DSC530-302 Data Exploration and Analysis

Author: Chitramoy Mukherjee

Date: 05/18/2023

Title: "DSC530-302 Week-05 Assignment- 112.1 and 12.2"

In [ ]:
```
Exercise 12.1 : The linear model I used in this chapter has the obvious drawback that
to expect prices to change linearly over time. We can add flexibility to the model by
Section 11.3. Use a quadratic model to fit the time series of daily prices, and use th
You will have to write a version of RunLinearModel that runs that quadratic model, but
to reuse code from the chapter to generate predictions.
```

In [16]:
```
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")

import numpy as np
import pandas as pd

import random

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

In [6]:
```
```

In [17]:
```
transactions = pd.read_csv("mj-clean.csv", parse_dates=[5])
transactions.head()

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

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 = GroupByQualityAndDay(transactions)

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

name = "high"
daily = dailies[name]

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

Out[17]:

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:	Sat, 20 May 2023	**Prob (F-statistic):**	4.57e-164
Time:	07:40:01	**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

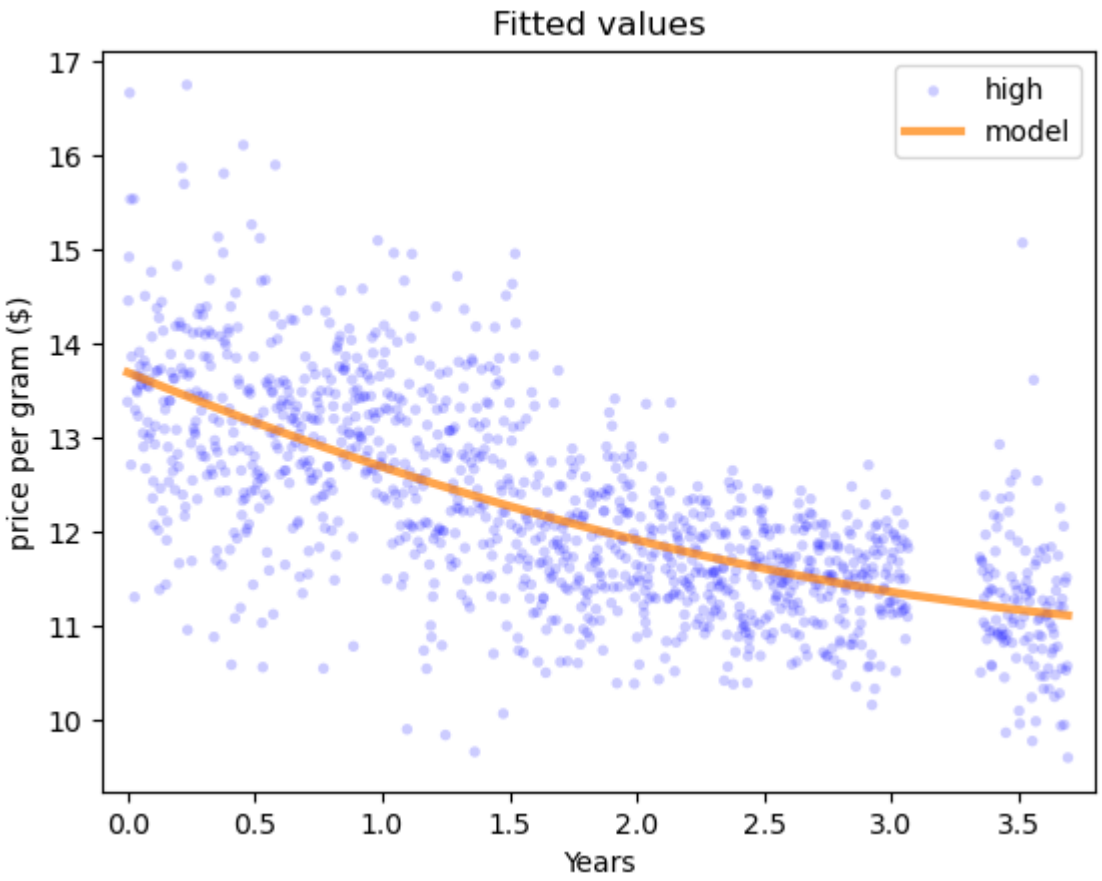
Omnibus:	49.112	**Durbin-Watson:**	1.885
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	113.885
Skew:	0.199	**Prob(JB):**	1.86e-25
Kurtosis:	4.430	**Cond. No.**	27.5

Notes:

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

In [19]:

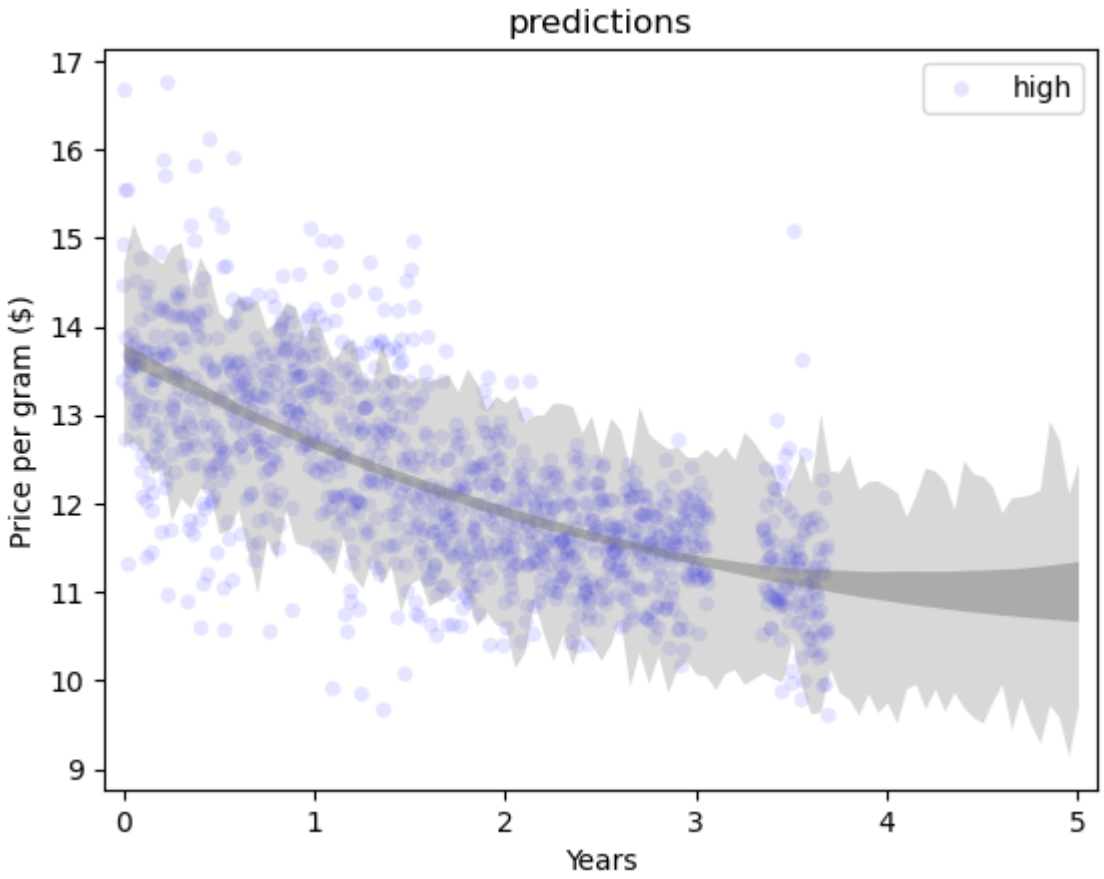
```
def PlotFittedValues(model, results, label=""):
    """Plots original data and fitted values.

    model: StatsModel model object
    results: StatsModel results object
    """
    years = model.exog[:, 1]
    values = model.endog
    thinkplot.Scatter(years, values, s=15, label=label)
    thinkplot.Plot(years, results.fittedvalues, label="model", color="#ff7f00")

# Solution

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



In [24]:

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

def SimulateResults(daily, iters=101, func=RunLinearModel):
    """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

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

def PlotPredictions(daily, years, iters=101, percent=90, func=RunLinearModel):
    """Plots predictions.

    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
    """
    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")

# Solution

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


predictions
high
Price per gram ($)
Years
17
16
15
14
13
12
11
10
9
0
1
2
3
4
5