xudongmit / Statistics-Computation

Branch: master Statistics-Computation / pset5 / pset5.md Find file Copy path

xudongmit pset5 3aae6d4 4 days ago

1 contributor

418 lines (327 sloc) 13.5 KB

Problem Set 5

```
import pandas as pd
import numpy as np
import os
from datetime import datetime
from statsmodels import tsa
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
import matplotlib.style as style
import seaborn as sns

os.chdir('e:/MIT4/statistics-Computation/pset5')
```

5.1 BPP Data Analysis

```
df_cpi = pd.read_csv('data/PriceStats_CPI.csv')
df_ber = pd.read_csv('data/T10YIE.csv')
df_cpi.head()

df_cpi['date'] = pd.to_datetime(df_cpi['date'])
```

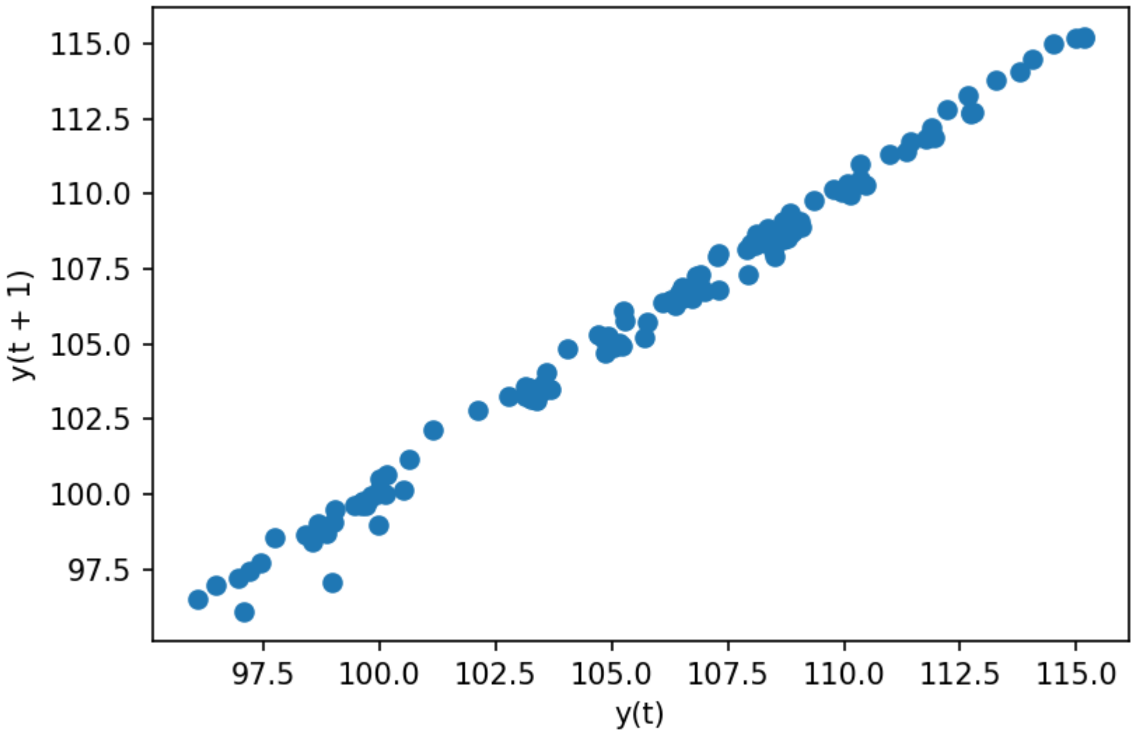
(a)

First, we will try to predict the monthly CPI without using the BER or PriceStats. Fit an AR model to the CPI data (take first CPI value of each month as that month's CPI, you may or may not want to work in log scale in order to make the model comparable to models you t in part (c)) and report the mean squared prediction error for 1 month ahead forecasts. Which order model gives the best predictions?

```
# Firstly we investigate the time series change in the dataset

def diff_month(d1, d2):
    '''
    Calculate the number of month between two timestamps
    '''
    return (d1.year - d2.year)*12 + d1.month - d2.month

# the first timestamp in the dataset
ini_date = df_cpi['date'][0]
# monthly data
df_cpi['month_index'] = df_cpi.apply(lambda row: diff_month(row.date, ini_date), axis=1)
# average cpi of every month
data_monthly= df_cpi.groupby('month_index').agg({'CPI': 'first','PriceStats':'mean'})

# plot the CPI of t+1 and t to find the time series pattern
pd.plotting.lag_plot(data_monthly.CPI)
plt.savefig("figure/1a0.png", dpi=150)
plt.show()

# the absolute changes in 1 month for all 122 months
```

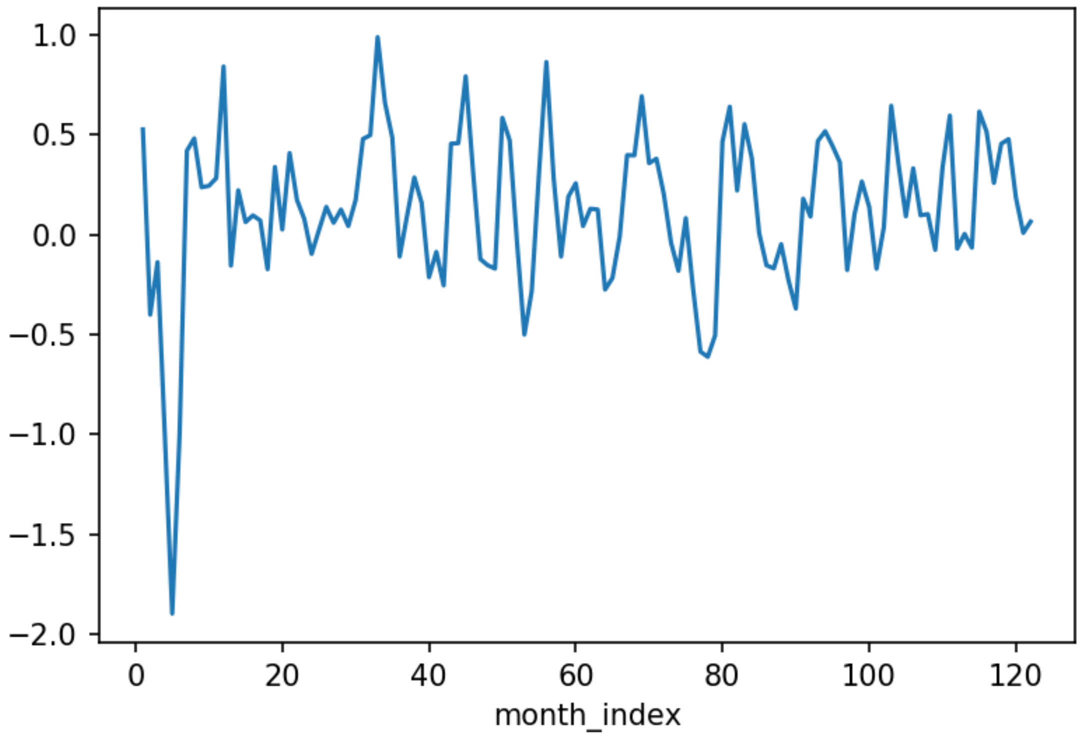
```
monthly_diff = data_monthly.diff()
monthly_diff.head()
monthly_diff.CPI.plot()
plt.savefig("figure/1a1.png", dpi=150)
plt.show()


all_diff = monthly_diff.iloc[1:]

# choose the observations before Sep 2013 for fitting models
n =  diff_month(pd.to_datetime('2013-09-01'), ini_date)
# split dataset
train = all_diff.iloc[0:n]
test = all_diff.iloc[n:-1]

# fit the model AR(1)
model = tsa.ar_model.AR(train.CPI.values)
fit1 = model.fit(maxlag=1)
fit1.predict(start=len(train), end=len(train))

def predict(order):
    train_set = train
    test_set = test
    model = tsa.ar_model.AR(train_set.CPI.values)
    fit = model.fit(maxlag = order)
    predictions = []
    for i in range(1, len(test)+1):
        result = fit.predict(start=len(train_set), end = len(train_set))
        predictions.append(result)
        train_set = all_diff.iloc[0:56+i]
        test_set = all_diff.iloc[56+i:]
        model = tsa.ar_model.AR(train_set.CPI.values)
        fit = model.fit(maxlag = order)
    mse = mean_squared_error(predictions, test.CPI.values)
    return mse

predict(1)

# Try AR(n), n = 1,2,...10
ar_results = pd.DataFrame(index = range(1, 10), columns = ['mse'])
mses = []
for idx in ar_results.index.tolist():
    ar_results['mse'][idx] = predict(idx)

ar_results.plot()
plt.savefig("mse.png", dpi=150)
plt.show()
# AR(5) model
model = tsa.ar_model.AR(train.CPI.values)
fit = model.fit(maxlag = 5)
fit.params
```

Plot the CPI of t+1 and t to find the time series pattern:



It is obvious that there is time series correlation.

Plot absolute changes in 1 month for all 122 months:



Fit the AR models:
To decide the best n for AR(n) model, we tried from AR(1) to AR(10) and plot the MSE. We found that n=5 has the best MSE.
We estimated an AR(5) model and got the parameters: (phi_i is the coefficient for X_t-i) phi_1 = 0.13896421
phi_2 = 0.46679018
phi_3 = -0.10417631
phi_4 = -0.29061451
phi_5 = 0.21927141
W = -0.07312825

(b)

How might you calculate monthly inflation rates from the CPI data and your 1 month ahead predictions? How about from PriceStats data? And BER data? (What dates would you use? Or would you use an average of many dates?) Overlay your estimates of monthly inflation rates (there should be 4 lines, one for each dataset, plus the predictions) over time (months from September 2013 onward).

```
# read the BER data
df_ber = df_ber.mask(df_ber == '.')
df_ber = df_ber.dropna()
df_ber.head()
df_ber['date1'] = pd.to_datetime(df_ber['DATE'])
df_ber['month_index'] = df_ber.apply(lambda row: diff_month(row.date1, ini_date), axis=1)
df_ber['BER'] = df_ber.apply(lambda row: float(row.T10YIE), axis=1)

monthly_ber = df_ber.groupby('month_index').agg({'BER':{'ber_avg': 'mean', 'ber_last': 'last'}})
monthly_ber.columns = monthly_ber.columns.droplevel(0)

monthly_other = df_cpi.groupby('month_index').agg({'CPI': 'first','PriceStats':'mean'})

monthly_all = monthly_ber.join(monthly_other, how='inner')
diff_df = monthly_all.diff().iloc[1:]
diff_df.head()
```

To calculate the inflation rate, r = CPI_t - CPI_(t-1)/CPI_t,

(c)

Next, we will include external regressors to try to improve the predictions. Include as external regressors monthly average PriceStats data and BER data tot a new AR model to the CPI. Report your prediction error. Try instead using PriceStats data and BER data from the last day of each month as your external regressors. Fit another AR model. Which model performs better in prediction? (Hint: Again, in order to match the units of your predictors and responses, you'll want to either work on a log scale or work with inflation rates. Please justify your choices in which values you decide to work with.)

```
train_ex = diff_df.iloc[0:n]
test_ex = diff_df.iloc[n:-1]

def update_regressor(train_set, regres_names):
    regres_list = []
    for name in regres_names:
        regres_list.append(train_set[name].values)
    regressors = np.column_stack(regres_list)
    return regressors

def predict_ex(order, regres_names):
    train_set = train_ex
    test_set = test_ex
    ex_vars = update_regressor(train_set, regres_names)
    model = tsa.arima_model.ARIMA(endog = train_set.CPI.values, order = (order,0,0), exog = ex_vars)
    fit = model.fit()
    predictions = []
    for i in range(1, len(test)+1):
        result = fit.predict(start=len(train_set), end = len(train_set), exog = ex_vars)
        predictions.append(result[0])
        train_set = diff_df.iloc[0:56+i]
        test_set = diff_df.iloc[56+i:]
        ex_vars = update_regressor(train_set, regres_names)
        model = tsa.arima_model.ARIMA(endog = train_set.CPI.values, order = (order,0,0), exog = ex_vars)
        fit = model.fit()
    mse = mean_squared_error(predictions, test_ex.CPI.values)
    return mse
```

```
predict_ex(1, ['ber_avg'])
```

```
predict_ex(2, ['ber_avg'])
```

(d)

Try to improve your model from part (c). What is the smallest prediction error you can obtain? You might consider including MA terms, adding a seasonal AR term, or adding multiple daily values (or values from different months) of PriceStats and BER data as external regressors.

```
ar_ex_results = pd.DataFrame(index = range(1, 4), columns = ['avg_mse', 'last_mse'])
for idx in ar_ex_results.index.tolist():
    ar_ex_results['avg_mse'][idx] = predict_ex(idx, ['ber_avg'])
    ar_ex_results['last_mse'][idx] = predict_ex(idx, ['ber_last'])
ar_ex_results
```

```
ex_vars = update_regressor(train_ex, ['ber_last'])
model = tsa.arima_model.ARIMA(endog = train_ex.CPI.values, order = (2,0,0), exog = ex_vars)
fit = model.fit()
print(fit.summary())
```

```
def predict_am(order_am, regres_names):
    train_set = train_ex
    test_set = test_ex
    ex_vars = update_regressor(train_set, regres_names)
    model = tsa.arima_model.ARIMA(endog = train_set.CPI.values, order = (2,0,order_am), exog = ex_vars)
    fit = model.fit()
    predictions = []
    for i in range(1, len(test)+1):
        result = fit.predict(start=len(train_set), end = len(train_set), exog = ex_vars)
        predictions.append(result[0])
        train_set = diff_df.iloc[0:56+i]
        test_set = diff_df.iloc[56+i:]
        ex_vars = update_regressor(train_set, regres_names)
        model = tsa.arima_model.ARIMA(endog = train_set.CPI.values, order = (2,0,order_am), exog = ex_vars)
        fit = model.fit(method = 'css')
    mse = mean_squared_error(predictions, test_ex.CPI.values)
    return mse
```

```
predict_am(1, ['ber_avg'])
```

```
predict_am(1, ['ber_last'])
```

ARMA model estimation result:

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   62
Model:                     ARMA(2, 0)   Log Likelihood                 -26.865
Method:                       css-mle   S.D. of innovations              0.372
Date:                Mon, 26 Nov 2018   AIC                             63.731
Time:                        18:28:29   BIC                             74.366
Sample:                             0   HQIC                            67.907

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1194      0.077      1.559      0.124      -0.031       0.270
x1             0.0752      0.164      0.459      0.648      -0.246       0.396
ar.L1.y        0.6585      0.124      5.298      0.000       0.415       0.902
ar.L2.y       -0.2735      0.126     -2.176      0.034      -0.520      -0.027
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.2037           -1.4856j            1.9120           -0.1416
AR.2            1.2037           +1.4856j            1.9120            0.1416
-----------------------------------------------------------------------------
```

(e)

Consider the MA(1) model, X_t = W_t + theta*W_t-1, where {W_t} ~ WN(0, sigma^2). Find the autocovariance function of {X_t}.

1e

(f)

Consider the AR(1) model, X_t = phi*X_t-1 + W_t, where {W_t} ~ WN(0, sigma^2). Suppose |phi| < 1. Find the autocovariance function of {X_t}.

1f

5.2 The Mauna Loa CO2 Concentration

```
# deleted the text rows in Excel
df_ml = pd.read_csv('data/CO2Data.csv')
df_ml.columns = ['Yr', 'Mn', 'Date_Excel', 'Date', 'CO2', 'seasonally_adjusted','fit', 'seasonally_adjusted_fit', 'CC

data = df_ml[['Yr', 'Mn', 'CO2']].copy()
data['date'] = data.apply(lambda row: datetime(year=int(row['Yr']), month=int(row['Mn']), day=1), axis=1)

ini_date = data['date'][0]
data['month_index'] = data.apply(lambda row: 1+diff_month(row.date, ini_date), axis=1)
data = data[data.CO2 != -99.99]
data.head()
```

(a)

Fit the data to a linear model.Plot the data and thefit.

```
y = data.CO2
X = data.month_index
X = sm.add_constant(X)
f1 = sm.OLS(y, X)
f1 = f1.fit()
f1.summary()
f1.params

X_prime = np.linspace(X.month_index.min(), X.month_index.max(), 100)[:, np.newaxis]
X_prime = sm.add_constant(X_prime)

y_hat = f1.predict(X_prime)

plt.scatter(X.month_index, y, alpha=0.3, s= 1, color = 'blue')  # Plot the raw data
plt.xlabel("Month")
plt.ylabel("CO2")
plt.plot(X_prime[:, 1], y_hat, 'r', alpha=0.9)
plt.savefig("figure/2a1.png", dpi=150)
plt.show()


fig = plt.figure(1)
fig.set_figheight(6)
fig.set_figwidth(8)
model_fitted_y = f1.fittedvalues
model_residuals = f1.resid
fig.axes[0] = sns.residplot(model_fitted_y, 'CO2', data=data,
                          lowess=True,
                          scatter_kws={'color': 'blue','alpha': 0.5},
                          line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
fig.axes[0].set_title('Residuals vs Fitted')
fig.axes[0].set_xlabel('Fitted values')
fig.axes[0].set_ylabel('Residuals')
plt.savefig("figure/2a2.png", dpi=150)
plt.show()
```

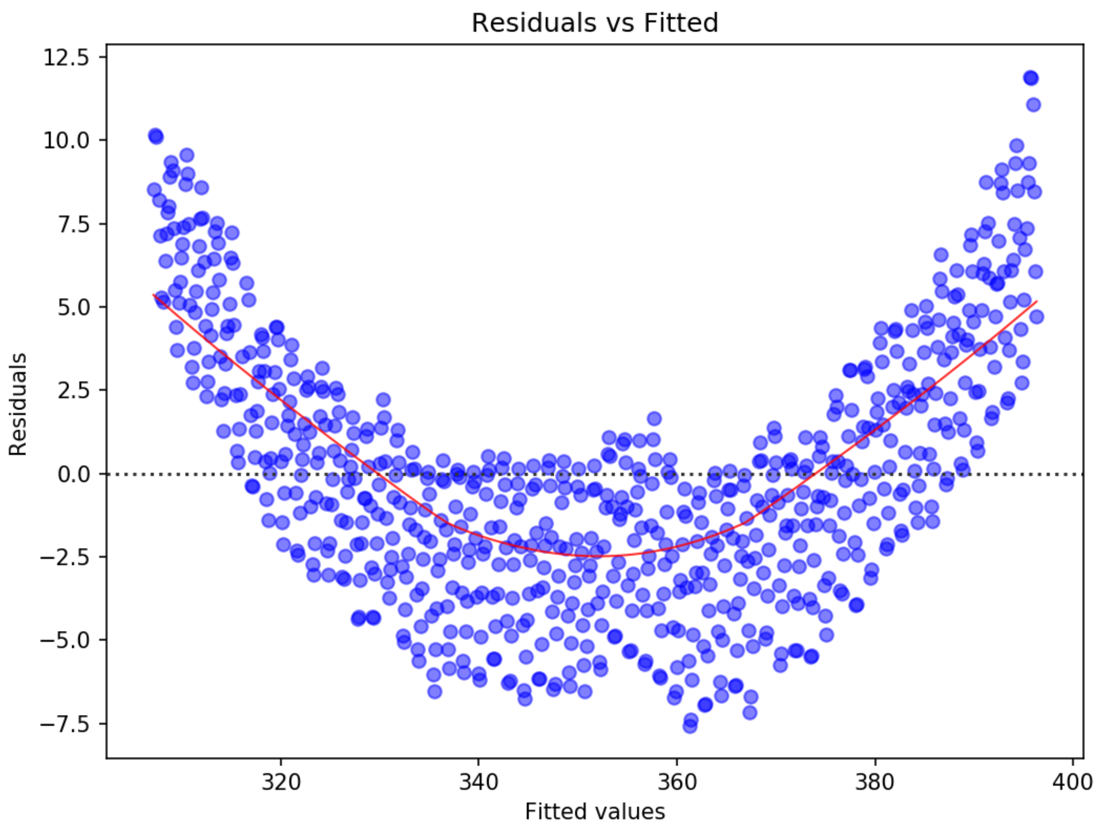
Regression result:

Dep. Variable:	CO2	**R-squared:**	0.978
Model:	OLS	**Adj. R-squared:**	0.978
Method:	Least Squares	**F-statistic:**	3.047e+04
Date:	Mon, 26 Nov 2018	**Prob (F-statistic):**	0.00
Time:	18:38:31	**Log-Likelihood:**	-1936.2
No. Observations:	698	**AIC:**	3876.
Df Residuals:	696	**BIC:**	3885.
Df Model:	1		
Covariance Type:	nonrobust		

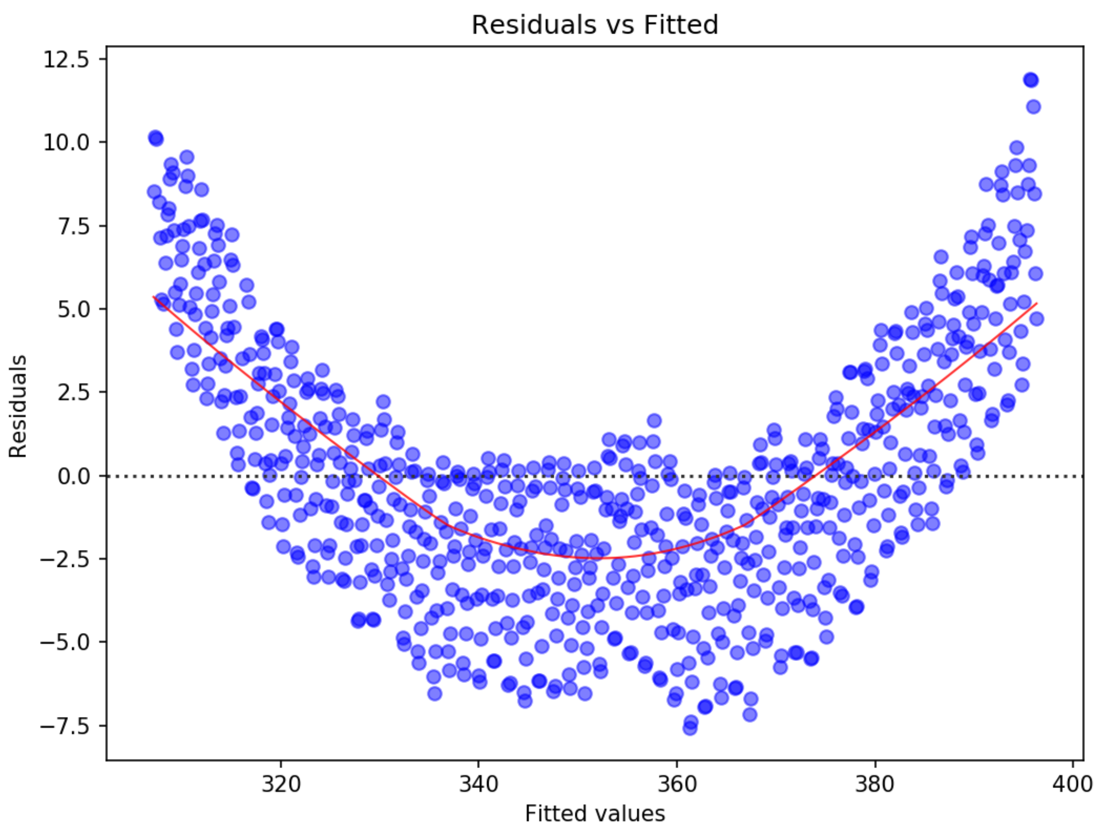
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	306.7671	0.298	1029.800	0.000	306.182	307.352
month_index	0.1270	0.001	174.566	0.000	0.126	0.128

Omnibus:	28.823	**Durbin-Watson:**	0.104
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	31.507
Skew:	0.512	**Prob(JB):**	1.44e-07
Kurtosis:	2.810	**Cond. No.**	830.

Linear model fit plot:



Residual-Fitted plot:



The estimated parameters:

const 306.767103
month_index 0.126996

The residual plot shows that the error term violates the assumption of OLS.

(b)

Fit the data to a quadratic model. Plot the data and the fit.

```
f2 = smf.ols( formula = 'CO2 ~ np.power(month_index, 2) + month_index + 1', data = data)
f2 = f2.fit()
f2.summary()
f2.params

plt.plot(X.month_index, f2.predict(X), 'r')
plt.scatter(X.month_index, y, alpha=0.3, s= 1, color = 'blue')
plt.savefig("figure/2b1.png", dpi=150)
plt.show()


fig = plt.figure(1)
fig.set_figheight(6)
fig.set_figwidth(8)
model_fitted_y = f2.fittedvalues
model_residuals = f2.resid
fig.axes[0] = sns.residplot(model_fitted_y, 'CO2', data=data,
                          lowess=True,
                          scatter_kws={'color': 'blue','alpha': 0.5},
                          line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
fig.axes[0].set_title('Residuals vs Fitted')
fig.axes[0].set_xlabel('Fitted values')
fig.axes[0].set_ylabel('Residuals')
plt.savefig("figure/2b2.png", dpi=150)
plt.show()
```

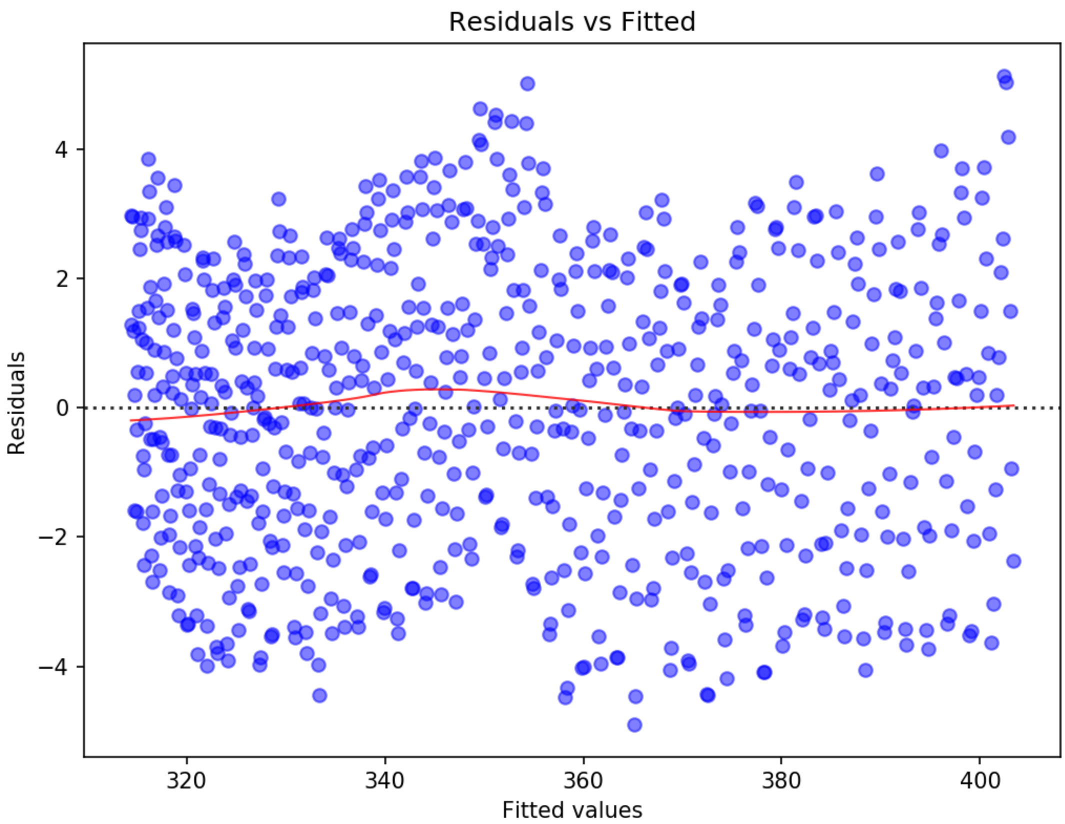
Regression result:

Dep. Variable:	CO2	**R-squared:**	0.993
Model:	OLS	**Adj. R-squared:**	0.993
Method:	Least Squares	**F-statistic:**	4.754e+04
Date:	Mon, 26 Nov 2018	**Prob (F-statistic):**	0.00
Time:	18:41:49	**Log-Likelihood:**	-1543.9
No. Observations:	698	**AIC:**	3094.
Df Residuals:	695	**BIC:**	3107.
Df Model:	2		
Covariance Type:	nonrobust		

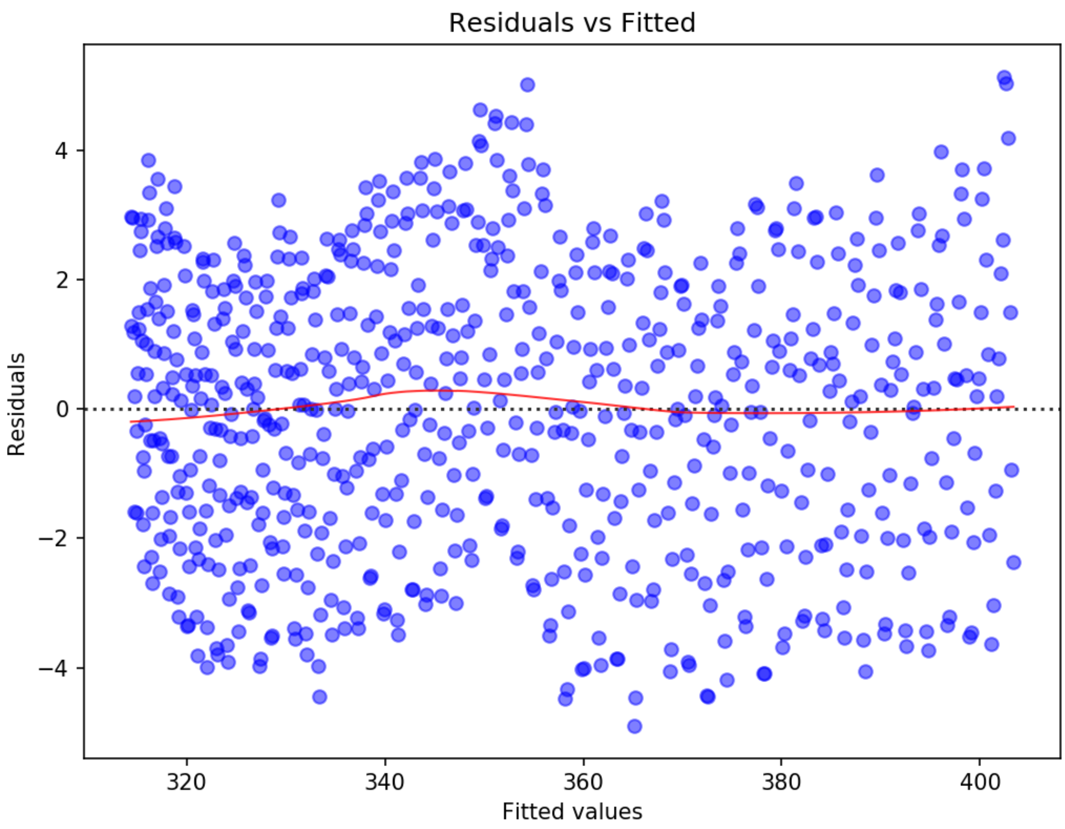
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	314.2065	0.259	1212.005	0.000	313.698	314.716
np.power(month_index, 2)	8.692e-05	2.29e-06	37.998	0.000	8.24e-05	9.14e-05
month_index	0.0652	0.002	38.838	0.000	0.062	0.068

Omnibus:	100.656	**Durbin-Watson:**	0.319
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	25.835
Skew:	-0.092	**Prob(JB):**	2.45e-06
Kurtosis:	2.076	**Cond. No.**	6.92e+05

Linear model fit plot:



Residual-Fitted plot:



The estimated parameters:
Intercept 314.206513
np.power(month_index, 2) 0.000087
month_index 0.065193

The residuals are closer to N(0, sigma).

(c)

Which fit (F1 or F2) is better in capturing the trend in the data? Explain.

The F2 is better. The residuals in F1 are not distributed like N(0, sigma). In addition, F2 has higher adjusted R-squared value than F1.

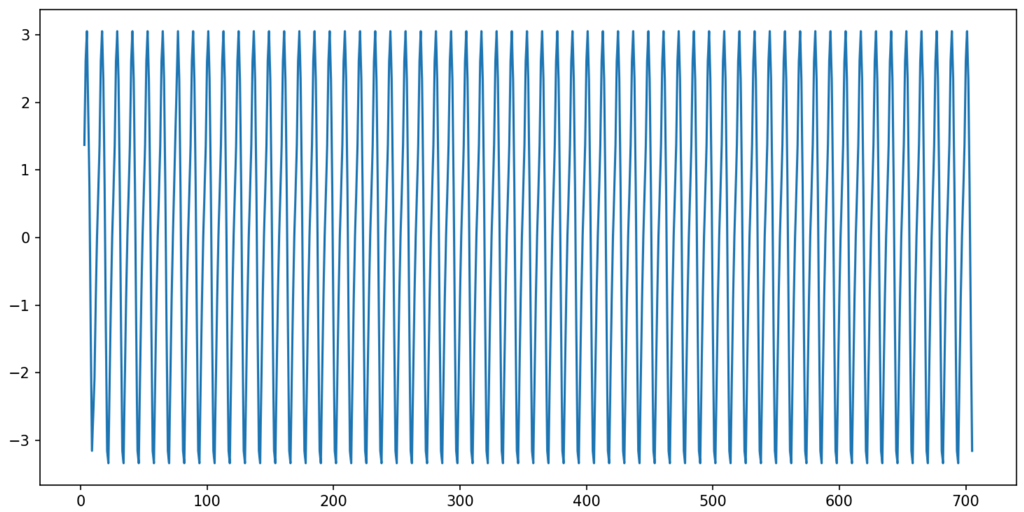
(d)

Consider F2(t). We will now extract the periodic component which appears in the data. Average the residual C_i - F2(t_i) over each month. Namely, collect all the data for Jan (resp.Feb, Mar, etc) and average them to get one data point for Jan (resp. Feb, Mar, etc). The collection of those points can be interpolated to form a periodic signal P_i. Plot P_i.

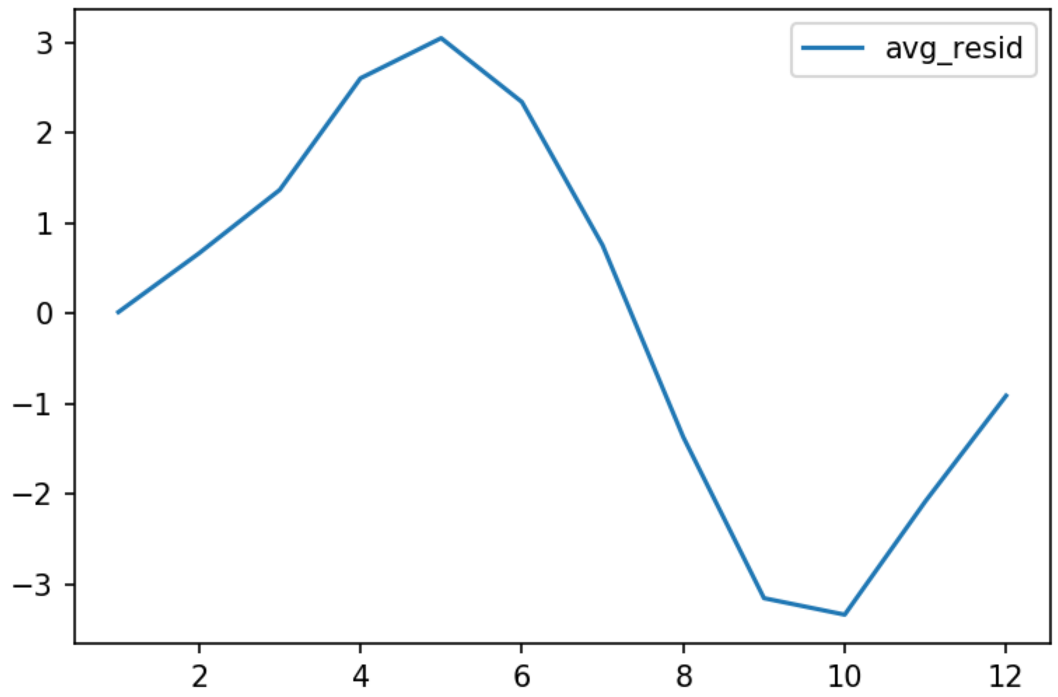
```
data['resid'] = pd.Series(f2.resid)
data.head()

def extract_month(m):
    temp = data.loc[data['Mn'] == m]
    return temp['resid'].mean()
months = pd.DataFrame(index = range(1, 13), columns = ['avg_resid'])
for idx in months.index.tolist():
    months['avg_resid'][idx] = extract_month(idx)
months

fig = plt.figure(1)
fig.set_figheight(6)
fig.set_figwidth(12)
fig = months.plot()
plt.savefig("figure/2d1.png", dpi=150)
```

```
plt.show()

data = data.join(months, on = 'Mn', how = 'left')

fig = plt.figure(1)
fig.set_figheight(6)
fig.set_figwidth(12)
fig = plt.plot(data.month_index, data.avg_resid)
plt.savefig("figure/2d.png", dpi=150)
plt.show()
```

Plot of periodic signal P_i to i:



Monthly Residual Plot in 1 Year:



(e)

Plot thet F2(t_i) + P_i. What can we conclude on the variation of the CO2 concentration since 1958?

```
fig = plt.figure(1)
fig.set_figheight(6)
fig.set_figwidth(12)
fig = plt.plot(data.month_index, data.avg_resid + f2.fittedvalues, 'r')
plt.scatter(X.month_index, y, alpha=0.8, s= 2)
plt.savefig("figure/2e.png", dpi=150)
plt.show()
```

Plot thet F2(t_i) + P_i: