Theodore Koby-Hercsky
DSC530-T302

# 9.2 Exercises

## Exercises 12.1-2

**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 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 [42]:
```
# First I went back to look over chapter 11.3 that used a quadratic
# relationship for age and weight from the previous week. Then I looked
# over the RunLinearModel again that uses the statsmodels to runs a
# linear model of price as a function of time to get a better
# understanding of what I wanted to do. Then I created a function that
# went off the two examples from last week and this week and used the
# prices data frame for the daily that will return the model and the
# results like the example above from this week.

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

In [43]:
```
# Next I went of the example above that found the results for the high
# quality category then displayed the results
name = 'high'
daily = dailies[name]

model, results = RunQuadModel(daily)
results.summary()
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:** | Fri, 06 Aug 2021 | **Prob (F-statistic):** | 4.57e-164 |
| **Time:** | 17:44:03 | **Log-Likelihood:** | -1497.4 |
| **No. Observations:** | 1241 | **AIC:** | 3001. |
| **Df Residuals:** | 1238 | **BIC:** | 3016. |
| **Df Model:** | 2 | | |
| **Covariance Type:** | nonrobust | | |

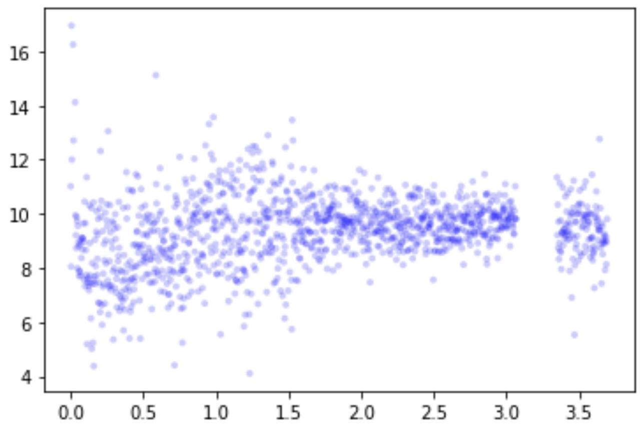
| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **Intercept** | 13.6980 | 0.067 | 205.757 | 0.000 | 13.567 | 13.829 |
| **years** | -1.1171 | 0.084 | -13.326 | 0.000 | -1.282 | -0.953 |
| **years2** | 0.1132 | 0.022 | 5.060 | 0.000 | 0.069 | 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 |

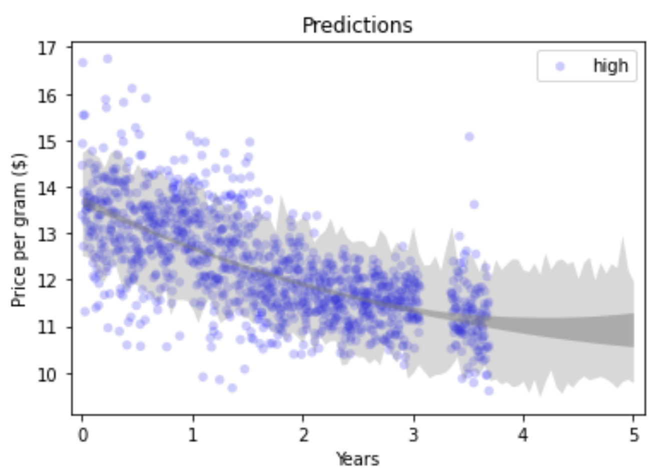
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [64]:
```python
# Next I went off the Linear Model function that plots the original
# data and the fitted curve

def PlotQuadModel(daily, name):
    model, results = RunQuadModel(daily)
PlotFittedValues(model, results, label=name)
thinkplot.Config(title='Fitted values', xlabel='Years',
                 xlim=[-0.1, 3.8], ylabel='price per gram ($)')
PlotLinearModel(daily, name)
```



In [115]:
```python
# Next I kinda went off the high quality category thinkplot to set my
# years and scatter plot to show the predictions for high for price per
# gram.
years = np.linspace(0, 5, 101)
thinkplot.Scatter(daily.years, daily.ppg, alpha=0.2, label=name)
PlotPredictions(daily, years, func=RunQuadModel)
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 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 [89]:
```
# First I will create a class definition that is named SerialCorrelation
# That works off the HypothesisTest from Section 9.2 by going back and
# looking over that assignmentthen I will compute the serial correlation
# of the series with the given lag and went off the permutation test fro
# chapter 9 tobuild these functions
class SerialCorrelationTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        series, lag = data
        series_data = abs(SerialCorr(series, lag))
        return series_data

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

In [105]:
```
# Next I will use the daily highs from this weeks assignment to
# test for correlation by running the SerialCorrelationTest from
# daily.ppg and also the p-value of the observed correlation.

correl = SerialCorrelationTest((daily.ppg, 1))
print("Pvalue:", "actual:" )
print( correl.PValue(), correl.actual)
```

```
Pvalue: actual:
0.0 0.485229376194738
```

In [113]:
```
# I used the layout seen above for prediction for the high quality
# category to help form my code to find the residuals for the linear
# model to find a serial correlation by setting the RunLinearModel
# to daily and serial data to use the SerialCorrelationTest and using
# the results.resid for the pvalue and actual

_, results = RunLinearModel(daily)
serial_data = SerialCorrelationTest((results.resid, 1))
print("Pvalue:", "actual:" )
print(serial_data.PValue(), serial_data.actual)
```

```
Pvalue: actual:
0.009 0.07570473767506254
```

In [114]:

```
# I used the layout seen above for prediction for the high quality
# category to help form my code to find the residuals for the linear
# model to find a serial correlation by setting the RunQuadModel
# to daily and serial data to use the SerialCorrelationTest and using
# the results.resid for the pvalue and actual

_, results = RunQuadModel(daily)
serial_data = SerialCorrelationTest((results.resid, 1))
print("Pvalue:", "actual:" )
print(serial_data.PValue(), serial_data.actual)
```

```
Pvalue: actual:
0.043 0.056073081612899096
```