import numpy

from pandas import Series

from matplotlib import pyplot

import matplotlib.pyplot as plt

from pandas.tools.plotting import autocorrelation\_plot

from statsmodels.graphics.tsaplots import plot\_pacf

from statsmodels.graphics.tsaplots import plot\_acf

from pandas import DataFrame

from statsmodels.tsa.arima\_model import ARIMA

import statsmodels.api as sm

series = Series.from\_csv('price\_data\_3.csv', header=0)

split\_point = len(series)

dataset, validation = series[0:split\_point], series[split\_point:]

print('Dataset %d, Validation %d' % (len(dataset), len(validation)))

dataset.to\_csv('dataset.csv')

validation.to\_csv('validation.csv')

print(series)

X=series.values

# Plot Data

pyplot.plot(X)

pyplot.show()



#

# plot PACF Partial Autocorrelation for ORIGINAL Data

print('PACF ORIGINAL Data')

pyplot.figure()

plot\_pacf(series, lags=60)

pyplot.show()



#Differenced Data

diff = list()

for i in range(1, len(X)):

 value = X[i] - X[i - 1]

 diff.append(value)

# Plot Differenced Data

pyplot.plot(diff)

pyplot.show()



print('ACF Differenced Data ')

plot\_acf(diff)

pyplot.show()

print('PACF Differenced Data')

pyplot.figure()

plot\_pacf(diff, lags=60)

pyplot.show()





series = Series.from\_csv('price\_data\_3.csv', header=0)

print(series)

X=series.values

X=numpy.array(X).astype('float')

order=(3,1,1) # (1,1,1)

#

print 'START ARIMA',order,' for price\_data\_3 Example '

model = ARIMA(X, order)

model\_fit = model.fit(disp=0)

print(model\_fit.summary())

# plot residual errors

residuals = DataFrame(model\_fit.resid)

print('Plot Residuals')

residuals.plot()

pyplot.show()

residuals.plot(kind='kde')

pyplot.show()

print(residuals.describe())



print 'Autocorrelation plot for Residuals and ARIMA',order,' for price\_data\_3 Example '

plot\_acf(residuals)

pyplot.show()

#

print 'Partial Autocorrelation plot for Residuals and ARIMA',order,' for price\_data\_3 Example'

plot\_pacf(residuals)

pyplot.show()



DWT = sm.stats.durbin\_watson(residuals)

print 'Durbin-Watson Test=',DWT

#

r,q,p = sm.tsa.acf(residuals, qstat=True)

data = numpy.c\_[range(1,41), r[1:], q, p]

table = DataFrame(data, columns=['lag', "AC", "Q", "Prob(>Q)"])

print table.set\_index('lag')

print

size = int(1350)

train = X[0:size] # множество за …..

test = X[size:len(X)] # множество за тестване

history = [x for x in train]

predictions = list()

for t in range(len(test)):

 model = ARIMA(history, order)

 model\_fit = model.fit(disp=0)

 output = model\_fit.forecast()

 yhat = output[0]

 predictions.append(yhat)

 obs = test[t]

 history.append(obs)

 print('=%i, predicted=%f, expected(real)=%f' % (size+t,yhat, obs))

“Results :”

=1350, predicted=8185.414492, expected(real)=8208.167000

=1351, predicted=8204.412456, expected(real)=8372.262000

=1352, predicted=8423.435396, expected(real)=8233.721000

=1353, predicted=8184.571883, expected(real)=8237.831000

=1354, predicted=8262.054539, expected(real)=8232.435000

……………………….

=1397, predicted=8366.265289, expected(real)=8436.937000

=1398, predicted=8462.984500, expected(real)=8503.049500

=1399, predicted=8503.507202, expected(real)=8532.541000

=1400, predicted=8527.602368, expected(real)=8506.588000

=1401, predicted=8497.237501, expected(real)=8507.403333

def difference(dataset, interval=1):

 diff = list()

for i in range(interval, len(dataset)):

 value = dataset[i] – dataset[i – interval]

 diff.append(value)

 return numpy.array(diff)

def inverse\_difference(history, yhat, interval=1):

 return yhat + history[-interval]

# load dataset

series = Series.from\_csv('dataset.csv', header=None)

# seasonal difference

X = series.values

days\_in\_year = 365

differenced = difference(X, days\_in\_year)

# fit model

model = ARIMA(differenced, order=(3,1,1))

model\_fit = model.fit(disp=0)

# one-step out-of sample forecast

forecast = model\_fit.forecast()[0]

# invert the differenced forecast to something usable

forecast = inverse\_difference(X, forecast, days\_in\_year)

print('Forecast: %f' % forecast)

print

print 'Durbin-Watson Test=',DWT

print('ARIMA ',order)

MFE = (predictions-test).mean()

print "MFE = ",MFE

import numpy

MAE = (numpy.abs((predictions-test).mean()) / predictions).mean()

print "MAE = ", MAE

#

from sklearn.metrics import mean\_squared\_error

rmse = numpy.sqrt(mean\_squared\_error(test, predictions))

print('Test RMSE: %.3f' % rmse)

# plot

pyplot.plot(test)

pyplot.plot(predictions, color='red')

pyplot.show()

