10.2 Exercise: Time Series Analysis

Page 161: 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 Section 11.3. 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 the chapter to generate predictions.

In [1]:
```
from os.path import basename, exists


def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve

        local, _ = urlretrieve(url, filename)
        print("Downloaded " + local)


download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkstats2.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkplot.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/mj-clean.csv")
```

In [2]:
```
import numpy as np
import pandas as pd

import random

import thinkstats2
import thinkplot
import statsmodels.formula.api as smf
```

In [3]:
```
transactions = pd.read_csv("mj-clean.csv", parse_dates=[5])
transactions.head()
```

Out[3]:

	city	state	price	amount	quality	date	ppg	state.name	lat	lon
0	Annandale	VA	100	7.075	high	2010-09-02	14.13	Virginia	38.830345	-77.213870
1	Auburn	AL	60	28.300	high	2010-09-02	2.12	Alabama	32.578185	-85.472820
2	Austin	TX	60	28.300	medium	2010-09-02	2.12	Texas	30.326374	-97.771258
3	Belleville	IL	400	28.300	high	2010-09-02	14.13	Illinois	38.532311	-89.983521
4	Boone	NC	55	3.540	high	2010-09-02	15.54	North Carolina	36.217052	-81.687983

The following function takes a DataFrame of transactions and compute daily averages.

In [4]:
```
def GroupByDay(transactions, func=np.mean):
    """Groups transactions by day and compute the daily mean ppg.

    transactions: DataFrame of transactions

    returns: DataFrame of daily prices
    """
    grouped = transactions[["date", "ppg"]].groupby("date")
    daily = grouped.aggregate(func)

    daily["date"] = daily.index
    start = daily.date[0]
    one_year = np.timedelta64(1, "Y")
    daily["years"] = (daily.date - start) / one_year

    return daily
```

The following function returns a map from quality name to a DataFrame of daily averages.

In [5]:
```
def GroupByQualityAndDay(transactions):
    """Divides transactions by quality and computes mean daily price.

    transaction: DataFrame of transactions

    returns: map from quality to time series of ppg
    """
    groups = transactions.groupby("quality")
    dailies = {}
    for name, group in groups:
        dailies[name] = GroupByDay(group)

    return dailies
```

dailies is the map from quality name to DataFrame.

In [6]:
```
dailies = GroupByQualityAndDay(transactions)
```

Rewriting the RunLinearModel into a Quadratic model.

We are using the OLS function to perfrom Ordinary Least Squares (OLS) regression analysis. Our dependent variable 'ppg' is being regressed against our two independent variables 'years' and 'years2'. The data is being pulled from our 'daily' dataset.

In [7]:
```
def RunQuadraticModel(daily):
    daily['years2'] = daily.years**2
    my_model = smf.ols("ppg ~ years + years2", data=daily)
    results = my_model.fit()
    return my_model, results
```

Run the quadratic model

In [8]:
```
name = "high"
daily = dailies[name]

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

Out[8]:

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:	Thu, 28 Nov 2024	**Prob (F-statistic):**	4.57e-164
Time:	11:13:16	**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.

We need to generate predictions and we want to qantify the uncertainty in the prediction. We can do that by resampling. The following function fits a model to the data, computes residuals, then resamples from the residuals to general fake datasets. It fits the same model to each fake dataset and returns a list of results.

In [9]:

```
def SimulateResults(daily, iters=101, func=RunQuadraticModel):
    """Run simulations based on resampling residuals.

    daily: DataFrame of daily prices
    iters: number of simulations
    func: function that fits a model to the data

    returns: list of result objects
    """
    _, results = func(daily)
    fake = daily.copy()

    result_seq = []
```

```
    for _ in range(iters):
        fake.ppg = results.fittedvalues + thinkstats2.Resample(results.resid)
        _, fake_results = func(fake)
        result_seq.append(fake_results)

    return result_seq
```

To generate predictions, we take the list of results fitted to resampled data. For each model, we use the predict method to generate predictions, and return a sequence of predictions.

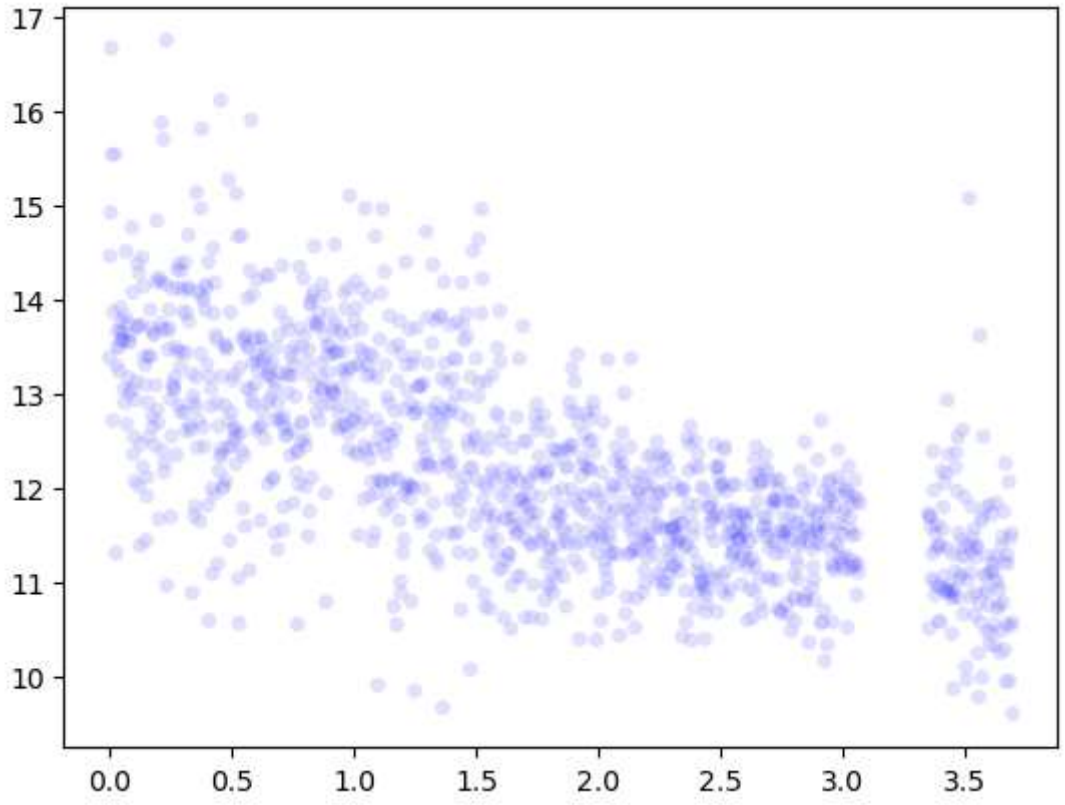
If add_resid is true, we add resampled residuals to the predicted values, which generates predictions that include predictive uncertainty (due to random noise) as well as modeling uncertainty (due to random sampling).

In [10]:
```
def GeneratePredictions(result_seq, years, add_resid=False):
    """Generates an array of predicted values from a list of model results.

    When add_resid is False, predictions represent sampling error only.

    When add_resid is True, they also include residual error (which is
    more relevant to prediction).

    result_seq: list of model results
    years: sequence of times (in years) to make predictions for
    add_resid: boolean, whether to add in resampled residuals

    returns: sequence of predictions
    """
    n = len(years)
    d = dict(Intercept=np.ones(n), years=years, years2=years**2)
    predict_df = pd.DataFrame(d)

    predict_seq = []
    for fake_results in result_seq:
        predict = fake_results.predict(predict_df)
        if add_resid:
            predict += thinkstats2.Resample(fake_results.resid, n)
        predict_seq.append(predict)

    return predict_seq
```

Now we visualize the redone predictions. The darker regions quantifies the modeling uncertainty and the lighter regions quantifies predictive uncertainty

In [11]:
```
years = np.linspace(0, 5, 101)
thinkplot.Scatter(daily.years, daily.ppg, alpha=0.1, label=name)
PlotPredictions(daily, years, func=RunQuadraticModel)
xlim = years[0] - 0.1, years[-1] + 0.1
thinkplot.Config(
    title="Predictions", xlabel="Years", xlim=xlim, ylabel="Price per gram ($)"
)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[11], line 3
      1 years = np.linspace(0, 5, 101)
      2 thinkplot.Scatter(daily.years, daily.ppg, alpha=0.1, label=name)
----> 3 PlotPredictions(daily, years, func=RunQuadraticModel)
      4 xlim = years[0] - 0.1, years[-1] + 0.1
      5 thinkplot.Config(
      6     title="Predictions", xlabel="Years", xlim=xlim, ylabel="Price per gram
($)"
      7 )

NameError: name 'PlotPredictions' is not defined
```



Page 161: 12-2 Write a definition for a class named SerialCorrelationTest that extends HypothesisTest from Section 9.2. 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 [ ]:
```
class SerialCorrelationTest(thinkstats2.HypothesisTest):
    def TestStatistic(self, data):
        series, lag = data
        # Now we compute the serial correlation of the series with the given lag
        # We can use the serial correlation function
        corr = abs(SerialCorr(series, lag))
        return corr
    def RunModel(self):
        series, lag = self.data
        # Now we generate a random permutation of the series
        permutation = series.reindex(np.random.permutation(series.index))
        return permutation, lag
```

A function to compute serial correlation with the given lag.

In [ ]:
```
def SerialCorr(series, lag=1):
    xs = series[lag:]
    ys = series.shift(lag)[lag:]
    corr = thinkstats2.Corr(xs, ys)
    return corr
```

We are going to test the serial correlation in raw price data (ppg) to see if it is statistically significant. We must test the correlation between consecutive prices.

In [ ]:
```
name = "high"
daily = dailies[name]

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

We also test the residuals of the linear model.

In [ ]:
```
def RunLinearModel(daily):
    model = smf.ols("ppg ~ years", data=daily)
    results = model.fit()
    return model, results
```

In [ ]:
```
_, results = RunLinearModel(daily)
series = results.resid
test = SerialCorrelationTest((series, 1))
pvalue = test.PValue()
print(test.actual, pvalue)
```

Now we test the residuals of the quadratic model.

In [ ]:
```
_, results = RunQuadraticModel(daily)
series = results.resid
test = SerialCorrelationTest((series, 1))
```

```
pvalue = test.PValue()
print(test.actual, pvalue)
```

Comparing the results of the serial correlation test between the residuals of the linear model and the residuals of the quadratic model we see the following:

For the Linear Model:

Observed serial correlation: Apprx. 0.076

P-value: Approx. 0.008

For the Quadratic Model:

Observed serial correlation: Apprx. 0.056

P-value: Approx. 0.044

Based on these results the linear model's residuals show a higher level of serial correlation compared to the quadratic model's residuals, and its serial correlation is statistically significant at a lower p-value threshold. In other words the linear model's leftover patterns seem to be more strongly connected to each other over time compared to the quadratice model's residuals, and the lower p-value indicates stronger evidence against the idea that the patterns we see in the residuals are just due to random chance.