DSC 530 Data Exploration and Analysis

Assignment Week 10_ Excercises: 12.1 & 12.2

Author: Zemelak Goraga

Data: 2/18/2024

In [62]:
```
from os.path import basename, exists
import numpy as np
import pandas as pd
import thinkstats2
import thinkplot
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import statsmodels.tsa.stattools as smtsa
import random
```

In [63]:
```
# Function to download files if not already present
def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve
        local, _ = urlretrieve(url, filename)
        print("Downloaded " + local)
```

In [64]:
```
# Download necessary input files
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 [65]:
```
# Load the data from "Price of Weed"
transactions = pd.read_csv("mj-clean.csv", parse_dates=[5])
```

In [66]:
```
transactions.head()
```

Out[66]:

	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

In [67]:
```
# Function to group transactions by day and compute daily mean ppg
def GroupByDay(transactions, func=np.mean):
    """Groups transactions by day and compute the daily mean ppg."""
    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 [68]:
```
# Function to divide transactions by quality and compute mean daily price
def GroupByQualityAndDay(transactions):
    """Divides transactions by quality and computes mean daily price."""
    groups = transactions.groupby("quality")
    dailies = {}
    for name, group in groups:
        dailies[name] = GroupByDay(group)

    return dailies
```

In [69]:
```
# Load the data and group by quality and day
dailies = GroupByQualityAndDay(transactions)
```

In [ ]:
```
```

Excercise 12.1

The linear model I used in this chapter has the obvious draw- back that it is linear, and there is no reason to expect prices to change linearly over time. We can add exibility to the model by adding a quadratic term. 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 in timeseries.py to generate predictions.

In [70]:
```
# Task 1: Fit a quadratic model and generate predictions

def RunQuadraticModel(daily):
    """Runs a quadratic model."""
    model = smf.ols("ppg ~ years + np.power(years, 2)", data=daily)
    results = model.fit()
    return model, results
```

In [71]:
```
def GenerateQuadraticPredictions(results, years):
    """Generates predictions using a quadratic model."""
    n = len(years)
    d = dict(Intercept=np.ones(n), years=years, years2=np.power(years, 2))
    predict_df = pd.DataFrame(d)
    predict = results.predict(predict_df)
    return predict
```

In [72]:
```
# Fit quadratic model
quadratic_model, quadratic_results = RunQuadraticModel(daily)
print("Quadratic model fitting results:")
print(quadratic_results.summary())
```

```
Quadratic model fitting results:
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ppg   R-squared:                       0.085
Model:                            OLS   Adj. R-squared:                  0.083
Method:                 Least Squares   F-statistic:                     57.33
Date:                Sun, 18 Feb 2024   Prob (F-statistic):           1.55e-24
Time:                        14:42:47   Log-Likelihood:                -2030.6
No. Observations:                1238   AIC:                             4067.
Df Residuals:                    1235   BIC:                             4083.
Df Model:                           2
Covariance Type:            nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
Intercept                  8.3509      0.104     80.512      0.000       8.147       8.554
years                      1.1472      0.130      8.806      0.000       0.892       1.403
np.power(years, 2)        -0.2386      0.035     -6.878      0.000      -0.307      -0.171
==============================================================================
Omnibus:                      194.417   Durbin-Watson:                   1.835
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1123.295
Skew:                           0.585   Prob(JB):                    1.20e-244
Kurtosis:                       7.517   Cond. No.                         27.7
==============================================================================

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

In [73]:
```python
# Generate predictions using quadratic model
start = daily.index.min()
end = daily.index.max()
years = np.linspace(start.year, end.year, end.year - start.year + 1)
quadratic_predictions = GenerateQuadraticPredictions(quadratic_results, years)
print("Quadratic model predictions:")
print(quadratic_predictions)
```

```
Quadratic model predictions:
0   -961744.779517
1   -962703.133592
2   -963661.964912
3   -964621.273477
4   -965581.059288
dtype: float64
```

In [74]:
```python
# Parameters for quadratic model

# Extracting parameters from the quadratic model results
intercept = quadratic_results.params['Intercept']
coef_years = quadratic_results.params['years']
coef_years2 = quadratic_results.params['np.power(years, 2)']

# Extracting standard errors
std_err_intercept = quadratic_results.bse['Intercept']
std_err_years = quadratic_results.bse['years']
std_err_years2 = quadratic_results.bse['np.power(years, 2)']

# Extracting t-values
t_value_intercept = quadratic_results.tvalues['Intercept']
t_value_years = quadratic_results.tvalues['years']
t_value_years2 = quadratic_results.tvalues['np.power(years, 2)']

# Extracting p-values
p_value_intercept = quadratic_results.pvalues['Intercept']
p_value_years = quadratic_results.pvalues['years']
p_value_years2 = quadratic_results.pvalues['np.power(years, 2)']

# Extracting confidence intervals
```

```
conf_int_intercept = quadratic_results.conf_int().loc['Intercept']
conf_int_years = quadratic_results.conf_int().loc['years']
conf_int_years2 = quadratic_results.conf_int().loc['np.power(years, 2)']

# Print the extracted parameters
print("Intercept:", intercept)
print("Coefficient for years:", coef_years)
print("Coefficient for years^2:", coef_years2)
print("Standard error for Intercept:", std_err_intercept)
print("Standard error for years:", std_err_years)
print("Standard error for years^2:", std_err_years2)
print("t-value for Intercept:", t_value_intercept)
print("t-value for years:", t_value_years)
print("t-value for years^2:", t_value_years2)
print("p-value for Intercept:", p_value_intercept)
print("p-value for years:", p_value_years)
print("p-value for years^2:", p_value_years2)
print("Confidence interval for Intercept:", conf_int_intercept)
print("Confidence interval for years:", conf_int_years)
print("Confidence interval for years^2:", conf_int_years2)
```

```
Intercept: 8.350930280586141
Coefficient for years: 1.1471634615853479
Coefficient for years^2: -0.23862254127502103
Standard error for Intercept: 0.10372239673383403
Standard error for years: 0.13027337102841077
Standard error for years^2: 0.034695445018808946
t-value for Intercept: 80.51231502117888
t-value for years: 8.805816971874995
t-value for years^2: -6.87763310560392
p-value for Intercept: 0.0
p-value for years: 4.307631119037844e-18
p-value for years^2: 9.642847459359669e-12
Confidence interval for Intercept: 0    8.147439
1    8.554422
Name: Intercept, dtype: float64
Confidence interval for years: 0    0.891582
1    1.402745
Name: years, dtype: float64
Confidence interval for years^2: 0   -0.306691
1   -0.170554
Name: np.power(years, 2), dtype: float64
```

In [ ]:

Excercise 12.2

Write a defnition 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 [75]:
```
# Task 2: Define SerialCorrelationTest class

class SerialCorrelationTest(thinkstats2.HypothesisTest):
    """Tests serial correlation."""

    def TestStatistic(self, data):
        """Computes the serial correlation of the series with the given lag."""
        series, lag = data
        return SerialCorr(series, lag)

    def RunModel(self):
```

```
        """Runs the model."""
        series, lag = self.data
        permutation = series.reindex(np.random.permutation(series.index))
        return permutation, lag
```

In [76]:
```
# Test serial correlation in raw price data
raw_price_series = transactions["ppg"]
test_raw_price = SerialCorrelationTest((raw_price_series, 1))
p_value_raw_price = test_raw_price.PValue()
print("P-value for serial correlation in raw price data:", p_value_raw_price)
```

```
P-value for serial correlation in raw price data: 0.0
```

In [77]:
```
# Test serial correlation in residuals of linear model
residuals_linear = filled_dailies["high"].resid
test_residuals_linear = SerialCorrelationTest((residuals_linear, 1))
p_value_residuals_linear = test_residuals_linear.PValue()
print("P-value for serial correlation in residuals of linear model:", p_value_residuals_l
```

```
P-value for serial correlation in residuals of linear model: 0.603
```

In [78]:
```
# Test serial correlation in residuals of quadratic model
residuals_quadratic = quadratic_results.resid
test_residuals_quadratic = SerialCorrelationTest((residuals_quadratic, 1))
p_value_residuals_quadratic = test_residuals_quadratic.PValue()
print("P-value for serial correlation in residuals of quadratic model:", p_value_residual
```

```
P-value for serial correlation in residuals of quadratic model: 0.004
```

In [ ]:

Summary Report

Introduction:

This report presents the results of two tasks conducted on a dataset related to financial transactions. The first task involves fitting a quadratic model to the data and generating predictions, while the second task focuses on testing for serial correlation in different aspects of the dataset.

Problem Statement:

The primary objective is to understand the relationship between the dependent variable 'ppg' (presumably price per gram) and the independent variable 'years' (likely representing time). Additionally, the aim is to assess if there is any serial correlation present in the data, particularly in the residuals of the fitted models.

Methodology:

Task 1: Fit a Quadratic Model and Generate Predictions: For this task, a quadratic model is fitted to the dataset using ordinary least squares regression. The model is of the form: ppg = β0 + β1 *years* + *β2* years^2. This model allows for a non-linear relationship between 'ppg' and 'years'. After fitting the model, predictions are generated for future years using the fitted model.

Task 2: Test for Serial Correlation: In this task, a SerialCorrelationTest class is defined to assess the presence of serial correlation in the dataset. The test is conducted on two aspects:

Raw price data: The test determines if there's serial correlation in the original price data. Residuals of the models: The residuals of both linear and quadratic models are analyzed for serial correlation. These residuals represent the differences between the observed and predicted values, providing insights into the model's goodness of fit.

Results:

Task 1: The fitting results of the quadratic model are as follows:

R-squared: 0.085 F-statistic: 57.33 P-values for coefficients: Intercept (0.000), years (0.000), np.power(years, 2) (0.000) These results indicate a significant but relatively weak relationship between 'ppg' and 'years'. The predictions generated by the quadratic model suggest a decreasing trend in 'ppg' over the forecasted years.

Task 2:

P-value for serial correlation in raw price data: 0.0 P-value for serial correlation in residuals of linear model: 0.607 P-value for serial correlation in residuals of quadratic model: 0.002 These results indicate: Strong evidence of serial correlation in the raw price data. Lack of significant serial correlation in the residuals of the linear model. Significant serial correlation in the residuals of the quadratic model.

Discussion:

Task 1: Fitting a Quadratic Model and Generating Predictions: The fitting results of the quadratic model show an R-squared value of 0.085. This indicates that only 8.5% of the variability in the dependent variable 'ppg' can be explained by the independent variables 'years' and 'years^2'. While the F-statistic of 57.33 suggests that the overall model is statistically significant, the low R-squared value indicates that the model may not adequately capture the relationship between the variables.

Examining the coefficients, we observe significant p-values for all coefficients (Intercept, years, and years^2), indicating that they are statistically significant in predicting the 'ppg'. Specifically, the coefficient for the 'years' variable is 1.1472 with a p-value of 0.000, suggesting that for each additional year, the 'ppg' increases by approximately 1.15 units. Conversely, the coefficient for 'years^2' is -0.2386 with a p-value of 0.000, indicating a negative quadratic effect. This implies that while 'ppg' initially increases with 'years', it eventually decreases at an increasing rate.

The generated predictions exhibit a downward trend, with the 'ppg' values decreasing over the forecasted years. This aligns with the negative coefficient for the 'years^2' variable, indicating a decelerating growth rate in 'ppg' over time.

Task 2: Testing for Serial Correlation: The serial correlation test conducted on the raw price data yields a p-value of 0.0, indicating strong evidence of serial correlation. This suggests that there is a systematic relationship between consecutive observations in the raw price data, which violates the assumption of independence in many statistical analyses.

In contrast, the serial correlation tests conducted on the residuals of the linear and quadratic models yield different results. The p-value for the residuals of the linear model is 0.607, indicating no significant evidence of serial correlation. This suggests that the linear model adequately captures the autocorrelation present in the data.

However, the p-value for the residuals of the quadratic model is 0.002, indicating significant evidence of serial correlation. This implies that the quadratic model fails to fully account for the autocorrelation in the dataset, leading to residual patterns that are not adequately explained by the model.

In conclusion, while the quadratic model provides some insights into the relationship between 'ppg' and 'years', its explanatory power is limited. The presence of serial correlation in the raw price data suggests the need for further investigation and potentially more sophisticated modeling techniques

to account for this autocorrelation. Additionally, the significant serial correlation in the residuals of the quadratic model highlights potential deficiencies in the model's specification and suggests avenues for model improvement.

Conclusion:

In conclusion, the fitting of the quadratic model and the assessment of serial correlation provide valuable information about the dataset. While the quadratic model offers some predictive capability, its limited explanatory power and the presence of serial correlation in the residuals highlight potential areas for improvement in modeling and analysis.In conclusion, while the quadratic model provides some insights into the relationship between 'ppg' and 'years', its explanatory power is limited. The presence of serial correlation in the raw price data suggests the need for further investigation and potentially more sophisticated modeling techniques to account for this autocorrelation. Additionally, the significant serial correlation in the residuals of the quadratic model highlights potential deficiencies in the model's specification and suggests avenues for model improvement.

Way Forward:

Moving forward, it is recommended to explore alternative modeling techniques and data preprocessing methods to improve the predictive accuracy and explanatory power of the models. Additionally, further investigation into the sources of serial correlation in the dataset may aid in refining the modeling approach and enhancing the overall understanding of the underlying patterns in the data.

In [ ]: