Assignment 10.2

Exercises 12 - 1 and 12 - 2

http://thinkstats2.com

In [1]:
```
# Imports
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

import thinkstats2
import thinkplot
```

In [2]:
```
from IPython.core.display import HTML
table_css = 'table {align:left;display:block} '
HTML('<style>{}</style>'.format(table_css))
```

Out[2]:

Exercise 12 - 1

The linear model I used in this chapter has the obvious drawback that it is linear, and there is no to expect prices to change linearly over time. We can add flexibility to the model by adding a qua 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 predic You will have to write a version of RunLinearModel that runs that quadratic model, but after that y should be able to reuse code from the chapter to generate predictions.

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

Out[3]:

	city	state	price	amount	quality	date	ppg	state.name	lat	
0	Annandale	VA	100	7.075	high	2010-09-02	14.13	Virginia	38.830345	-7
1	Auburn	AL	60	28.300	high	2010-09-02	2.12	Alabama	32.578185	-8
2	Austin	TX	60	28.300	medium	2010-09-02	2.12	Texas	30.326374	-9
3	Belleville	IL	400	28.300	high	2010-09-02	14.13	Illinois	38.532311	-8
4	Boone	NC	55	3.540	high	2010-09-02	15.54	North Carolina	36.217052	-8

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

    args:
        transactions (DataFrame): transactions

    returns:
        daily (DataFrame): 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
```

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

    args:
        transaction (DataFrame): transactions

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

    return dailies
```

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

In [38]:
```python
dailies = GroupByQualityAndDay(transactions)

name = 'high'
daily = dailies[name]
```

In [39]:
```python
def RunQuadraticModel(daily):
    daily['years2'] = daily.years**2
    model = smf.ols("ppg ~ years + years2", data=daily)
    results = model.fit()

    return model, results
```

In [40]:
```python
model, results = RunQuadraticModel(daily)
display(results.summary())
```

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:	Tue, 17 May 2022	**Prob (F-statistic):**	4.57e-164
Time:	17:10:26	**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	-1.1171	0.084	-13.326	0.000	-1.282	-0.953
years2	0.1132	0.022	5.060	0.000	0.069	0.157

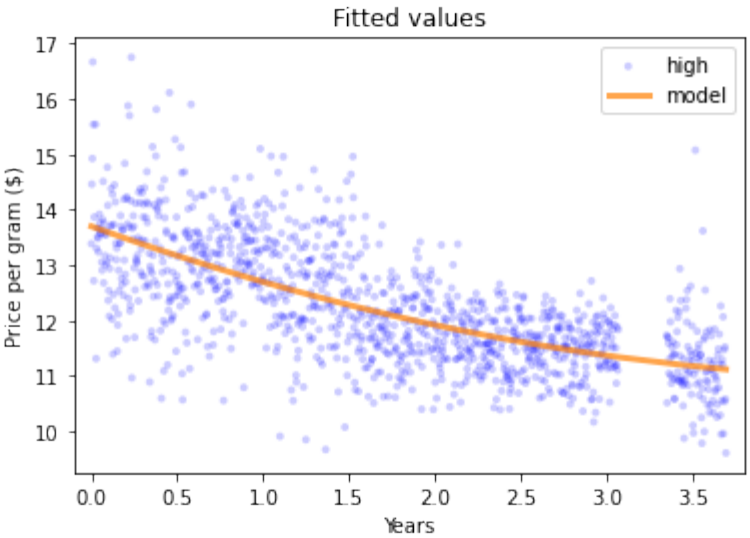
Omnibus:	49.112	**Durbin-Watson:**	1.885
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	113.885
Skew:	0.199	**Prob(JB):**	1.86e-25
Kurtosis:	4.430	**Cond. No.**	27.5

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

In [9]:
```
def PlotFittedValues(model, results, label=""):
    """
    Plots original data and fitted values.

    args:
        model (object): StatsModel model object
        results (obejct): StatsModel results object

    returns:
        None
    """
    years = model.exog[:, 1]
    values = model.endog
    thinkplot.Scatter(years, values, s=15, label=label)
    thinkplot.Plot(years, results.fittedvalues, label="model", color="#ff
```

In [12]:
```python
def PlotLinearModel(daily, name):
    """
    Plots a linear fit to a sequence of prices, and the residuals.

    args:
        daily (DataFrame): daily prices
        name (string): label name

    returns:
        None
    """
    model, results = RunQuadraticModel(daily)
    PlotFittedValues(model, results, label=name)
    thinkplot.Config(
        title="Fitted values",
        xlabel="Years",
        xlim=[-0.1, 3.8],
        ylabel="Price per gram ($)",
    )
```

In [13]:
```python
name = "high"
daily = dailies[name]

PlotLinearModel(daily, name)
```

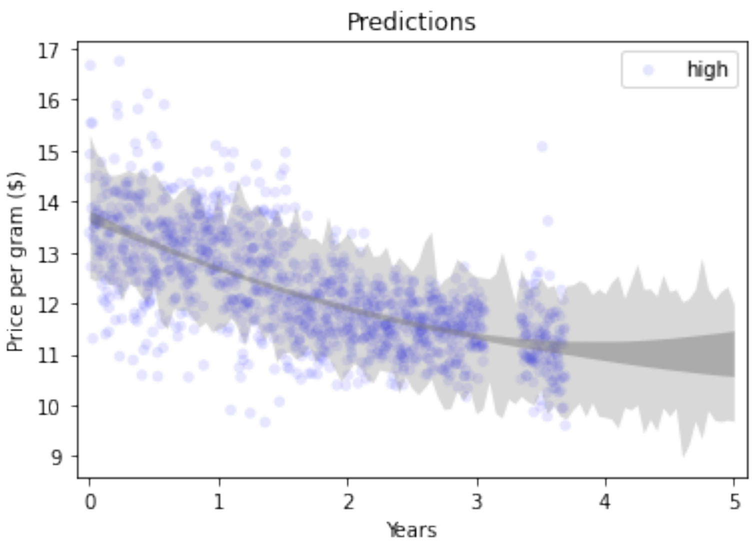


In [15]:
```
def SimulateResults(daily, iters=101, func = RunQuadraticModel):
    """
    Run simulations based on resampling residuals.

    args:
        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.re
        _, fake_results = func(fake)
        result_seq.append(fake_results)

    return result_seq
```

In [16]:
```
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).

    args:
        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
```

In [18]:
```
def PlotPredictions(daily, years, iters=101, percent=90, func = RunQuadra
    """
    Plots predictions.

    args:
        daily: DataFrame of daily prices
        years: sequence of times (in years) to make predictions for
        iters: number of simulations
        percent: what percentile range to show
        func: function that fits a model to the data

    returns:
        None
    """
    result_seq = SimulateResults(daily, iters=iters, func=func)
    p = (100 - percent) / 2
    percents = p, 100 - p

    predict_seq = GeneratePredictions(result_seq, years, add_resid=True)
    low, high = thinkstats2.PercentileRows(predict_seq, percents)
    thinkplot.FillBetween(years, low, high, alpha=0.3, color="gray")

    predict_seq = GeneratePredictions(result_seq, years, add_resid=False)
    low, high = thinkstats2.PercentileRows(predict_seq, percents)
    thinkplot.FillBetween(years, low, high, alpha=0.5, color="gray")
```

In [19]:
```
years = np.linspace(0, 5, 101)
thinkplot.Scatter(daily.years, daily.ppg, alpha=0.1, label=name)
PlotPredictions(daily, years)
xlim = years[0] - 0.1, years[-1] + 0.1
thinkplot.Config(
    title="Predictions", xlabel="Years", xlim=xlim, ylabel="Price per grai
)
```



Exercise 12 - 2

Write a definition for a class named SerialCorrelationTest that extends HypothesisTest from Sect It should take a series and a lag as data, compute the serial correlation of the series with the give 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. Als the residuals of the linear model and (if you did the previous exercise), the quadratic model.

In [26]:
```
class HypothesisTest(object):

    def __init__(self, data):
        self.data = data
        self.MakeModel()
        self.actual = self.TestStatistic(data)

    def PValue(self, iters=1000):
        self.test_stats = [self.TestStatistic(self.RunModel())
                           for _ in range(iters)]

        count = sum(1 for x in self.test_stats if x >= self.actual)
        return count / iters

    def TestStatistic(self, data):
        raise UnimplementedMethodException()

    def MakeModel(self):
        pass

    def RunModel(self):
        raise UnimplementedMethodException()
```

In [29]:
```
def SerialCorr(series, lag = 1):
    xs = series[lag:]
    ys = series.shift(lag)[lag:]

    corr = thinkstats2.Corr(xs, ys)

    return corr
```

In [30]:
```
class SerialCorrelationTest(HypothesisTest):
    def TestStatistic(self, data):
        series, lag = data
        test_stat = abs(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 [41]:
```
name = "high"
daily = dailies[name]

series = daily.ppg
test = SerialCorrelationTest((series, 1))
pvalue = test.PValue()
print(f"Correlaton: {test.actual}")
print(f"P-Value: {pvalue}")
```

```
Correlaton: 0.4852293761947381
P-Value: 0.0
```

There is a 0.49 correlation between consecutive prices and the p-value is close to 0 indicating th significant.

In [42]:
```
# test for serial correlation in residuals of the linear model

_, results = RunLinearModel(daily)
series = results.resid
test = SerialCorrelationTest((series, 1))
pvalue = test.PValue()
print(f"Residuals Correlaton: {test.actual}")
print(f"P-Value: {pvalue}")
```

```
Correlaton: 0.07570473767506262
P-Value: 0.009
```

The residuals have a correlation of 0.08 and the p-value is 0.01 which is significant

In [43]:
```
# test for serial correlation in residuals of the quadratic model

_, results = RunQuadraticModel(daily)
series = results.resid
test = SerialCorrelationTest((series, 1))
pvalue = test.PValue()
print(f"Correlaton: {test.actual}")
print(f"P-Value: {pvalue}")
```

```
Correlaton: 0.05607308161289913
P-Value: 0.051
```

The residuals in a quadratic model have a correlation of 0.06 and a p-value which is significant.