Bilenkin530Week10_Exercises_10.2

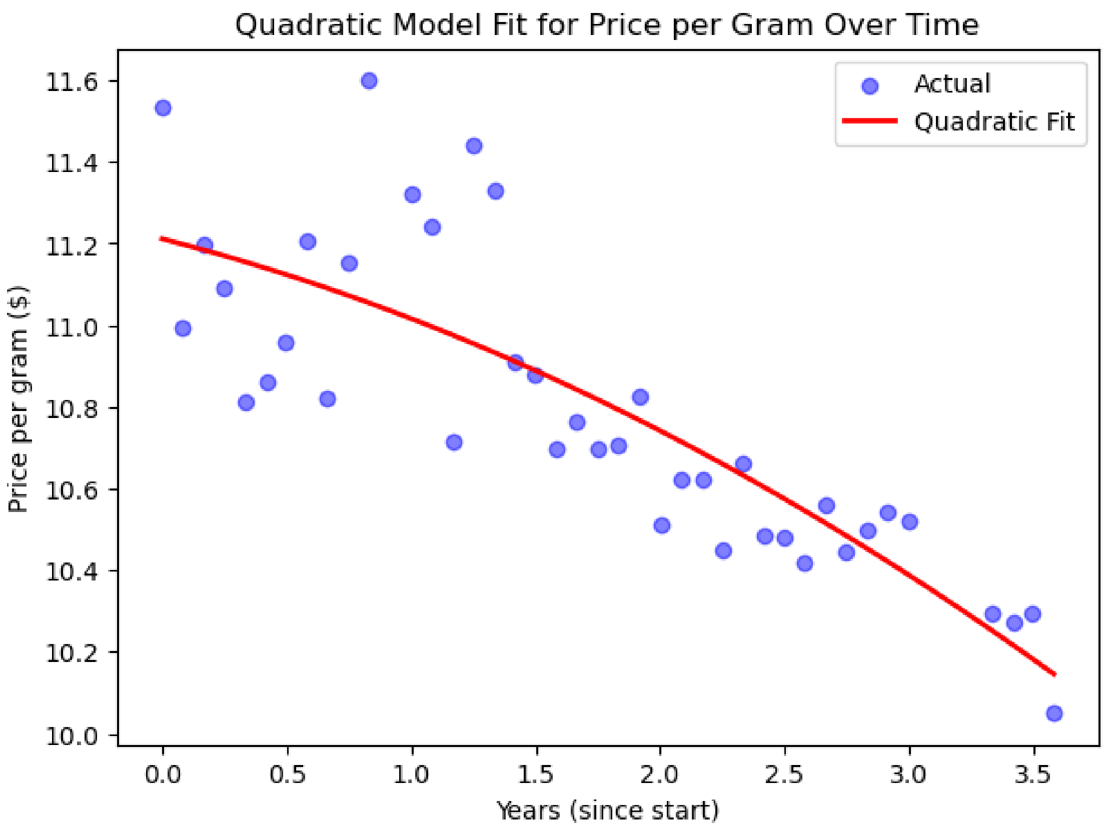
February 16, 2025

0.0.1 Chapter 12 (page 161. Exercise 12-1)

Use a quadratic model to fit the time series of daily prices, and use the model to generate predictions.

[19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Loading the dataset from GitHub
file_url = "https://github.com/AllenDowney/ThinkStats2/raw/master/code/mj-clean.
↪csv"
df = pd.read_csv(file_url)

# Converting 'date' to datetime
df['date'] = pd.to_datetime(df['date'], errors='coerce')

# Filtering data to keep only recent years
df = df[df['date'].dt.year >= 2010]

# Aggregating prices by month to smooth the trend
df['YearMonth'] = df['date'].dt.to_period('M')
df = df.groupby('YearMonth', as_index=False)['ppg'].mean()

# Converting 'YearMonth' to datetime using .to_timestamp()
df['YearMonth'] = df['YearMonth'].dt.to_timestamp()

# Converting 'YearMonth' to fractional years
df['Years'] = df['YearMonth'].dt.year + (df['YearMonth'].dt.dayofyear / 365.25)

# Shifting to start from zero
df['Years'] -= df['Years'].min()

# Removing extreme outliers
df = df[(df['ppg'] > df['ppg'].quantile(0.01)) & (df['ppg'] < df['ppg'].
↪quantile(0.99))]

# Defining the function for fitting a quadratic model and generating predictions
```

```python
def RunQuadraticModel(df):
    # Prepare data for quadratic regression
    x = df['Years']
    y = df['ppg']
    X = np.column_stack((x, x**2))
    X = sm.add_constant(X)

    # Fitting the quadratic model
    model = sm.OLS(y, X).fit()

    # Generating predictions
    x_pred = np.linspace(x.min(), x.max(), 300)
    X_pred = np.column_stack((x_pred, x_pred**2))
    X_pred = sm.add_constant(X_pred)
    y_pred = model.predict(X_pred)

    return model, x_pred, y_pred

# Calling the function with the dataframe
model, x_pred, y_pred = RunQuadraticModel(df)

# Plotting the results
plt.figure(figsize=(7, 5))
plt.scatter(df['Years'], df['ppg'], color='blue', alpha=0.5, label='Actual')
plt.plot(x_pred, y_pred, color='red', linewidth=2, label='Quadratic Fit')
plt.xlabel("Years (since start)")
plt.ylabel("Price per gram ($)")
plt.title("Quadratic Model Fit for Price per Gram Over Time")
plt.legend()
plt.show()

# Printing model summary
print(model.summary())
```




```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ppg   R-squared:                       0.694
Model:                            OLS   Adj. R-squared:                  0.677
Method:                 Least Squares   F-statistic:                     41.96
Date:                Sun, 16 Feb 2025   Prob (F-statistic):           3.06e-10
Time:                        03:00:16   Log-Likelihood:                 6.9812
No. Observations:                  40   AIC:                            -7.962
Df Residuals:                      37   BIC:                            -2.896
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         11.2113      0.092    122.139      0.000      11.025      11.397
x1            -0.1556      0.119     -1.307      0.199      -0.397       0.086
x2            -0.0397      0.033     -1.210      0.234      -0.106       0.027
==============================================================================
Omnibus:                        5.947   Durbin-Watson:                   1.434
Prob(Omnibus):                  0.051   Jarque-Bera (JB):                4.828
Skew:                           0.827   Prob(JB):                       0.0894
```

```
Kurtosis:                       3.403   Cond. No.                        25.9
==============================================================================

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

Chapter 12 (page 161. Exercise 12-2)

Write a defition for a class named SerialCorrelationTest that extends HypothesisTest from Section 9.2.

[20]:
```python
class HypothesisTest:
    def __init__(self, series):

        self.series = series

    def test_statistic(self):

        raise NotImplementedError("Subclasses should implement this method.")

    def p_value(self):

        raise NotImplementedError("Subclasses should implement this method.")
```

[21]:
```python
# Creating a dummy pandas series to test
series_data = pd.Series([1, 2, 3, 4, 5])

# Instantiating the HypothesisTest class
test = HypothesisTest(series_data)

# Printing the series attribute to confirm it's correctly stored
print(test.series)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

Write a definition for a class named SerialCorrelationTest

[22]:
```python
import numpy as np
import scipy.stats as stats

class SerialCorrelationTest(HypothesisTest):
    def __init__(self, series, lag):
        super().__init__(series)
        self.lag = lag
```

```
    def serial_correlation(self):
        series_lagged = self.series.shift(self.lag)
        correlation = self.series.corr(series_lagged)
        return correlation

    def p_value(self):
        correlation = self.serial_correlation()
        n = len(self.series)
        # Test statistic for serial correlation
        t_statistic = correlation * np.sqrt(n - self.lag - 1) / np.sqrt(1 -␣
↪correlation**2)
        p_value = 2 * (1 - stats.t.cdf(abs(t_statistic), df=n - 2))
        return p_value
```

Compute the serial correlation of the series with the given lag, and then compute the p-value of the observed correlation.

[23]:
```
# Testing the raw price data with a lag of 1
raw_data_test = SerialCorrelationTest(df['ppg'], lag=1)
print("Serial correlation for raw price data:", raw_data_test.
 ↪serial_correlation())
print("p-value for raw price data:", raw_data_test.p_value())

# Testing the residuals of the linear model with a lag of 1
residuals_linear = model.resid
linear_model_test = SerialCorrelationTest(residuals_linear, lag=1)
print("\nSerial correlation for linear model residuals:", linear_model_test.
 ↪serial_correlation())
print("p-value for linear model residuals:", linear_model_test.p_value())

# Testing the residuals of the quadratic model with a lag of 1 (if applicable)
residuals_quadratic = model.resid
quadratic_model_test = SerialCorrelationTest(residuals_quadratic, lag=1)
print("\nSerial correlation for quadratic model residuals:",␣
 ↪quadratic_model_test.serial_correlation())
print("p-value for quadratic model residuals:", quadratic_model_test.p_value())
```

```
Serial correlation for raw price data: 0.7515547672236665
p-value for raw price data: 2.2890221051952153e-08

Serial correlation for linear model residuals: 0.2584706824756061
p-value for linear model residuals: 0.10731621024129523

Serial correlation for quadratic model residuals: 0.2584706824756061
p-value for quadratic model residuals: 0.10731621024129523
```