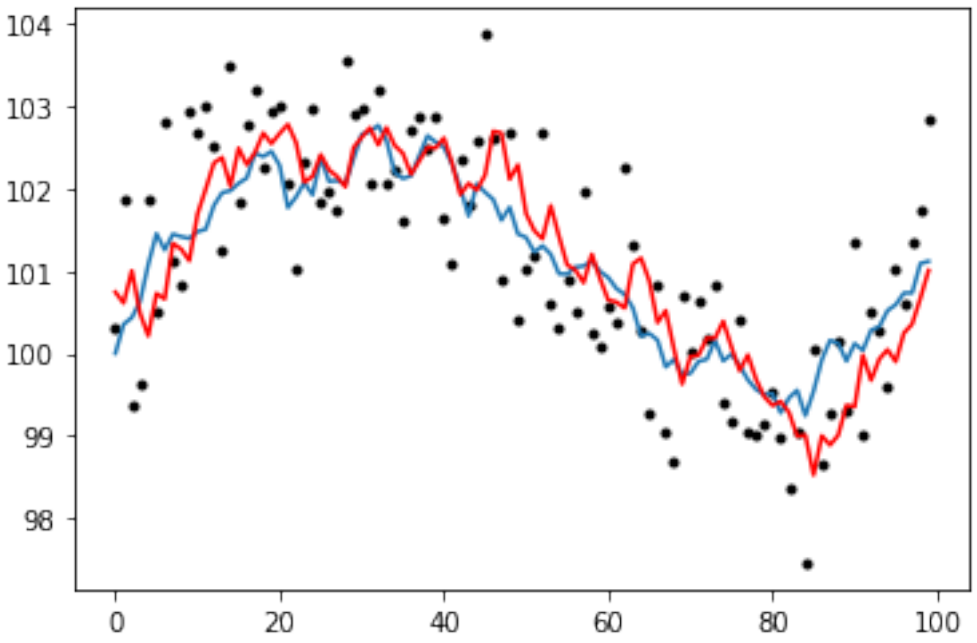
[5]:
```
plt.subplot(1,2,1)
fit = SimpleExpSmoothing(x).fit(smoothing_level=0.9, optimized=False) # choose␣
 ↪smoothing level explicitly
xhat = fit.fittedvalues
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), xhat, color='red')
plt.title('smoothing level alpha = 0.9')

plt.subplot(1,2,2)
fit = SimpleExpSmoothing(x).fit(smoothing_level=0., optimized=False) # choose␣
 ↪smoothing level explicitly
xhat = fit.fittedvalues
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), xhat, color='red')
plt.title('smoothing level alpha = 0.1')
```

[5]:
```
Text(0.5, 1.0, 'smoothing level alpha = 0.1')
```



[6]:
```
fit = SimpleExpSmoothing(x).fit() # choose smoothing level to best fit data
xhat = fit.fittedvalues
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), xhat, color='red')
```

[6]: [<matplotlib.lines.Line2D at 0x7fbfca4826d0>]



- Change noise std and s and see how it affects x.
- Change the alpha parameter to see how it affects the red line, and to make the red line track x.
- Remove the arguments to fit() to let SimpleExpSmoothing fit alpha for you.

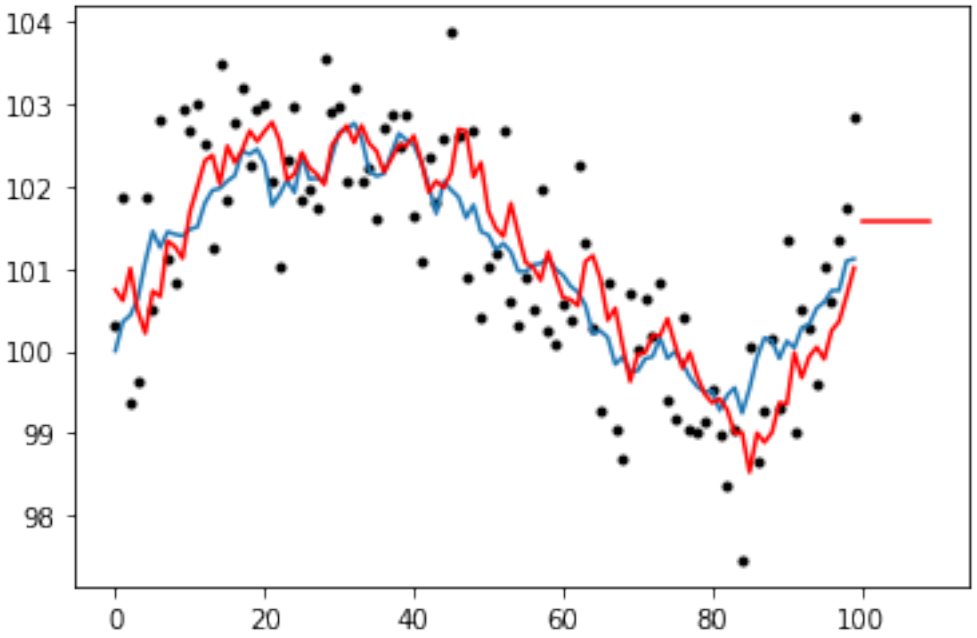
[7]:
```
fit = SimpleExpSmoothing(x).fit() # choose smoothing level to best fit data
fit.params # fit.params shows the chosen values for all parameters
```

[7]:
```
{'smoothing_level': 0.31305162196958597,
 'smoothing_slope': nan,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 100.7417392654899,
 'initial_slope': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[8]:
```
xhat = fit.fittedvalues
L = 10
forecast = fit.forecast(L)

# plot with forecast
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
```

```
plt.plot(range(n), xhat, color='red')
plt.plot(range(n,n+L), forecast, color='red')
```

[8]: [<matplotlib.lines.Line2D at 0x7fc009199610>]

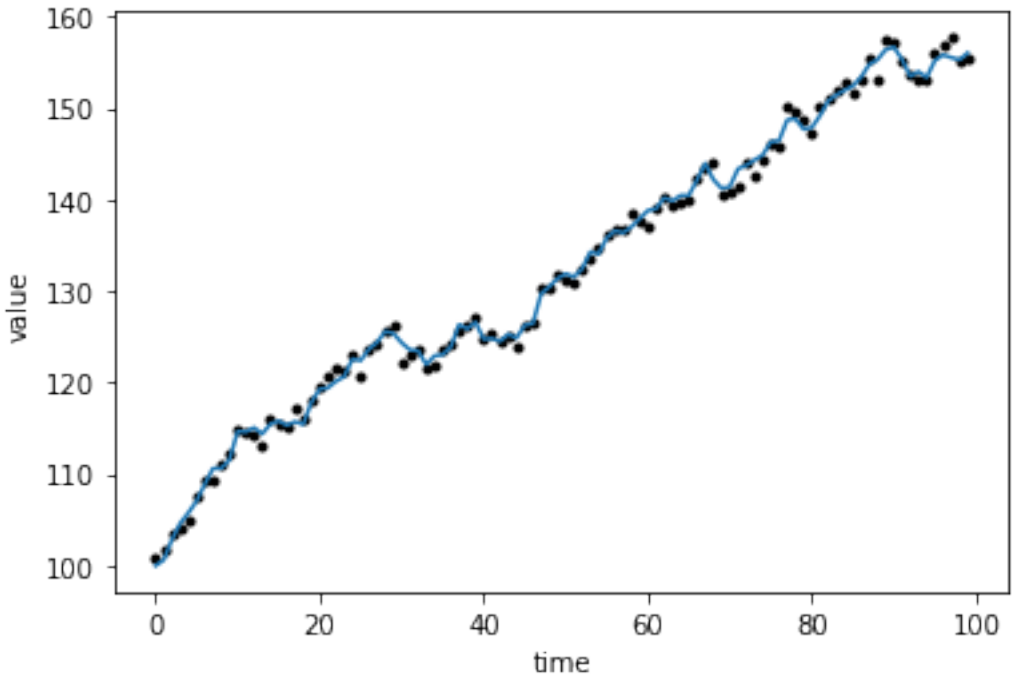


2 Used to generate Lecture 15 slides, section on Holt's nonseasonal model

[9]:
```
n = 100
a = np.zeros(n)
b = np.zeros(n)
a[0] = 100 # initial level
b[0] = 1 # initial trend

for i in range(1,n):
    b[i] = b[i-1] + 0.1*randn()       # trend wanders
    a[i] = a[i-1] + b[i-1] + randn() # level moves with the trend + extra␣
 ↪random movement

# We observe the level + extra noise
x = a + 0.8*randn(n)

# plot
plt.scatter(range(n), x, marker='.', color='black')
```

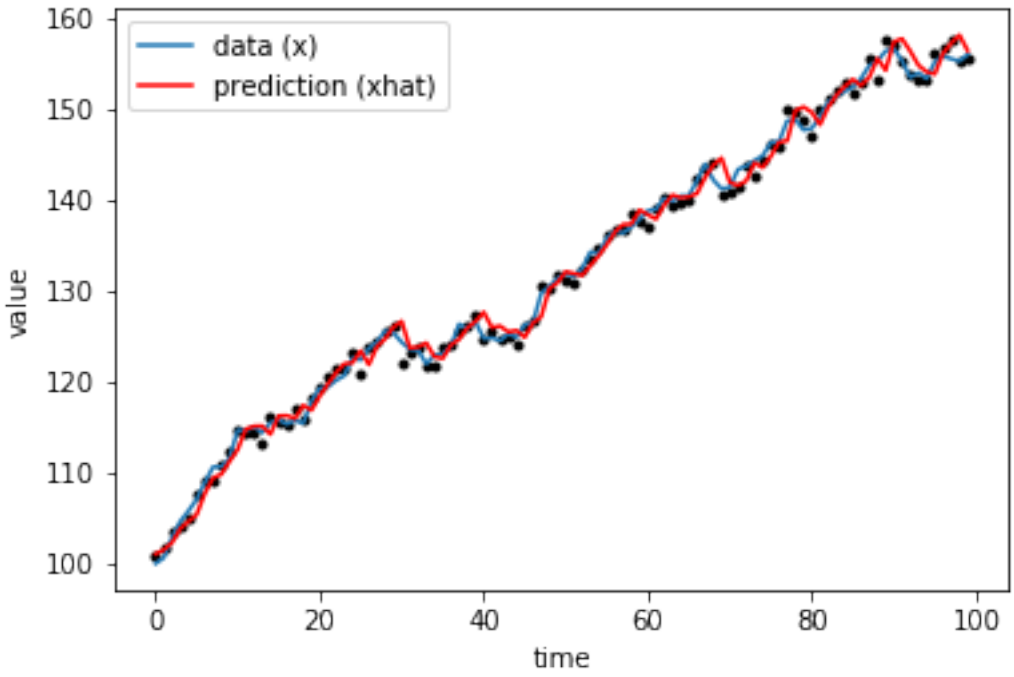
```
p = plt.plot(a)
plt.xlabel('time')
plt.ylabel('value')
```

[9]: Text(0, 0.5, 'value')



[10]:
```
# fit the model
# fit = Holt(x).fit(smoothing_level=0.1,smoothing_slope=.1,optimized=False)
fit = Holt(x).fit() # optimize over the parameters
xhat = fit.fittedvalues


# plot
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), xhat, color='red')
plt.legend(['data (x)','prediction (xhat)'])
plt.xlabel('time')
plt.ylabel('value')
```

[10]: Text(0, 0.5, 'value')



[11]:
```
fit.params
```

[11]:
```
{'smoothing_level': 0.7758755658994141,
 'smoothing_slope': 0.0,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 100.49865806955047,
 'initial_slope': 0.5522313840715876,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

Lec15, Question 3: Suppose I change the code to set initial value of trend b[0] = -1. What will happen? * A. The blue line will trend downward * B. The blue line will trend upward more quickly * C. The blue line will stay at 0 * D. The blue line will bounce around more * E. The black dots will be further away from the blue line.

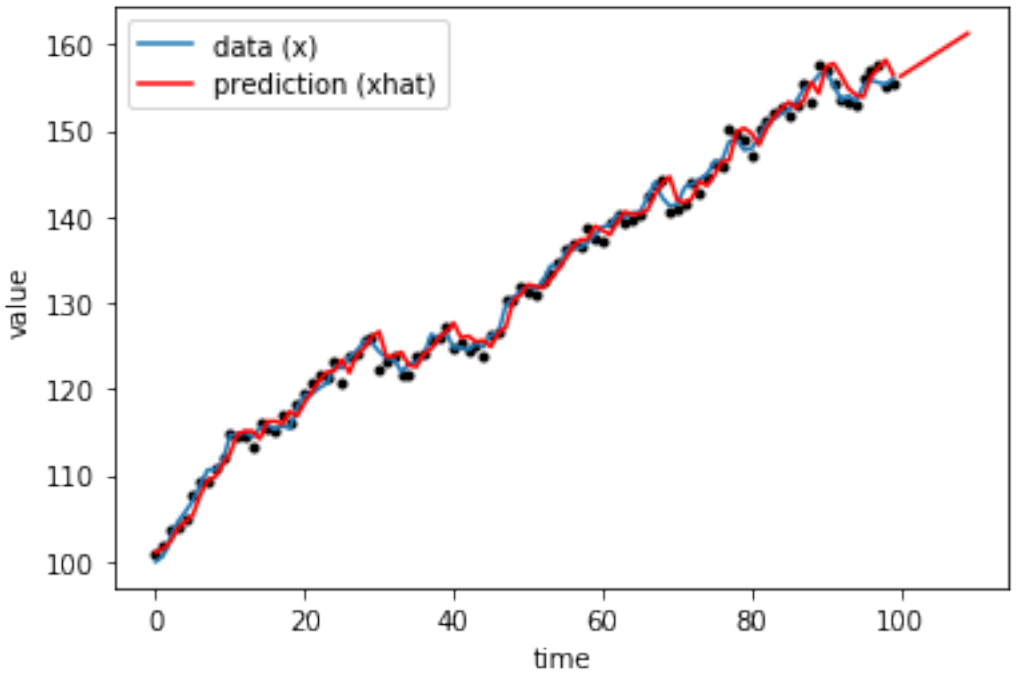
Lec15, Question 4: Suppose I change 0.8 to 5 in defining the observations `x = a + 0.8*randn(n)`. What will happen? * A. The blue line will trend downward * B. The blue line will trend upward more quickly * C. The blue line will stay at 0 * D. The blue line will bounce around more * E. The black dots will be further away from the blue line.

Lec15, Question 5: Suppose I multiply `randn()` by 3 in defining the level `a[i] = a[i-1] + b[i-1] + randn()`. What will happen? * A. The blue line will trend downward * B. The blue line will trend upward more quickly * C. The blue line will stay at 0 * D. The blue line will bounce around

more * E. The black dots will be further away from the blue line.

[12]:
```
# fit
fit = Holt(x).fit()
xhat = fit.fittedvalues

# forecast, too!
xhat = fit.fittedvalues
l = 10 # how far out to forecast
forecast = fit.forecast(l)

# plot with forecast
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), xhat, color='red')
plt.plot(range(n,n+l), forecast, color='red')
plt.legend(['data (x)','prediction (xhat)'])
plt.xlabel('time')
plt.ylabel('value')
```

[12]:
```
Text(0, 0.5, 'value')
```

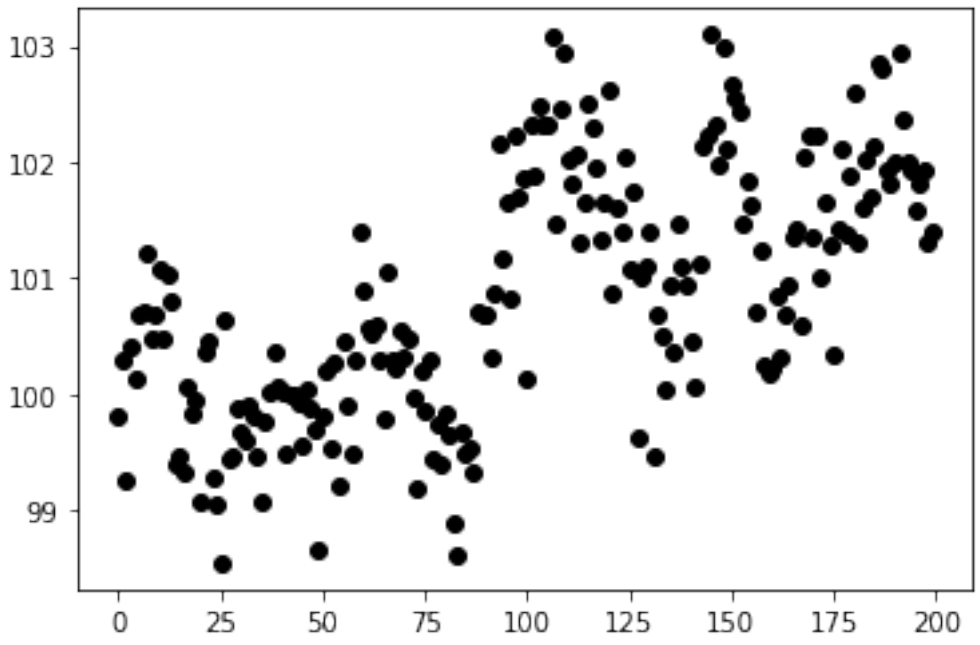


3 Review of the forecasting material covered so far (April 13, 2022)

[118]:
```
# Example with wandering mean and no trend
n = 200
a = np.zeros(n)
b = np.zeros(n)
a[0] = 100 # initial level

for i in range(1,n):
    a[i] = a[i-1] + 0.3*randn()

# The data we observe is the wandering mean + extra noise
x = a + 0.5*randn(n)
```

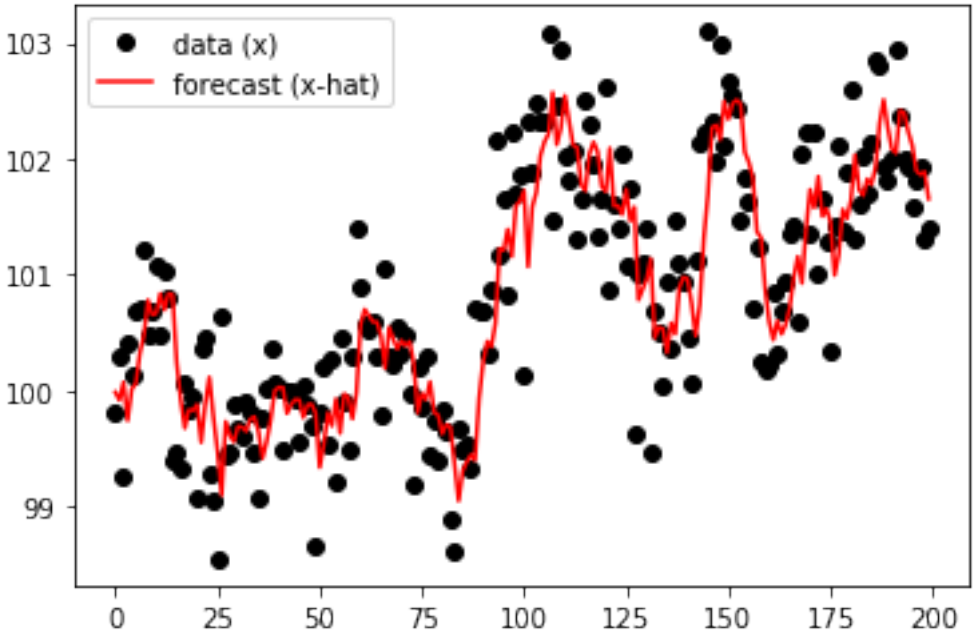
[119]:
```
# In a real-world prediction task, we only observe the black dots
plt.plot(x,'o',color='black')
```

[119]: [<matplotlib.lines.Line2D at 0x7fbfe8ff9b10>]



[126]:
```
# Prediction using Simple Exponential Smoothing
fit = SimpleExpSmoothing(x).fit()
plt.plot(x,'o',color='black')
plt.plot(range(n),fit.fittedvalues,'-',color='red')
plt.legend(['data (x)','forecast (x-hat)'])
```

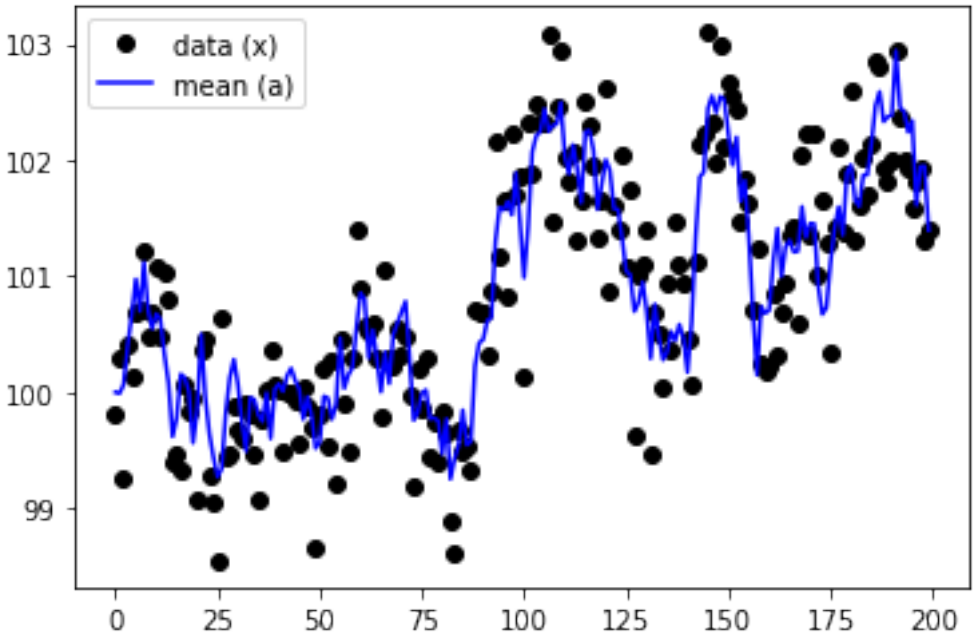
[126]: <matplotlib.legend.Legend at 0x7fbfe9135f50>



3.0.1 What is the blue line?

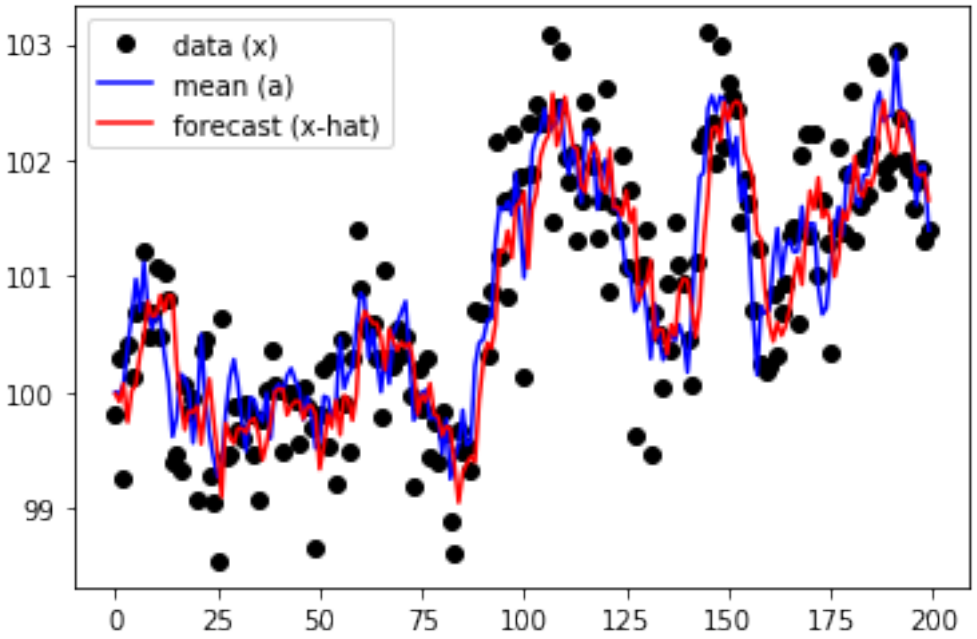
[123]:
```
# To better understand how to do well in the real-world prediction task,
# also look at the blue line.  The best that we can do in our prediction
# is to predict a value equal to the blue line.
plt.plot(x,'o',color='black')
plt.plot(a,'-',color='blue')
plt.legend(['data (x)','mean (a)'])
```

[123]: <matplotlib.legend.Legend at 0x7fbfb8ce0790>



[124]:
```
# Our forecast
fit = Simple ExpSmoothing(x).fit()
plt.plot(x,'o',color='black')
plt.plot(a,'-',color='blue')
plt.plot(range(n),fit.fittedvalues,'-',color='red')
plt.legend(['data (x)','mean (a)', 'forecast (x-hat)'])
```

[124]: <matplotlib.legend.Legend at 0x7fbfb8b5f210>

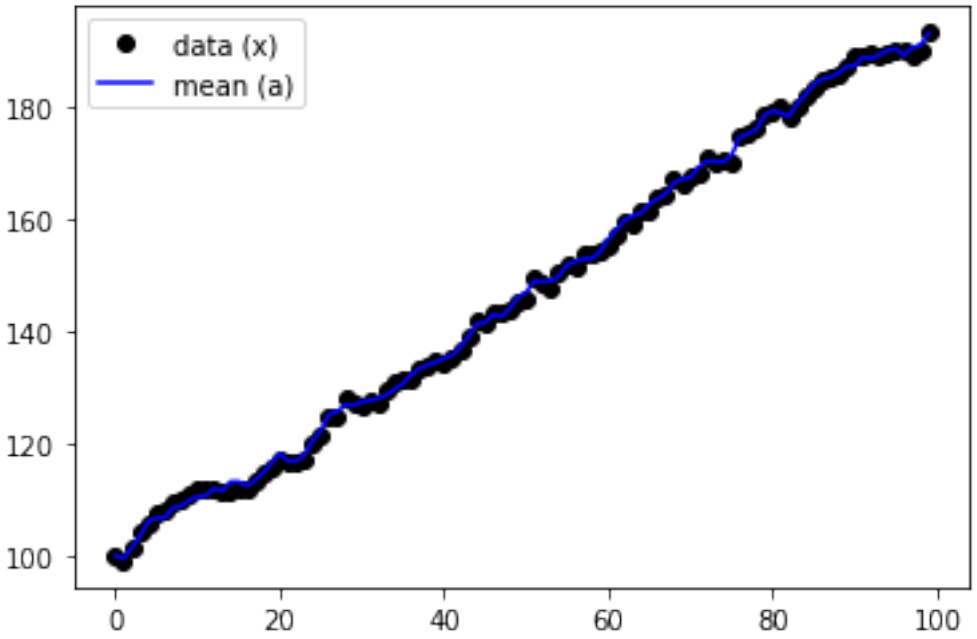


3.0.2 Wandering mean with a trend

[136]:
```
# Example data, wandering mean with a trend that DOESN'T CHANGE
n = 100
a = np.zeros(n)
a[0] = 100 # initial mean
b = 1

for i in range(1,n):
    # mean has a trend + wanders with extra random movement
    a[i] = a[i-1] + b + randn()

# We observe the level + extra noise
x = a + 0.8*randn(n)
```

[137]:
```
plt.plot(x,'o',color='black')
plt.plot(a,'-',color='blue')
plt.legend(['data (x)','mean (a)'])
```

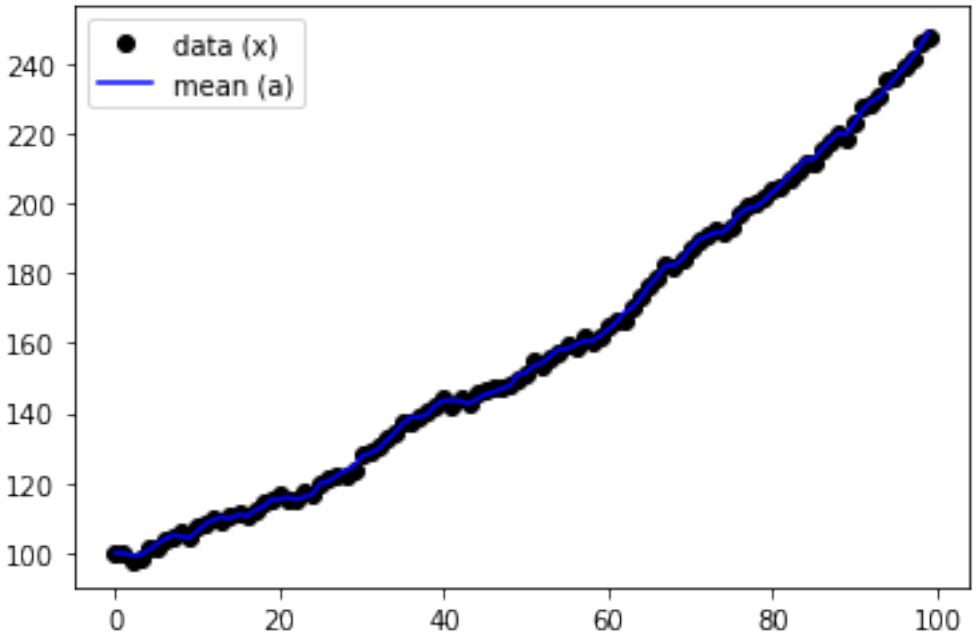
[137]: <matplotlib.legend.Legend at 0x7fbfe928bdd0>



[141]:
```
# Example data, wandering mean with a trend that ALSO WANDERS
n = 100
a = np.zeros(n)
b = np.zeros(n)
a[0] = 100 # initial mean
b[0] = 1 # initial trend

for i in range(1,n):
    # trend wanders
    b[i] = b[i-1] + 0.1*randn()

    # mean has a trend + wanders with extra random movement
    a[i] = a[i-1] + b[i-1] + randn()

# We observe the level + extra noise
x = a + 0.8*randn(n)
```

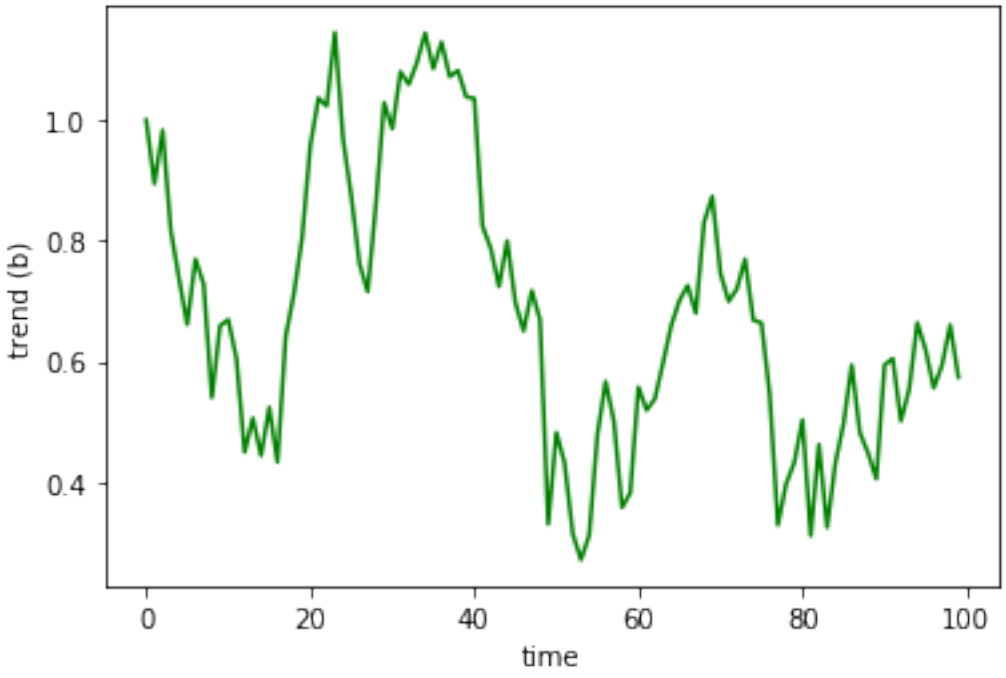
[142]:
```
plt.plot(x,'o',color='black')
plt.plot(a,'-',color='blue')
plt.legend(['data (x)','mean (a)'])
```

[142]: <matplotlib.legend.Legend at 0x7fbff8da4150>



[147]:
```
plt.plot(b,color='green')
plt.xlabel('time')
plt.ylabel('trend (b)')
```

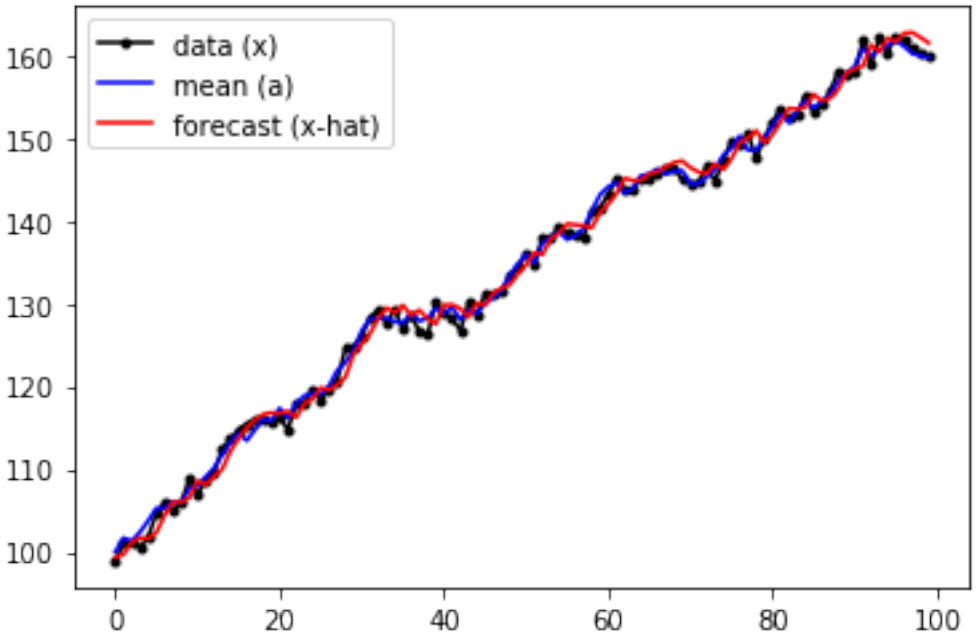
[147]: Text(0, 0.5, 'trend (b)')



[148]:
```
# Actual example from the slides

n = 100
a = np.zeros(n)
b = np.zeros(n)
a[0] = 100 # initial level
b[0] = 1 # initial trend
for i in range(1,n):
    # trend wanders
    b[i] = b[i-1] + 0.1*randn()
    # level moves with the trend + extra random movement
    a[i] = a[i-1] + b[i-1] + randn()
# We observe the level + extra noise
x = a + 0.8*randn(n)
```

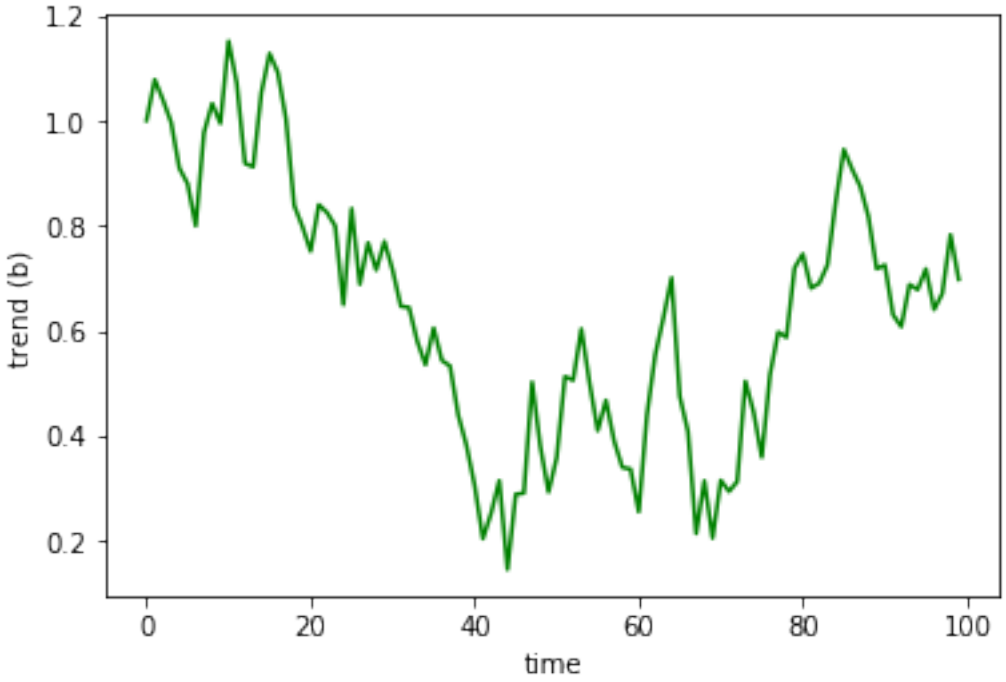
[155]:
```
fit = Holt(x).fit()
xhat = fit.fittedvalues
plt.plot(range(n), x, marker='.', color='black')
p = plt.plot(a,color='blue')
plt.plot(range(n), xhat, color='red')
plt.legend(['data (x)','mean (a)','forecast (x-hat)'])
```

[155]: <matplotlib.legend.Legend at 0x7fbff8e57890>



[152]:
```
plt.plot(b,color='green')
plt.xlabel('time')
plt.ylabel('trend (b)')
```

[152]:
```
Text(0, 0.5, 'trend (b)')
```



4 Examples of fitting various exponential smoothing models, including Holt-Winters

Let's show how to use the `ExponentialSmoothing` class to control how we fit the level, slope, and seasonality.

[13]:
```
ExponentialSmoothing?
```

[14]:
```
# call method with just a timeseries
# -> fits a model with just a level (no slope or seasonality)
# -> learns the smoothing_level parameter
model = ExponentialSmoothing(x).fit()
model.params
```

[14]:
```
{'smoothing_level': 0.9468381948039039,
 'smoothing_slope': nan,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
```

```
 'initial_level': 100.96527562941907,
 'initial_slope': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[15]:
```
# you can control the smoothing_level parameter yourself if you prefer
model = ExponentialSmoothing(x).fit(smoothing_level=0.2)
model.params
```

[15]:
```
{'smoothing_level': 0.2,
 'smoothing_slope': nan,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 105.49579907145247,
 'initial_slope': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[16]:
```
# can also add an (additive or multiplicative) trend
model = ExponentialSmoothing(x, trend='add').fit()
model.params
```

[16]:
```
{'smoothing_level': 0.7758755658994141,
 'smoothing_slope': 0.0,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 100.49865806955047,
 'initial_slope': 0.5522313840715876,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[17]:
```
# you can control the smoothing_level and smoothing_slope parameters yourself␣
↪if you prefer
model = ExponentialSmoothing(x).fit(smoothing_level=0.2,smoothing_slope=.5)
model.params
```

[17]:
```
{'smoothing_level': 0.2,
 'smoothing_slope': 0.5,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 105.49579907145247,
```

```
 'initial_slope': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[18]:
```
# can add a trend and seasonality
model = ExponentialSmoothing(x, trend='add', seasonal='mul',␣
 ↪seasonal_periods=12).fit()
model.params
```

```
/usr/local/anaconda3/lib/python3.7/site-
packages/statsmodels/tsa/holtwinters.py:712: ConvergenceWarning: Optimization
failed to converge. Check mle_retvals.
  ConvergenceWarning)
```

[18]:
```
{'smoothing_level': 0.15789473684210525,
 'smoothing_slope': 0.05262996452280781,
 'smoothing_seasonal': 0.8421052631578947,
 'damping_slope': nan,
 'initial_level': 131.51476467407463,
 'initial_slope': 0.7993806515762261,
 'initial_seasons': array([0.81379309, 0.80571977, 0.80549195, 0.79369524,
0.78386635,
        0.79127054, 0.79209727, 0.78851996, 0.79527574, 0.80089008,
        0.8173773 , 0.81699996]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[19]:
```
# can control parameters explicitly
# -> can control some parameters and optimize over others
model = ExponentialSmoothing(x, trend='add', seasonal='mul',␣
 ↪seasonal_periods=12)
model.fit(smoothing_seasonal=.5)
model.params
```

```
/usr/local/anaconda3/lib/python3.7/site-
packages/statsmodels/tsa/holtwinters.py:712: ConvergenceWarning: Optimization
failed to converge. Check mle_retvals.
  ConvergenceWarning)
```

[19]:
```
{'smoothing_level': 0.49999957379338855,
 'smoothing_slope': 3.0750509295697556e-05,
 'smoothing_seasonal': 0.5,
 'damping_slope': nan,
 'initial_level': 131.51146176256987,
```

```
 'initial_slope': 0.7650481179847815,
 'initial_seasons': array([0.75455581, 0.75022466, 0.76069261, 0.75624499,
0.74888182,
        0.75234204, 0.74640352, 0.74358706, 0.7447482 , 0.74532364,
        0.75663378, 0.75849235]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

5 Application: predicting level in Lake Huron

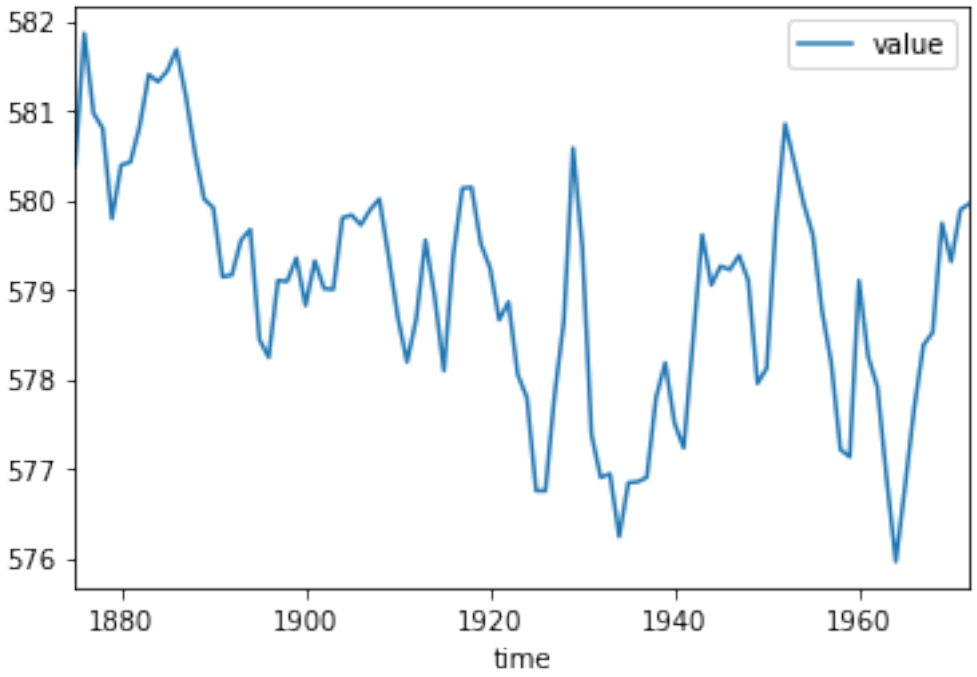
[22]:
```
huron = pd.read_csv('LakeHuron.csv')
huron.head()
```

[22]:

	Unnamed: 0	time	value
0	1	1875	580.38
1	2	1876	581.86
2	3	1877	580.97
3	4	1878	580.80
4	5	1879	579.79

[23]:
```
huron.plot('time','value')
```

[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfb872c950>
```

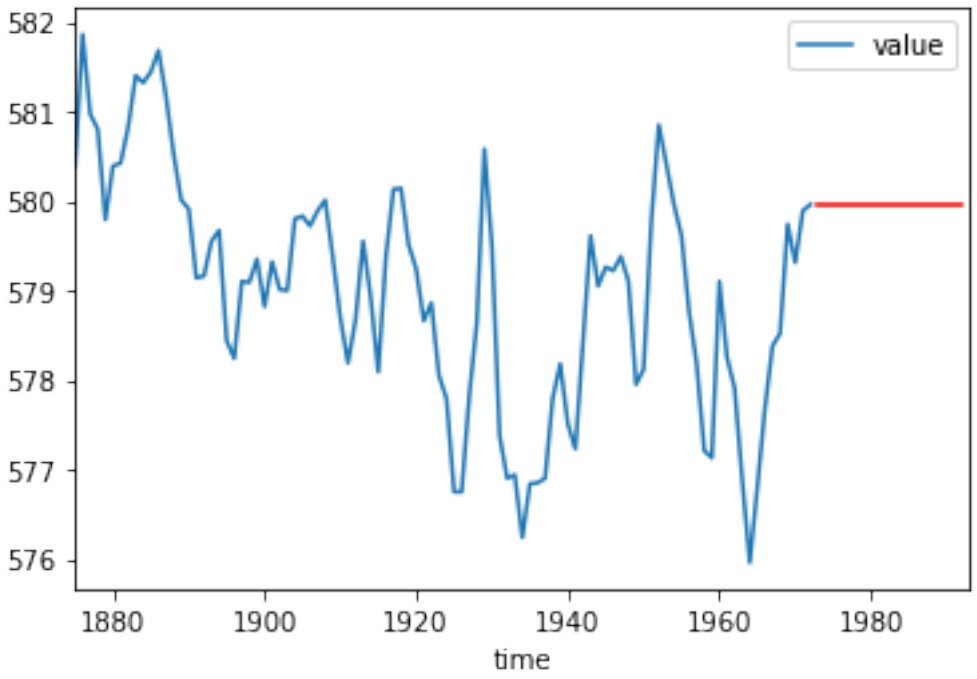


[24]:
```
# forecast with no trend
model = ExponentialSmoothing(huron['value']).fit()
model.params
```

[24]:
```
{'smoothing_level': 1.0,
 'smoothing_slope': nan,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 580.38,
 'initial_slope': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[25]:
```
# forecast using this model
lookahead = 20
forecast = model.forecast(lookahead)

# plot the forecast
last_obs = int(huron['time'].tail(1)) # last observed year
huron.plot('time','value')
# plt.plot(huron['time'], model.fittedvalues, color='red')
plt.plot(range(last_obs+1, last_obs+lookahead+1), model.forecast(lookahead),␣
 ↪'r')
plt.xlim(huron['time'][0],last_obs+lookahead+1)
```

[25]:
```
(1875, 1993)
```



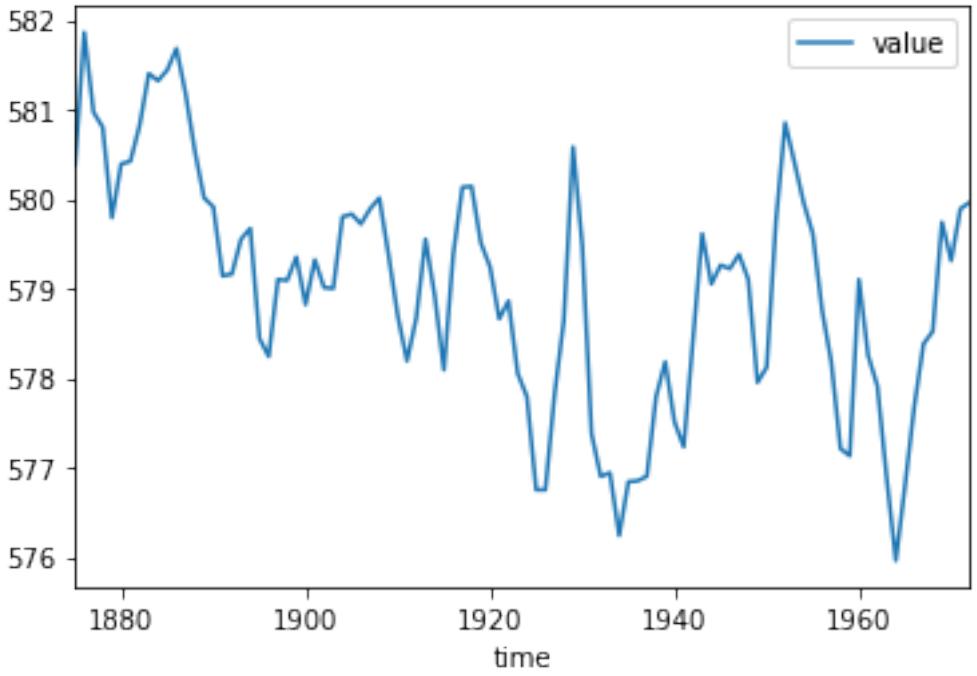
[26]:
```
# forecast; allow a trend
# the following two lines of code are equivalent
model = ExponentialSmoothing(huron['value'], trend='add').fit()
model = Holt(huron['value']).fit()
# statsmodels decides data has no trend: smoothing_slope = 0

# we can also force a trend to be fit
# model = Holt(huron['value']).fit(smoothing_slope=.1) # force a trend
model.params
```

[26]:
```
{'smoothing_level': 1.0,
 'smoothing_slope': 0.0,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 580.3800000784883,
 'initial_slope': 0.0,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[27]:
```
lookahead = 20
forecast = model.forecast(lookahead)

# plot the forecast
```

```
last_obs = int(huron['time'].tail(1)) # last observed year
huron.plot('time','value')
# plt.plot(huron['time'], model.fittedvalues, color='red')
plt.plot(range(last_obs+1, last_obs+lookahead+1), forecast, 'r')
```

[27]: [<matplotlib.lines.Line2D at 0x7fbfb87bb210>]



6 Application: US population

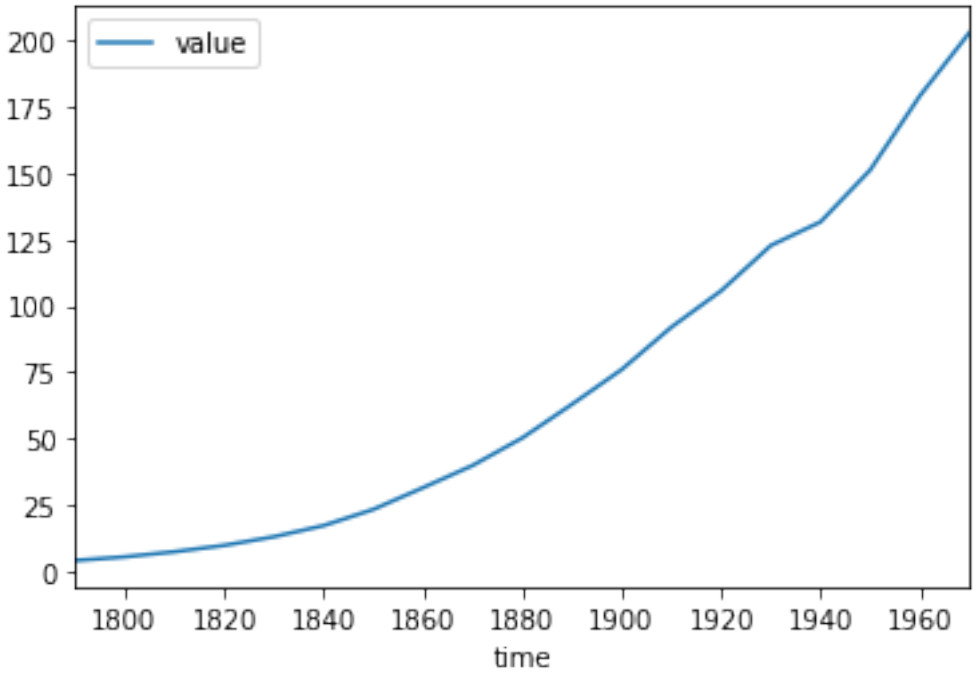
[29]:
```
pop = pd.read_csv('uspop.csv')
pop.head()
```

[29]:

	Unnamed: 0	time	value
0	1	1790	3.93
1	2	1800	5.31
2	3	1810	7.24
3	4	1820	9.64
4	5	1830	12.90

[30]:
```
pop.plot('time','value')
```

[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbfe8c40350>

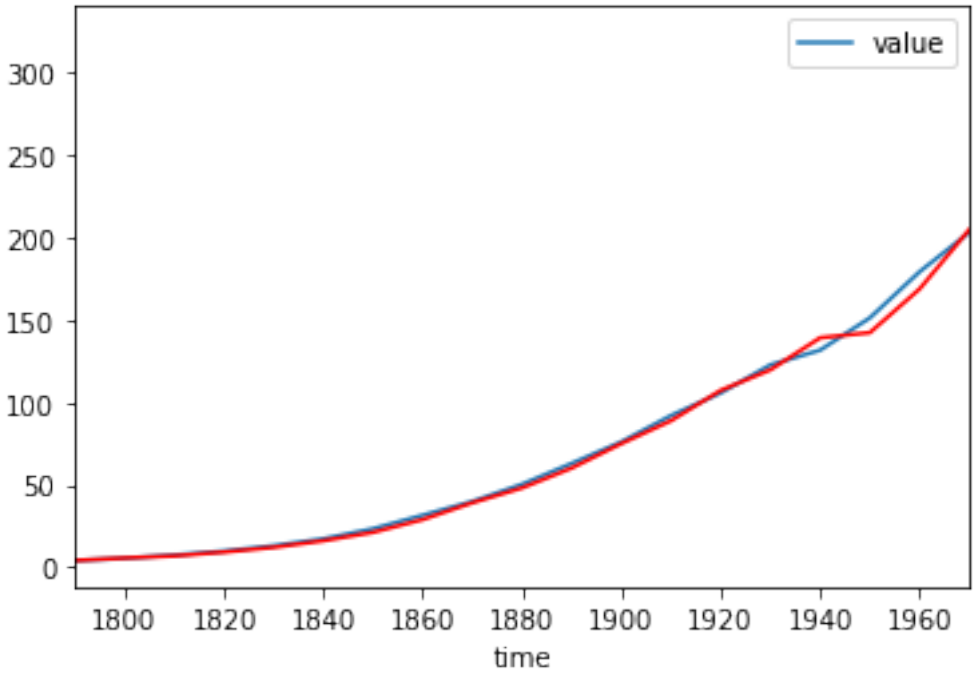


[31]:
```
model = Holt(pop['value']).fit()
# model = ExponentialSmoothing(pop['value'], trend='add').fit() # equivalent
model.params
```

[31]:
```
{'smoothing_level': 1.0,
 'smoothing_slope': 0.7703405139056307,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 2.3843199665165553,
 'initial_slope': 1.5456813435767331,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[32]:
```
lookahead = 5
forecast = model.forecast(lookahead)

# plot the forecast
last_obs = int(pop['time'].tail(1)) # last observed year
pop.plot('time','value')
plt.plot(pop['time'], model.fittedvalues, color='red')
plt.plot(range(last_obs+10, last_obs+10*lookahead+10,10), forecast, 'r')
```

[32]:
```
[<matplotlib.lines.Line2D at 0x7fbfe8c87350>]
```

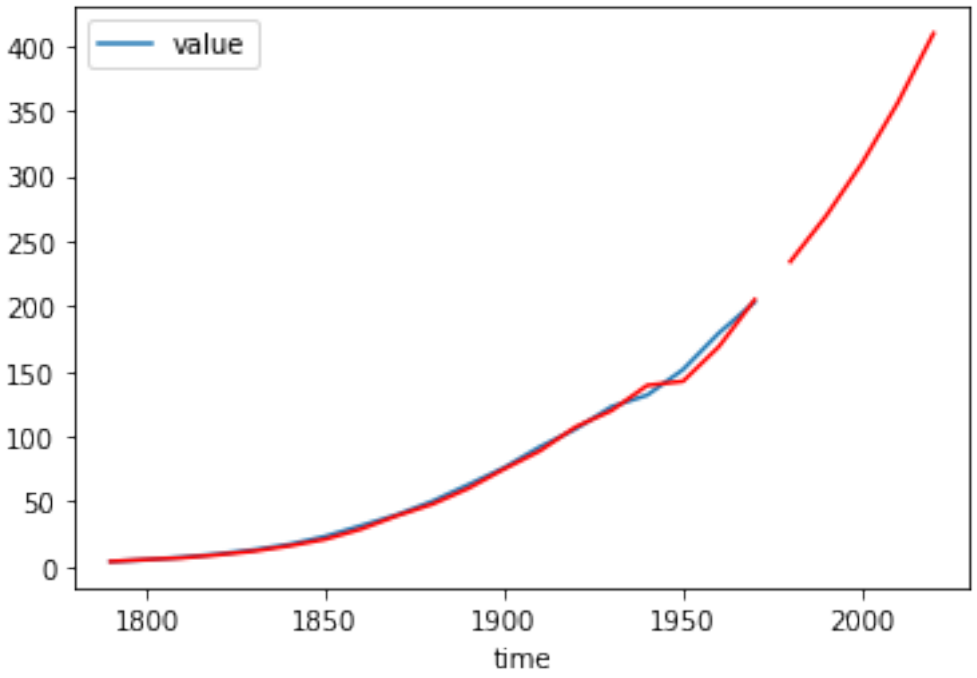


[33]:
```
# would a multiplicative trend forecast better?
modelmul = ExponentialSmoothing(pop['value'], trend='mul').fit()
modelmul.params
```

[33]:
```
{'smoothing_level': 0.8811957674352418,
 'smoothing_slope': 0.7155589057143448,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 2.917387918320182,
 'initial_slope': 1.3523602243770463,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[35]:
```
lookahead = 5
forecast = modelmul.forecast(lookahead)

# plot the forecast
last_obs = int(pop['time'].tail(1)) # last observed year
pop.plot('time','value')
plt.plot(pop['time'], model.fittedvalues, color='red')
plt.plot(range(last_obs+10, last_obs+10*lookahead+10,10), forecast, 'r')
plt.xlim(1780,2030)
```

[35]: (1780, 2030)



Now the curve increases exponentially! Interestingly, true behaviour for US pop is somewhere between these two predictions: 2020 population is 330M.

To compare models, we can look at the sum of square errors: sum(forecast - observed value)^2

[36]:
```
print("Linear trend model has error", model.sse)
print("Multiplicative trend model has error", modelmul.sse)
```

```
Linear trend model has error 299.5536250607196
Multiplicative trend model has error 318.49393589058934
```

Linear trend model gets slightly lower error.

7 Application: predicting co2 levels

[37]:
```
co2 = sm.datasets.co2.load()
co2 = pd.DataFrame(co2.data)
co2.head()
```

```
/usr/local/anaconda3/lib/python3.7/site-
packages/statsmodels/datasets/utils.py:344: FutureWarning: load will return
datasets containing pandas DataFrames and Series in the Future.  To suppress
this message, specify as_pandas=False
  FutureWarning)
```

[37]:
```
        index    co2
0 1958-03-29  316.1
1 1958-04-05  317.3
2 1958-04-12  317.6
3 1958-04-19  317.5
4 1958-04-26  316.4
```

[38]:
```
co2['index']
```

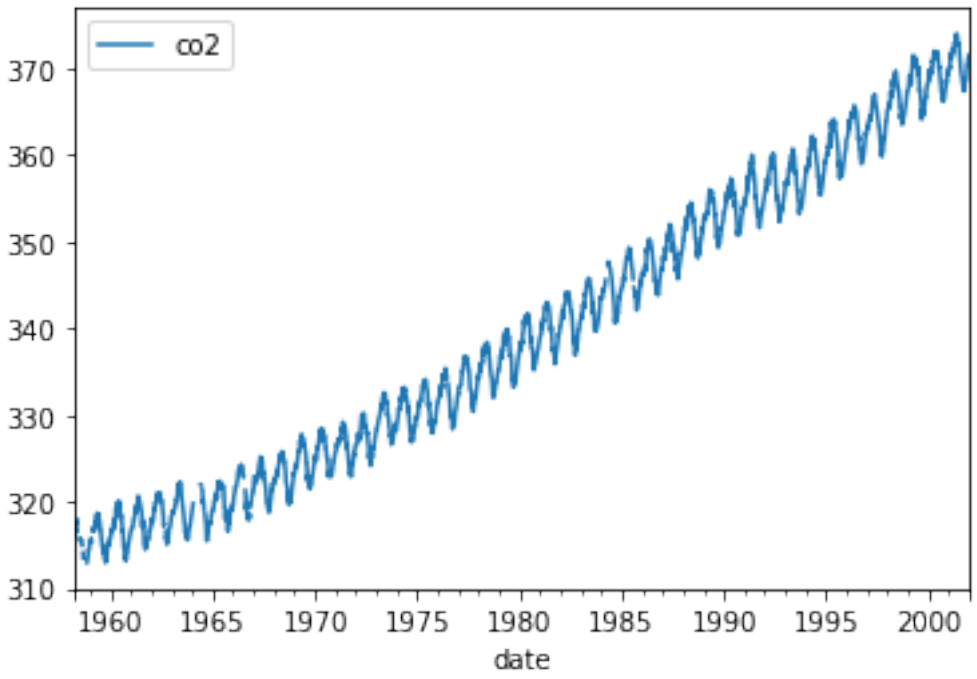
[38]:
```
0      1958-03-29
1      1958-04-05
2      1958-04-12
3      1958-04-19
4      1958-04-26
          …
2279   2001-12-01
2280   2001-12-08
2281   2001-12-15
2282   2001-12-22
2283   2001-12-29
Name: index, Length: 2284, dtype: datetime64[ns]
```

[39]:
```
# let's tell pandas that the first column consists of datetimes
dates = [pd.to_datetime(d, format = '%Y%m%d') for d in co2['index']]
co2['date'] = dates

co2.head()
```

[39]:
```
        index    co2       date
0 1958-03-29  316.1 1958-03-29
1 1958-04-05  317.3 1958-04-05
2 1958-04-12  317.6 1958-04-12
3 1958-04-19  317.5 1958-04-19
4 1958-04-26  316.4 1958-04-26
```

[40]:
```
co2.plot('date','co2')
```

[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfb88ca590>
```



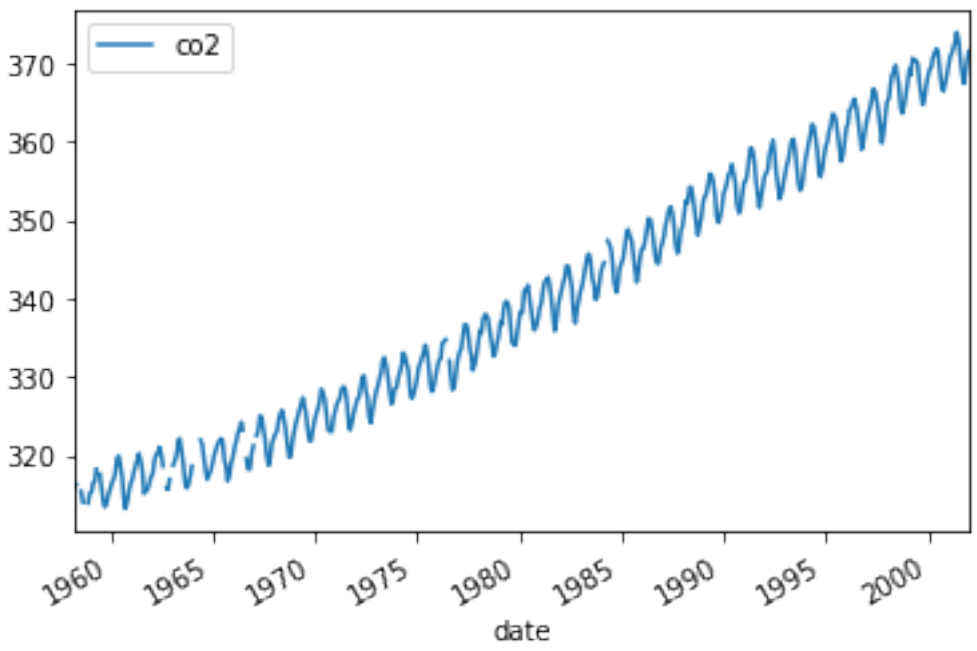
[52]:
```
# let's resample the data on a monthly basis
# to do this, we need the data to be indexed by datetime
co2.index = dates
co2monthly = co2.resample('M').ffill()

# change index back to integers when we're done
co2monthly.index = range(1,len(co2monthly.index)+1)
co2monthly.head()
```

[52]:

	index	co2	date
1	1958-03-29	316.1	1958-03-29
2	1958-04-26	316.4	1958-04-26
3	1958-05-31	NaN	1958-05-31
4	1958-06-28	NaN	1958-06-28
5	1958-07-26	315.5	1958-07-26

[53]:
```
co2monthly.plot('date','co2')
```

[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbff877e7d0>
```



[57]:
```
# drop all but the last 200 rows
rows_to_keep = 200
co2monthly = co2monthly.tail(rows_to_keep)
co2monthly.index = range(1,rows_to_keep+1)
co2monthly.head()
```

[57]:

	index	co2	date
1	1985-05-25	348.1	1985-05-25
2	1985-06-29	347.4	1985-06-29
3	1985-07-27	345.7	1985-07-27
4	1985-08-31	343.7	1985-08-31
5	1985-09-28	342.1	1985-09-28

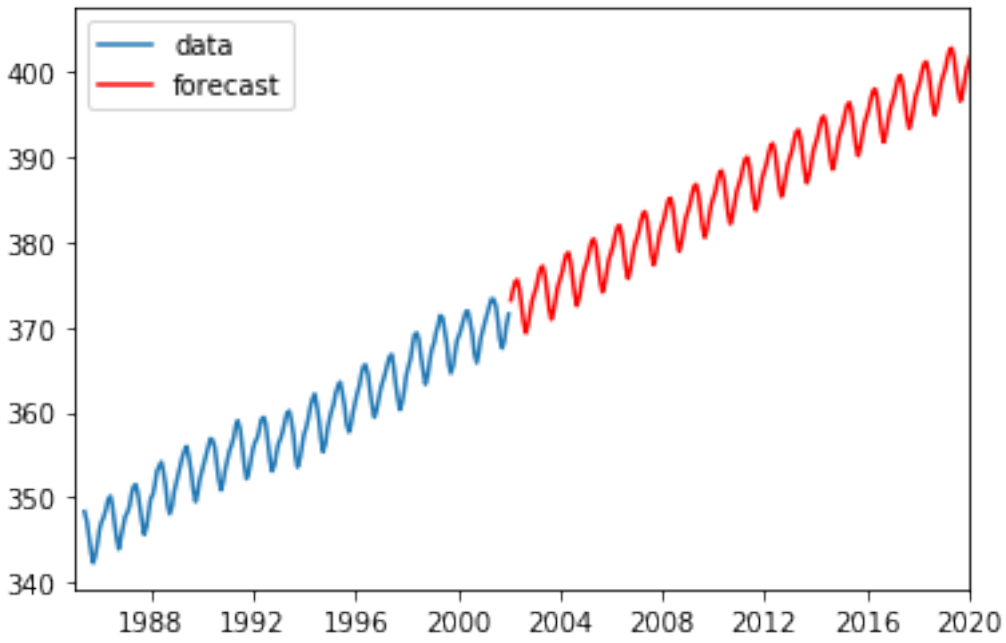
[58]:
```
# no trend
# model = SimpleExpSmoothing(co2monthly['co2']).fit()

# no seasonality
# model = Holt(co2monthly['co2']).fit()

# trend + seasonality
model =␣
 ↪ExponentialSmoothing(co2monthly['co2'],trend='add',seasonal='add',seasonal_periods=12).
 ↪fit()

model.params
```

[58]:
```
{'smoothing_level': 0.430051728736709,
 'smoothing_slope': 3.928092497252184e-14,
 'smoothing_seasonal': 1.0255978230728398e-12,
 'damping_slope': nan,
 'initial_level': 360.1937859121016,
 'initial_slope': 0.13335287409594263,
 'initial_seasons': array([-11.98338193, -13.16971381, -15.13251755,
-17.38356078,
        -18.75217884, -17.99727221, -16.57184508, -15.35811044,
        -14.64724174, -13.96190419, -12.95776703, -12.09736358]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In the full seasonal model, smoothing_level and smoothing_seasonal are very small. In other words, we want to average over *all* the data to estimate the seasonal component and slope: the statistics of the time series are changing very slowly with time

[70]:
```
# plot values and fitted values
#co2monthly.plot('date','co2')
plt.plot(co2monthly['date'], model.fittedvalues)

# plot forecast
lookahead = 220 # use 220 to get to 2020
future = 2002
future_dates = pd.date_range(start='2001-12-29', periods=lookahead+1,␣
 ↪freq='M')[1:]
plt.plot(future_dates, model.forecast(lookahead), 'r')
plt.legend(['data','forecast'])
plt.xlim('01-01-1985','01-01-2020')
```

[70]:
```
(724642.0, 737425.0)
```



daily CO2 today (April 2020): around 415ppm. Not a bad prediction!

8 Application: Airline passengers

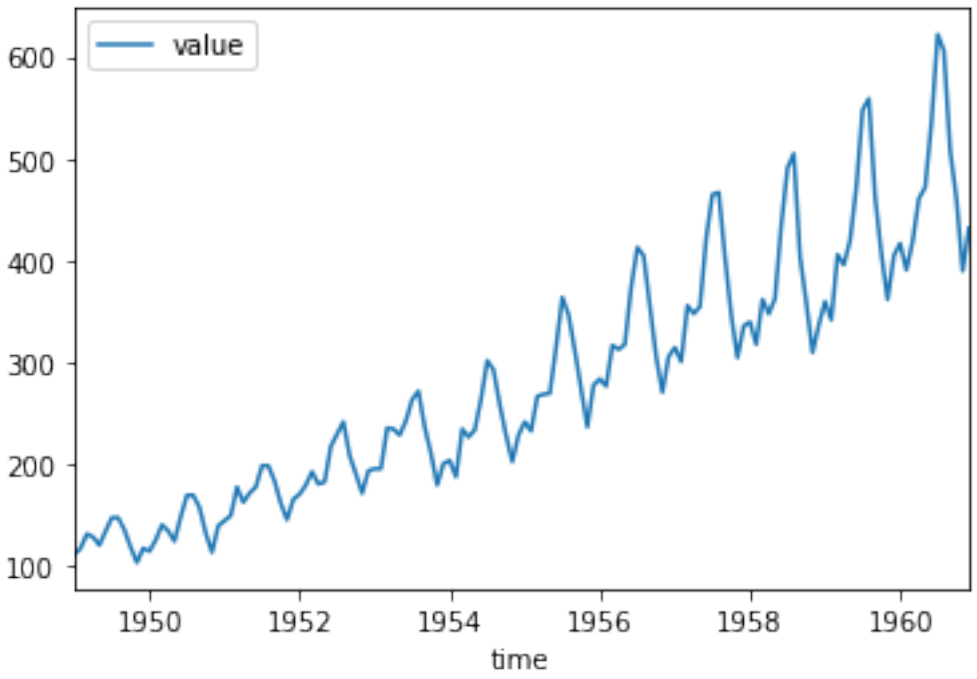
[73]:
```
air = pd.read_csv('AirPassengers.csv')
air.head(12)
```

[73]:

	Unnamed: 0	time	value
0	1	1949.000000	112
1	2	1949.083333	118
2	3	1949.166667	132
3	4	1949.250000	129
4	5	1949.333333	121
5	6	1949.416667	135
6	7	1949.500000	148
7	8	1949.583333	148
8	9	1949.666667	136
9	10	1949.750000	119
10	11	1949.833333	104
11	12	1949.916667	118

[74]:
```
air.plot('time','value')
```

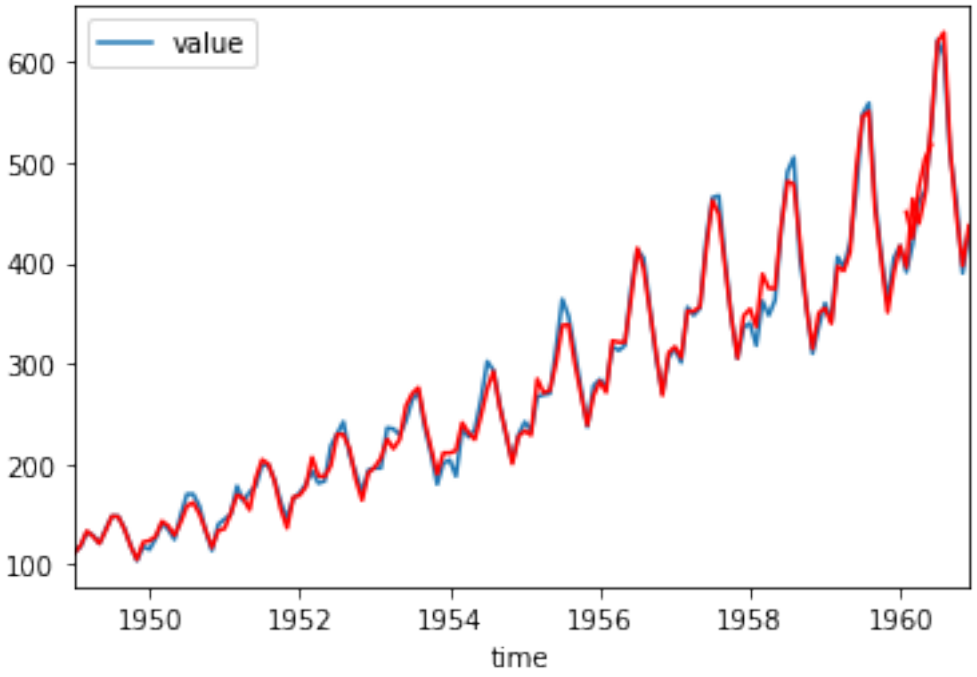
[74]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfd8821650>
```



[75]:
```
# try modeling with additive vs multiplicative trend and seasonality
model = ExponentialSmoothing(air['value'], trend='mul', seasonal='mul', \
                             seasonal_periods=12).fit()
print("Mean square error:", model.sse / len(air.index))

lookahead = 5
forecast = model.forecast(lookahead)

# plot the forecast
last_obs = int(air['time'].tail(1)) # last observed year
air.plot('time','value')
plt.plot((np.array(range(lookahead))+1)/12+last_obs, forecast, 'r')
plt.plot(air['time'], model.fittedvalues, color='red')
```

```
Mean square error: 109.75901886450116
```

[75]:
```
[<matplotlib.lines.Line2D at 0x7fbfd8894a90>]
```



[76]:
```
# formally we could check all 4 options:
for trend in ['add', 'mul']:
    for seasonal in ['add', 'mul']:
        model = ExponentialSmoothing(air['value'], \
                                     trend=trend, \
                                     seasonal=seasonal, \
                                     seasonal_periods=12).fit()
        print("Mean square error (trend="+trend+", seasonal="+seasonal+"):", \
              model.sse / len(air.index))
```

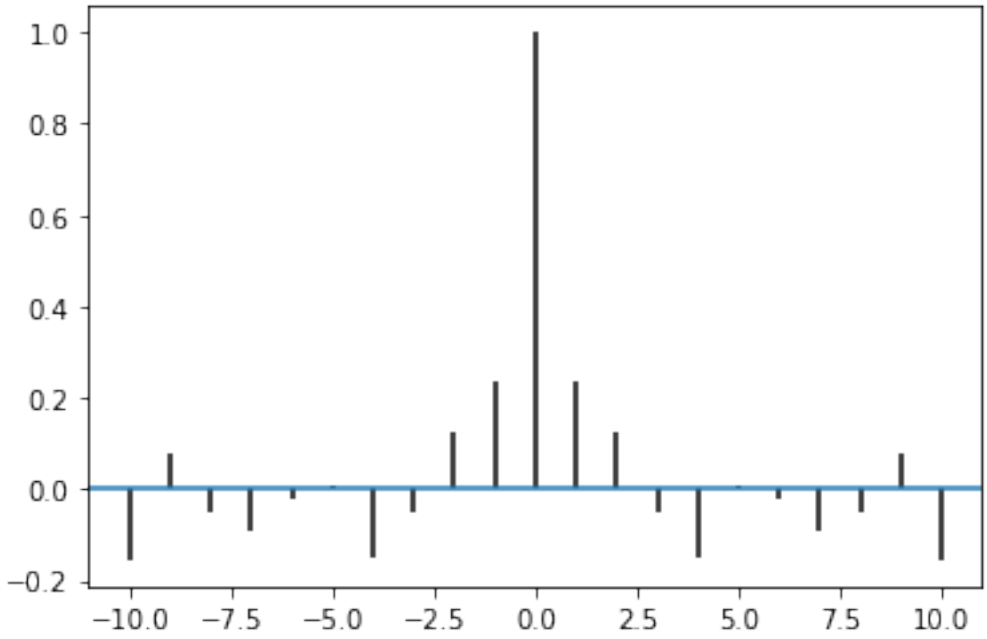
```
/usr/local/anaconda3/lib/python3.7/site-
packages/statsmodels/tsa/holtwinters.py:712: ConvergenceWarning: Optimization
failed to converge. Check mle_retvals.
  ConvergenceWarning)

Mean square error (trend=add, seasonal=add): 155.04359563173978
Mean square error (trend=add, seasonal=mul): 110.78389191154712
Mean square error (trend=mul, seasonal=add): 155.84362531520958
Mean square error (trend=mul, seasonal=mul): 109.75901886450116
```

- model with multiplicative trend and seasonality achieves lowest error
- check residual autocorrelations of each model: in a good model, residuals should be uncorrelated

[77]:
```
model = ExponentialSmoothing(air['value'], trend='mul', seasonal='mul', \
                             seasonal_periods=12).fit()
```

```
plt.acorr(model.resid)
```

[77]:
```
(array([-10,  -9,  -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,   2,
          3,   4,   5,   6,   7,   8,   9,  10]),
 array([-0.15537875,  0.08069027, -0.05103508, -0.09040599, -0.02280288,
         0.00533036, -0.15015506, -0.04931813,  0.12217292,  0.23305559,
         1.        ,  0.23305559,  0.12217292, -0.04931813, -0.15015506,
         0.00533036, -0.02280288, -0.09040599, -0.05103508,  0.08069027,
        -0.15537875]),
 <matplotlib.collections.LineCollection at 0x7fbfca95b190>,
 <matplotlib.lines.Line2D at 0x7fbfca95bf90>)
```



Autocorrelations are smallest for mul,mul model. (But still not perfect…)

9 Application: predicting sales

Stove Top makes food products that tend to be used around holidays (e.g., Thanksgiving, Christmas, Easter) * three years of monthly data starting Apr 2000 * strong seasonality

[79]:
```
sales = pd.read_csv("SalesAndForecastsTimeSeries.csv")
sales.head()
```

[79]:

	ID	PartNumber	MonthNumber	Year	Sales	ForecastSalesNextMonth	\
0	1	285210	4	2000	14268	8793.0	
1	2	285280	4	2000	2321	NaN	
2	3	285330	4	2000	13244	8610.0	

```
3   4      285430            4  2000   2014                  3808.0
4   5      285440            4  2000   1416                  3098.0

   ForecastTwoMonthsOut
0                   NaN
1                   NaN
2                   NaN
3                   NaN
4                   NaN
```

[80]:
```
# add a datetime column, for nicer plotting
df = pd.DataFrame()
df['year'] = sales['Year']
df['month'] = sales['MonthNumber']
df['day'] = 1
dates = pd.to_datetime(df)
sales['date'] = dates
sales.head()
```

[80]:
```
   ID  PartNumber  MonthNumber  Year  Sales  ForecastSalesNextMonth  \
0   1      285210            4  2000  14268                  8793.0
1   2      285280            4  2000   2321                     NaN
2   3      285330            4  2000  13244                  8610.0
3   4      285430            4  2000   2014                  3808.0
4   5      285440            4  2000   1416                  3098.0

   ForecastTwoMonthsOut       date
0                   NaN 2000-04-01
1                   NaN 2000-04-01
2                   NaN 2000-04-01
3                   NaN 2000-04-01
4                   NaN 2000-04-01
```

9.1 plot the data

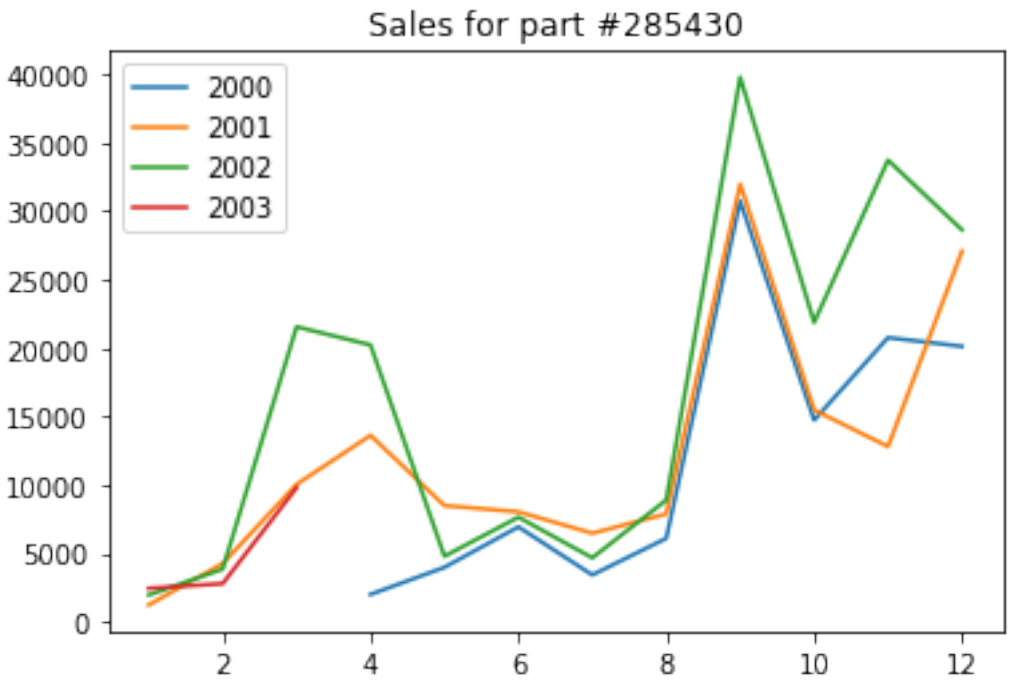
[81]:
```
# select rows corresponding to a single part number
part = 285430  # clear seasonality
# part = 285280 # no clear seasonality
df = sales.loc[sales['PartNumber']==part]
# add an integer index - needed for forecast so python knows rows are ordered
df.index = range(len(df.index))
df.head()
```

[81]:
```
   ID  PartNumber  MonthNumber  Year  Sales  ForecastSalesNextMonth  \
0   4      285430            4  2000   2014                  3808.0
1  21      285430            5  2000   3992                  6420.0
2  38      285430            6  2000   6946                  3838.0
```

```
3  55      285430            7  2000   3438                 4890.0
4  72      285430            8  2000   6110                25066.0

   ForecastTwoMonthsOut       date
0                   NaN 2000-04-01
1                   NaN 2000-05-01
2                   NaN 2000-06-01
3                   NaN 2000-07-01
4                   NaN 2000-08-01
```

[82]:
```
# plot sales over time
years = set(df['Year'])
for year in years:
    # select rows corresponding to given year
    dfyear = df.loc[df['Year']==year]
    # plot sales during that year
    plt.plot(dfyear['MonthNumber'], dfyear['Sales'], label=year)
plt.legend()
plt.title("Sales for part #"+str(part))
```

[82]:
```
Text(0.5, 1.0, 'Sales for part #285430')
```



[83]:
```
# forecast sales
model = ExponentialSmoothing(df['Sales'], trend="add", \
```

```
                                seasonal="add", seasonal_periods=12).fit()
model.params
```

[83]:
```
{'smoothing_level': 0.052631578947367204,
 'smoothing_slope': 0.05263157894736842,
 'smoothing_seasonal': 0.15789473684212466,
 'damping_slope': nan,
 'initial_level': 11957.33333333306,
 'initial_slope': 242.01388888888332,
 'initial_seasons': array([ -9943.33333333,  -7965.33333333,  -5011.33333333,
-8519.33333333,
         -5847.33333333,  18776.66666667,   2780.66666667,   8808.66666667,
          8166.66666667, -10713.33333333,  -7685.33333333,  -1905.33333333]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

[84]:
```
plt.plot(df['date'], df['Sales'], label='sales')
plt.plot(df['date'], model.fittedvalues, label='fitted values')

lookahead = 20
lastmonth = df['date'].iloc[-1]
future_dates = pd.date_range(start=lastmonth, periods=lookahead+1, freq='M')[1:]
plt.plot(future_dates, model.forecast(lookahead), label='forecast')
plt.legend()
```

[84]:
```
<matplotlib.legend.Legend at 0x7fbff8838ad0>
```

10 Prediction intervals for forecasts (not covered in Lecture 15)

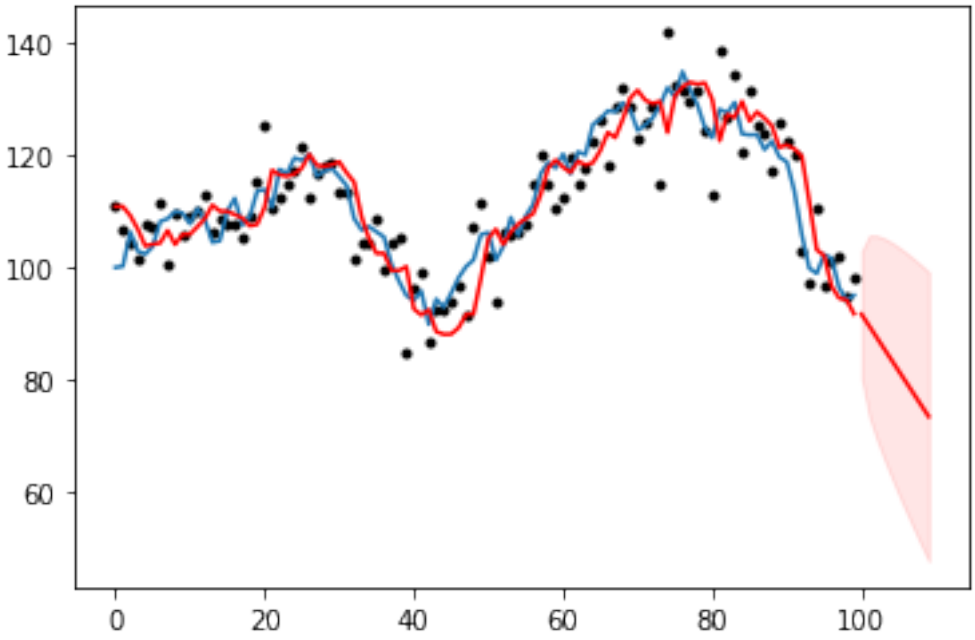
[85]:
```
# generate synthetic data
np.random.seed(0)
n = 100
a = np.zeros(n)
b = np.zeros(n)
a[0] = 100 # initial level
b[0] = -1 # initial trend

for i in range(1,n):
    b[i] = b[i-1] + 0.2*randn()      # trend changes slowly
    a[i] = a[i-1] + b[i-1] + 3*randn() # level moves with the trend + extra␣
 ↪random movement

# We observe the level + extra noise
x = a + 5*randn(n)

# fit the model
# fit = Holt(x).fit(smoothing_level=0.1,smoothing_slope=.1,optimized=False)
fit = Holt(x).fit() # optimize over the parameters
xhat = fit.fittedvalues
forecast = fit.forecast(10)
```

```
/usr/local/anaconda3/lib/python3.7/site-
packages/statsmodels/tsa/holtwinters.py:712: ConvergenceWarning: Optimization
failed to converge. Check mle_retvals.
  ConvergenceWarning)
```

[86]:
```
# these prediction intervals are approximate
#
# for more correct, use https://www.statsmodels.org/stable/generated/
 ↪statsmodels.tsa.statespace.exponential_smoothing.ExponentialSmoothing.
 ↪html#statsmodels.tsa.statespace.exponential_smoothing.ExponentialSmoothing

# error estimate involves the standard error of the noise in the observations
# and an estimate of how much the level changes (randomly) with time
se = np.sqrt(fit.sse / len(x)) # standard error of predictions
level_se = np.std(np.diff(xhat)) # how quickly model predictions change
conf_level = .1
l = len(forecast)
ub = forecast + norm.ppf(conf_level/2)*(se + level_se*np.sqrt(np.arange(l)))
lb = forecast + norm.ppf(1-conf_level/2)*(se + level_se*np.sqrt(np.arange(l)))

# plot
```

```
plt.scatter(range(n), x, marker='.', color='black')
p = plt.plot(a)
plt.plot(range(n), xhat, color='red')
plt.plot(range(n,n+l), forecast, color='red')
plt.fill_between(range(n,n+l), lb, ub, color='r', alpha=.1)
```

[86]: <matplotlib.collections.PolyCollection at 0x7fbfe8da8090>



11 Regression for forecasting (not covered in Lectuer 15)

advantages: * can use any covariates you want! * can use standard regression tools to select model, assess significance, build confidence/prediction intervals, etc

disadvantages: more complicated (beyond this class) to * update model given new (future) observations * discount observations from the distant past

[156]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
from patsy import dmatrices
```

[157]:
```
df
df['time'] = df['Year'] - 2000 + df['MonthNumber']/12
df.head()
```

/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[157]:

	ID	PartNumber	MonthNumber	Year	Sales	ForecastSalesNextMonth	\
0	4	285430	4	2000	2014	3808.0	
1	21	285430	5	2000	3992	6420.0	
2	38	285430	6	2000	6946	3838.0	
3	55	285430	7	2000	3438	4890.0	
4	72	285430	8	2000	6110	25066.0	

	ForecastTwoMonthsOut	date	time	lagged_sales	lagged_sales1	\
0	NaN	2000-04-01	0.333333	NaN	NaN	
1	NaN	2000-05-01	0.416667	2014.0	2014.0	
2	NaN	2000-06-01	0.500000	3992.0	3992.0	
3	NaN	2000-07-01	0.583333	6946.0	6946.0	
4	NaN	2000-08-01	0.666667	3438.0	3438.0	

	lagged_sales2	lagged_sales3
0	NaN	NaN
1	NaN	NaN
2	2014.0	NaN
3	3992.0	2014.0
4	6946.0	3992.0

We'll use the `C` function to tell statsmodels that the `MonthNumber` is a categorical variable. That tells it to fit a separate offset for each value of the month. Equivalently, we are adding a new binary covariate for each month: the covariate for November for the ith observation, for example, is 1 if observation i's month is November and 0 otherwise.

[89]:
```
# form the design matrix and target

# trend
Y, X = dmatrices("Sales ~ time", data=df, return_type='dataframe')

# just seasonality
Y, X = dmatrices("Sales ~ C(MonthNumber)", data=df, return_type='dataframe')

# trend and seasonality
# Y, X = dmatrices("Sales ~ time + C(MonthNumber)", data=df,␣
 ↪return_type='dataframe')

X.head()
```

[89]:
```
   Intercept  C(MonthNumber)[T.2]  C(MonthNumber)[T.3]  C(MonthNumber)[T.4]  \
0        1.0                  0.0                  0.0                  1.0
1        1.0                  0.0                  0.0                  0.0
2        1.0                  0.0                  0.0                  0.0
3        1.0                  0.0                  0.0                  0.0
4        1.0                  0.0                  0.0                  0.0

   C(MonthNumber)[T.5]  C(MonthNumber)[T.6]  C(MonthNumber)[T.7]  \
0                  0.0                  0.0                  0.0
1                  1.0                  0.0                  0.0
2                  0.0                  1.0                  0.0
3                  0.0                  0.0                  1.0
4                  0.0                  0.0                  0.0

   C(MonthNumber)[T.8]  C(MonthNumber)[T.9]  C(MonthNumber)[T.10]  \
0                  0.0                  0.0                   0.0
1                  0.0                  0.0                   0.0
2                  0.0                  0.0                   0.0
3                  0.0                  0.0                   0.0
4                  1.0                  0.0                   0.0

   C(MonthNumber)[T.11]  C(MonthNumber)[T.12]
0                   0.0                   0.0
1                   0.0                   0.0
2                   0.0                   0.0
3                   0.0                   0.0
4                   0.0                   0.0
```

Instead of forming the design matrix in one step, and fitting the model in the next, we can fit a model directly from the formula using the statsmodels formula API.

[90]:
```
# just trend
model = smf.ols(formula="Sales ~ time", data=df).fit()

# just seasonality
model = smf.ols(formula="Sales ~ C(MonthNumber)", data=df).fit()

# trend + seasonality
model = smf.ols(formula="Sales ~ time + C(MonthNumber)", data=df).fit()
model.summary()
```

[90]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.880
Model:                            OLS   Adj. R-squared:                  0.817
```

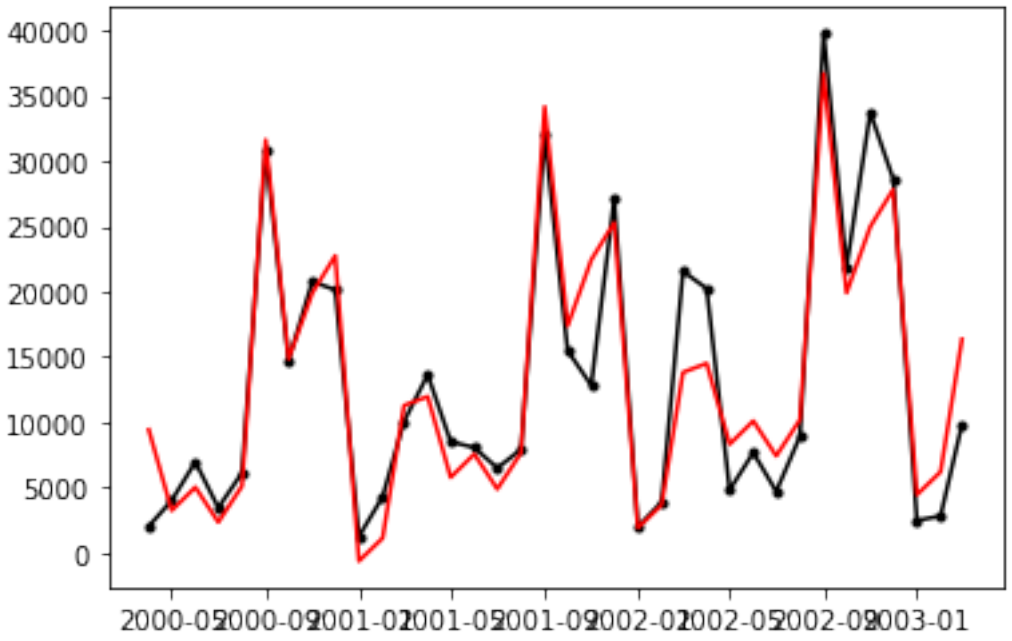
```
Method:                 Least Squares   F-statistic:                     14.01
Date:                Tue, 12 Apr 2022   Prob (F-statistic):           7.70e-08
Time:                        22:49:26   Log-Likelihood:                -346.04
No. Observations:                  36   AIC:                             718.1
Df Residuals:                      23   BIC:                             738.7
Df Model:                          12
Covariance Type:            nonrobust
================================================================================
========
                           coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--------
Intercept            -3396.0208   3244.832     -1.047      0.306   -1.01e+04
3316.425
C(MonthNumber)[T.2]   1541.8125   3695.591      0.417      0.680   -6103.101
9186.726
C(MonthNumber)[T.3]   1.149e+04   3697.995      3.107      0.005    3840.405
1.91e+04
C(MonthNumber)[T.4]   1.197e+04   3759.176      3.184      0.004    4192.573
1.97e+04
C(MonthNumber)[T.5]   5563.5000   3745.755      1.485      0.151   -2185.184
1.33e+04
C(MonthNumber)[T.6]   7136.6458   3733.872      1.911      0.069    -587.457
1.49e+04
C(MonthNumber)[T.7]   4245.1250   3723.543      1.140      0.266   -3457.611
1.19e+04
C(MonthNumber)[T.8]   6800.2708   3714.781      1.831      0.080    -884.339
1.45e+04
C(MonthNumber)[T.9]   3.312e+04   3707.596      8.932      0.000    2.54e+04
4.08e+04
C(MonthNumber)[T.10]  1.611e+04   3701.999      4.351      0.000    8448.394
2.38e+04
C(MonthNumber)[T.11]  2.096e+04   3697.995      5.669      0.000    1.33e+04
2.86e+04
C(MonthNumber)[T.12]   2.36e+04   3695.591      6.386      0.000     1.6e+04
3.12e+04
time                  2538.2500    923.697      2.748      0.011     627.436
4449.064
==============================================================================
Omnibus:                        2.765   Durbin-Watson:                   1.364
Prob(Omnibus):                  0.251   Jarque-Bera (JB):                1.721
Skew:                          -0.173   Prob(JB):                        0.423
Kurtosis:                       4.014   Cond. No.                         27.8
==============================================================================

Warnings:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

The coefficient for time is positive; it indicates that sales rise by about $2000 each month, on average, controlling for seasonality.

[91]:
```
# plot forecasts
# plt.scatter(df['date'], df['Sales'], marker='.', color='black')
plt.plot(df['date'], df['Sales'], marker='.', color='black')
plt.plot(df['date'], model.fittedvalues, color='r')
```

[91]:
```
[<matplotlib.lines.Line2D at 0x7fbff894d7d0>]
```



[92]:
```
lagged_sales = [np.nan] + list(df['Sales'][:-1])
df['lagged_sales'] = lagged_sales
df.head()
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[92]:
```
   ID  PartNumber  MonthNumber  Year  Sales  ForecastSalesNextMonth  \
0   4      285430            4  2000   2014                  3808.0
1  21      285430            5  2000   3992                  6420.0
2  38      285430            6  2000   6946                  3838.0
3  55      285430            7  2000   3438                  4890.0
4  72      285430            8  2000   6110                 25066.0

   ForecastTwoMonthsOut       date      time  lagged_sales
0                   NaN 2000-04-01  0.333333           NaN
1                   NaN 2000-05-01  0.416667        2014.0
2                   NaN 2000-06-01  0.500000        3992.0
3                   NaN 2000-07-01  0.583333        6946.0
4                   NaN 2000-08-01  0.666667        3438.0
```

[93]:
```
# drop the first observation, since lagged sale is unobserved
dflagged = df.loc[1:]
dflagged.head()
```

[93]:
```
   ID  PartNumber  MonthNumber  Year  Sales  ForecastSalesNextMonth  \
1  21      285430            5  2000   3992                  6420.0
2  38      285430            6  2000   6946                  3838.0
3  55      285430            7  2000   3438                  4890.0
4  72      285430            8  2000   6110                 25066.0
5  89      285430            9  2000  30734                     NaN

   ForecastTwoMonthsOut       date      time  lagged_sales
1                   NaN 2000-05-01  0.416667        2014.0
2                   NaN 2000-06-01  0.500000        3992.0
3                   NaN 2000-07-01  0.583333        6946.0
4                   NaN 2000-08-01  0.666667        3438.0
5                   NaN 2000-09-01  0.750000        6110.0
```

[94]:
```
model_lagged = smf.ols(formula="Sales ~ time + C(MonthNumber) + lagged_sales",␣
↪data=dflagged).fit()
model_lagged.summary()
```

[94]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.903
Model:                            OLS   Adj. R-squared:                  0.843
Method:                 Least Squares   F-statistic:                     15.08
Date:                Tue, 12 Apr 2022   Prob (F-statistic):           8.22e-08
Time:                        22:49:29   Log-Likelihood:                -332.54
No. Observations:                  35   AIC:                             693.1
Df Residuals:                      21   BIC:                             714.9
```

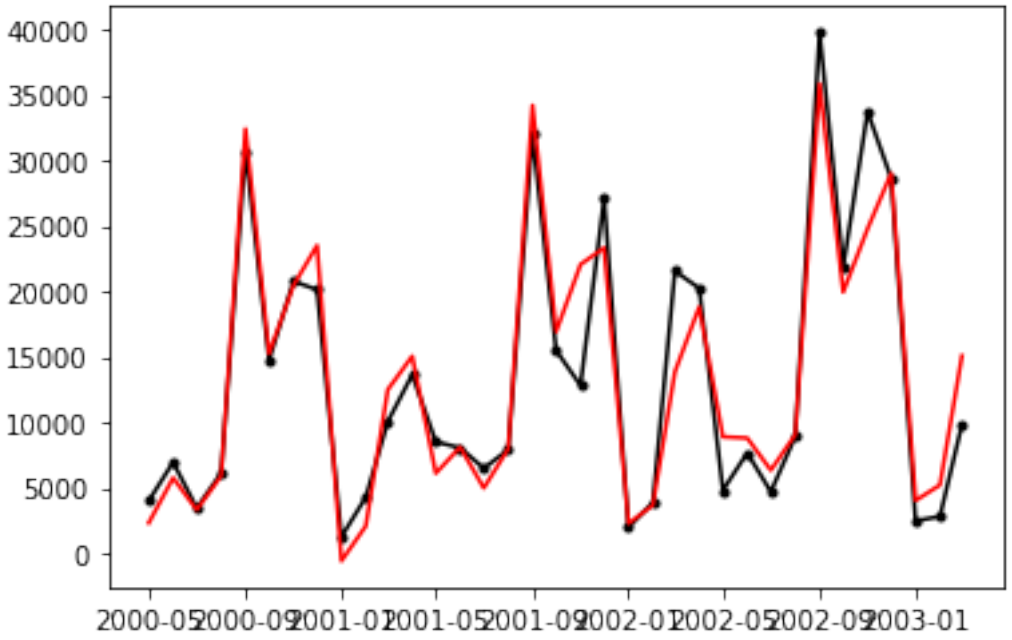
```
Df Model:                           13
Covariance Type:            nonrobust
==============================================================================
========
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
Intercept            -6237.4067   4998.212     -1.248      0.226   -1.66e+04
4156.944
C(MonthNumber)[T.2]   6375.7082   5989.215      1.065      0.299   -6079.547
1.88e+04
C(MonthNumber)[T.3]   1.606e+04   5735.775      2.800      0.011    4131.848
2.8e+04
C(MonthNumber)[T.4]   1.732e+04   4231.650      4.093      0.001    8518.893
2.61e+04
C(MonthNumber)[T.5]   7533.8793   4190.701      1.798      0.087   -1181.160
1.62e+04
C(MonthNumber)[T.6]   1.045e+04   5061.217      2.066      0.051     -70.972
2.1e+04
C(MonthNumber)[T.7]   7292.3890   4824.752      1.511      0.146   -2741.233
1.73e+04
C(MonthNumber)[T.8]   1.048e+04   5264.556      1.991      0.060    -465.880
2.14e+04
C(MonthNumber)[T.9]   3.633e+04   4873.796      7.454      0.000    2.62e+04
4.65e+04
C(MonthNumber)[T.10]  1.403e+04   3963.312      3.540      0.002    5788.913
2.23e+04
C(MonthNumber)[T.11]  2.238e+04   3748.317      5.972      0.000    1.46e+04
3.02e+04
C(MonthNumber)[T.12]  2.409e+04   3455.346      6.971      0.000    1.69e+04
3.13e+04
time                  1441.5606   1076.814      1.339      0.195    -797.798
3680.919
lagged_sales             0.2028      0.208      0.974      0.341      -0.230
0.636
==============================================================================
Omnibus:                        5.287   Durbin-Watson:                   1.776
Prob(Omnibus):                  0.071   Jarque-Bera (JB):                5.761
Skew:                           0.195   Prob(JB):                       0.0561
Kurtosis:                       4.949   Cond. No.                     3.29e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.29e+05. This might indicate that there are
```

```
strong multicollinearity or other numerical problems.
"""
```

notice that * model fit improves * effect of trend and seasons changes: eg, coefficient for trend is now lower ($1400/year) with lower p value

[95]:
```
plt.plot(dflagged['date'], dflagged['Sales'], marker='.', color='black')
plt.plot(dflagged['date'], model_lagged.fittedvalues, color='r')
```

[95]:
```
[<matplotlib.lines.Line2D at 0x7fbfd8a54dd0>]
```



[96]:
```
# can add as many lagged outcomes as you'd like!
lmax = 3
for lag in range(1,lmax+1):
    df['lagged_sales'+str(lag)] = [np.nan]*lag + list(df['Sales'][:-lag])

dflagged = df.loc[lag:]
dflagged.head()
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

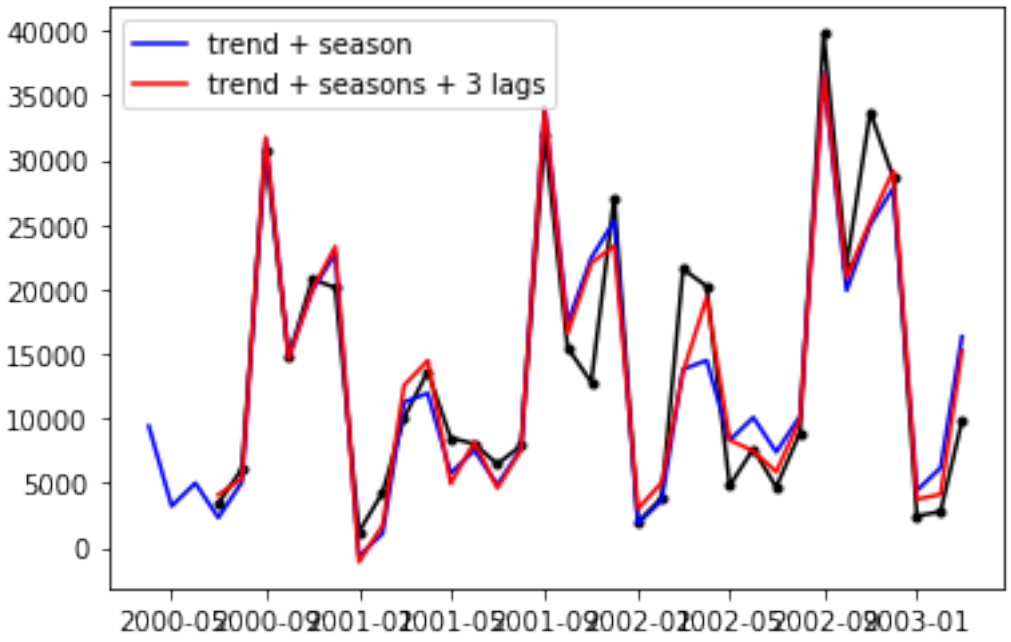
```
 after removing the cwd from sys.path.
```

[96]:
```
    ID  PartNumber  MonthNumber  Year  Sales  ForecastSalesNextMonth  \
3   55      285430            7  2000   3438                  4890.0
4   72      285430            8  2000   6110                 25066.0
5   89      285430            9  2000  30734                     NaN
6  106      285430           10  2000  14738                     NaN
7  123      285430           11  2000  20766                     NaN

   ForecastTwoMonthsOut       date      time  lagged_sales  lagged_sales1  \
3                   NaN 2000-07-01  0.583333        6946.0         6946.0
4                   NaN 2000-08-01  0.666667        3438.0         3438.0
5                   NaN 2000-09-01  0.750000        6110.0         6110.0
6                   NaN 2000-10-01  0.833333       30734.0        30734.0
7                   NaN 2000-11-01  0.916667       14738.0        14738.0

   lagged_sales2  lagged_sales3
3         3992.0         2014.0
4         6946.0         3992.0
5         3438.0         6946.0
6         6110.0         3438.0
7        30734.0         6110.0
```

[97]:
```
formula = "Sales ~ time + C(MonthNumber) + lagged_sales1 + lagged_sales2 +␣
 ↪lagged_sales3"
model_llags = smf.ols(formula=formula, data=dflagged).fit()
model_llags.summary()
```

[97]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.904
Model:                            OLS   Adj. R-squared:                  0.820
Method:                 Least Squares   F-statistic:                     10.73
Date:                Tue, 12 Apr 2022   Prob (F-statistic):           7.05e-06
Time:                        22:49:32   Log-Likelihood:                -313.70
No. Observations:                  33   AIC:                             659.4
Df Residuals:                      17   BIC:                             683.3
Df Model:                          15
Covariance Type:            nonrobust
================================================================================
========
                               coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--------
```

```
Intercept            -5236.5598   7536.443     -0.695      0.497   -2.11e+04
1.07e+04
C(MonthNumber)[T.2]   8098.5451   7202.451      1.124      0.276   -7097.299
2.33e+04
C(MonthNumber)[T.3]   1.712e+04   8236.001      2.078      0.053    -259.679
3.45e+04
C(MonthNumber)[T.4]   1.473e+04   7197.506      2.046      0.057    -458.258
2.99e+04
C(MonthNumber)[T.5]   4849.6842   5735.396      0.846      0.410   -7250.943
1.7e+04
C(MonthNumber)[T.6]   1.017e+04   6306.049      1.613      0.125   -3132.913
2.35e+04
C(MonthNumber)[T.7]   6846.9345   6580.330      1.041      0.313   -7036.349
2.07e+04
C(MonthNumber)[T.8]   9211.7542   7292.200      1.263      0.224   -6173.443
2.46e+04
C(MonthNumber)[T.9]   3.505e+04   7078.331      4.952      0.000    2.01e+04
5e+04
C(MonthNumber)[T.10]  1.137e+04   6348.473      1.792      0.091   -2020.458
2.48e+04
C(MonthNumber)[T.11]  2.183e+04   6151.236      3.548      0.002    8848.870
3.48e+04
C(MonthNumber)[T.12]   2.66e+04   5715.529      4.653      0.000    1.45e+04
3.87e+04
time                 2202.2761   1595.767      1.380      0.185   -1164.497
5569.049
lagged_sales1           0.2413      0.258      0.934      0.364      -0.304
0.786
lagged_sales2          -0.0413      0.264     -0.157      0.877      -0.599
0.516
lagged_sales3          -0.1516      0.240     -0.631      0.536      -0.658
0.355
==============================================================================
Omnibus:                        4.999   Durbin-Watson:                   2.084
Prob(Omnibus):                  0.082   Jarque-Bera (JB):                5.068
Skew:                           0.181   Prob(JB):                       0.0793
Kurtosis:                       4.885   Cond. No.                     6.14e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 6.14e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[98]:
```
plt.plot(dflagged['date'], dflagged['Sales'], marker='.', color='black')
plt.plot(df['date'], model.fittedvalues, color='b', label='trend + season')
plt.plot(dflagged['date'], model_llags.fittedvalues, color='r', label='trend +␣
↪seasons + 3 lags')
plt.legend()
```

[98]:
```
<matplotlib.legend.Legend at 0x7fbfd8a75910>
```