Chapter 12

Miles A. Peña

DSC 530

02/18/2024

Exercise 12-1: The linear model I used in this chapter has the obvious drawback that it is linear, and there is no reason to expect prices to change linearly over time. We can add flexibility to the model by adding a quadratic term, as we did in "Nonlinear Relationships" on page 133. Use a quadratic model to fit the time series of daily prices, and use the model to generate predictions. You will have to write a version of RunLinearModel that runs that quadratic model, but after that you should be able to reuse code from timeseries.py to generate predictions.

In [8]:
```
from __future__ import print_function, division

import pandas
import numpy as np
import statsmodels.formula.api as smf

import thinkplot
import thinkstats2
import regression
import timeseries
```

In [27]:
```
def RunQuadraticModel(daily):

    daily['years2'] = daily.years**2
    model = smf.ols('ppg ~ years + years2', data = daily)
    results = model.fit()
    return model, results
```

transactions = timeseries.ReadData()

dailies = timeseries.GroupByQualityAndDay(transactions) name = 'high' daily = dailies[name]

model, results = RunQuadraticModel(daily) results.summary()

In [28]:
```
transactions = timeseries.ReadData()

name = 'high'
daily = dailies[name]

model, results = RunQuadraticModel(daily)
results.summary()
```

Out[28]:

OLS Regression Results

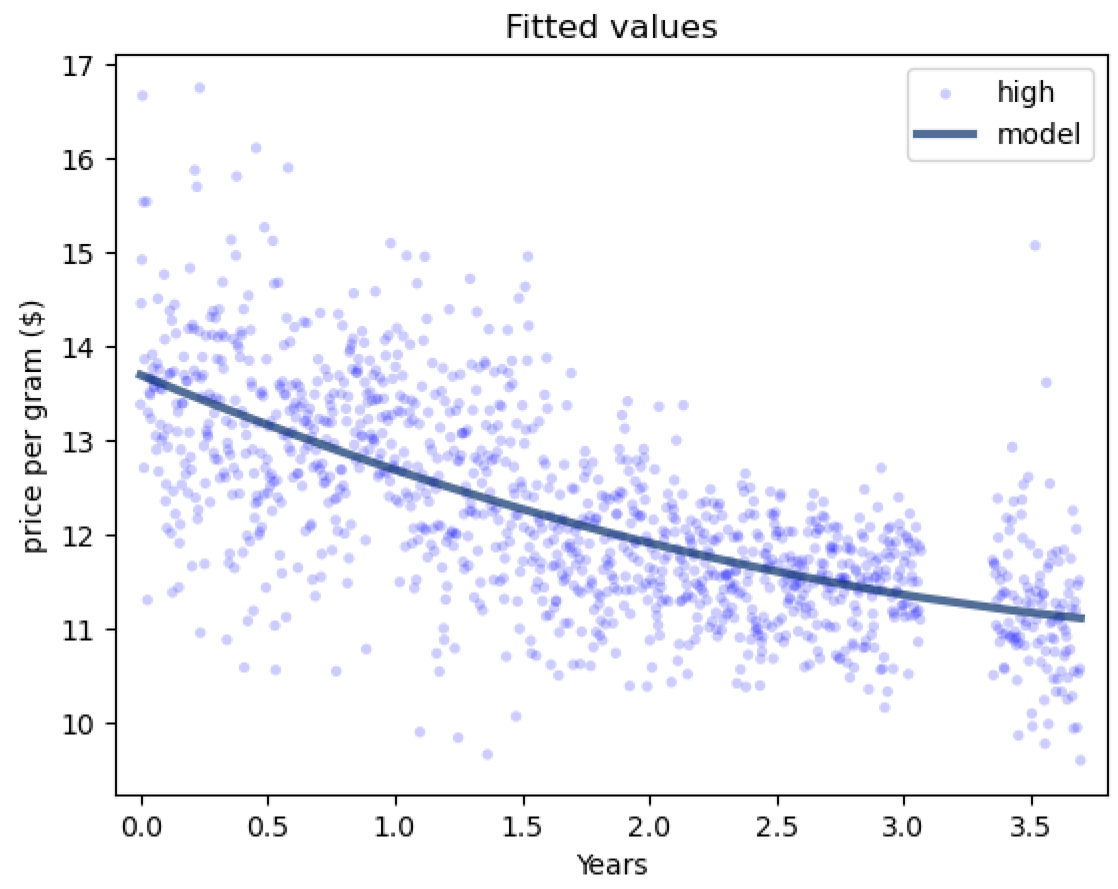
Dep. Variable:	ppg	**R-squared:**	0.455
Model:	OLS	**Adj. R-squared:**	0.454
Method:	Least Squares	**F-statistic:**	517.5
Date:	Sun, 18 Feb 2024	**Prob (F-statistic):**	4.57e-164
Time:	21:06:44	**Log-Likelihood:**	-1497.4
No. Observations:	1241	**AIC:**	3001.
Df Residuals:	1238	**BIC:**	3016.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	13.6980	0.067	205.757	0.000	13.567	13.829
years	-1.1164	0.084	-13.326	0.000	-1.281	-0.952
years2	0.1131	0.022	5.060	0.000	0.069	0.157

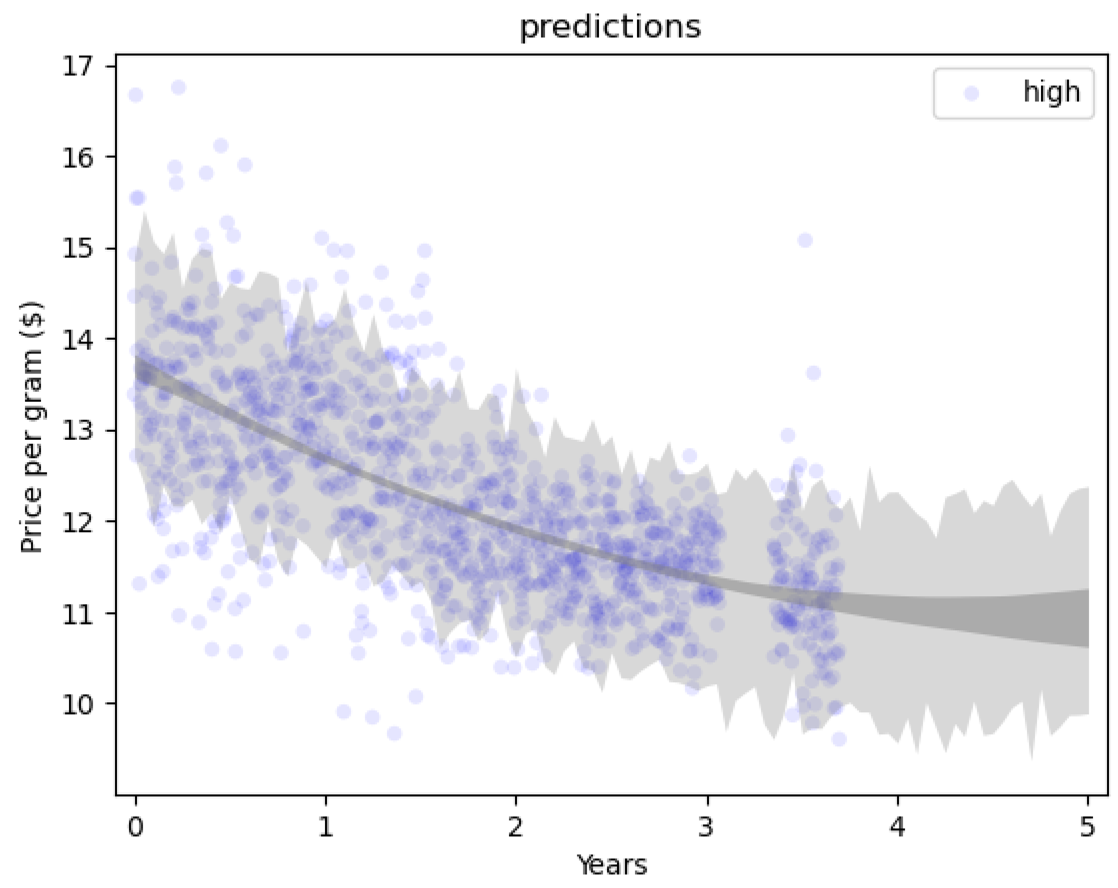
Omnibus:	49.112	**Durbin-Watson:**	1.885
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	113.885
Skew:	0.199	**Prob(JB):**	1.86e-25
Kurtosis:	4.430	**Cond. No.**	27.5

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

In [38]:
```
timeseries.PlotFittedValues(model, results, label=name)
thinkplot.Config(title='Fitted values',
                 xlabel='Years',
                 xlim=[-0.1, 3.8],
                 ylabel='price per gram ($)')
```



In [43]:
```
years = np.linspace(0, 5, 101)
thinkplot.Scatter(daily.years, daily.ppg, alpha=0.1, label=name)
timeseries.PlotPredictions(daily, years, func=RunQuadraticModel)
thinkplot.Config(title='predictions',
                 xlabel='Years',
                 xlim=[years[0]-0.1, years[-1]+0.1],
                 ylabel='Price per gram ($)')
```



Exercise 12-2: Write a definition for a class named SerialCorrelationTest that extends HypothesisTest from "Hypothesis Test" on page 102. It should take a series and a lag as data, compute the serial correlation of the series with the given lag, and then compute the p-value of the observed correlation. Use this class to test whether the serial correlation in raw price data is statistically significant. Also test the residuals of the linear model and (if you did the previous exercise), the quadratic model.

In [51]:
```
class SerialCorrelationTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):

        series, lag = data
        test_stat = abs(thinkstats2.SerialCorr(series, lag))
        return test_stat

    def RunModel(self):

        series, lag = self.data
        permutation = series.reindex(np.random.permutation(series.index))
        return permutation, lag
```

In [54]:
```
# test correlation between consecutive prices

series = daily.ppg
test = SerialCorrelationTest((series, 1))
pvalue = test.PValue()
print(test.actual, pvalue)
```

```
0.4852293761947382 0.0
```

In [56]:
```
# test serial correlation in res of linear mod

_, results = timeseries.RunLinearModel(daily)
series = results.resid
test = SerialCorrelationTest((series, 1))
pvalue = test.PValue()
print(test.actual, pvalue)
```

```
0.07570473767506265 0.009
```

In [57]:
```
# test serial correlation in res of quadratic mod

_, results = RunQuadraticModel(daily)
series = results.resid
test = SerialCorrelationTest((series, 1))
pvalue = test.PValue()
print(test.actual, pvalue)
```

```
0.05607308161289924 0.049
```