Solution Documents for Oil price and its commodities prices trends and relation.

Python required following libraries to be imported.

import pandas as pd

from matplotlib import pyplot as plt

from scipy import stats

import statsmodels.api as sm

import seaborn as sns

from pandas.plotting import autocorrelation\_plot

from statsmodels.tsa.arima\_model import ARIMA

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

## **Data:**

We have data in the form of csv files. All csv files contain data from 1989 to 2019.

Csv files have two columns first is time series (monthly) and second commodity price.

Date,WTI

1989-01-01,18.02

1989-02-01,17.94

1989-03-01,19.48

To read a csv file we have function:

def readData(csvFile):

data = pd.read\_csv(csvFile, index\_ ='Date', parse\_dates=True)

return data

This takes a csv file as input and return data-frame which has date are index and commodity as data.

### Function: priceTrend

Input: dataframe which has date as index and corresponding commodity data.

def priceTrend(data):

data.plot(figsize=(10,5), linewidth=1)

plt.show()

it plots commodity price with respect to time.

### Function: getVar

def getVAR(data, col):

return data[col].var()

This returns variance of column data.

### Function: getMean

def getMean(data, col):

return data[col].mean()

This returns mean of column data.

### Function: getMean

def getMean(data, col):

return data[col].mean()

This return mean of column data.

### Function: getSkewAndKurt

def getSkewAndKurt(data, col):

mean = data[col].mean()

median = data[col].median()

mode = data[col].mode()

skewness = data[col].skew()

kurtosis = data[col].kurt()

print('Mean = {}'.format(mean))

print('Median = {}'.format(median))

print('Mode = ', mode[0])

print('Skewness = ', skewness)

print('Kurtosis = ', kurtosis)

plt.figure(figsize=(10, 5))

plt.hist(data[col], bins=10, color='grey')

plt.axvline(mean,color='red',label='Mean')

plt.axvline(median,color='yellow',label='Median')

plt.axvline(mode[0],color='green',label='Mode')

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.legend()

plt.show()

return skewness,kurtosis

This return skewness, kurtosis of column data and plot distribution of data.

### Function: drawNormal

def drawNormal(data, col):

sns.distplot(data[col], hist = False, kde = True,

kde\_kws = {'shade': True, 'linewidth': 3},

label = col)

plt.ylabel('Density')

plt.title('Density Plot')

plt.show()

This draw normal distribution of data.

### Function: drawAutoCorr

def drawAutoCorr(df, col):

autocorrelation\_plot(df[col])

plt.show()

This draw correlation of data available in different columns.

### Function: arimaModelFitAnalysis

def arimaModelFitAnalysis(data, col):

# fit model

model = ARIMA(data[col], order=(5,1,0))

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

print(model\_fit.summary())

# plot residual erros

residuals = pd.DataFrame(model\_fit.resid)

residuals.plot()

residuals.plot(kind='kde')

plt.show()

print(residuals.describe())

This function provide ARIMA model and plot of residuals.

### Function: arimaModelRollingForCast

def arimaModelRollingForCast(data, col):

X = data[col]

size = int(len(X) \* 0.66)

train, test = X[0:size], X[size:len(X)]

history = [x for x in train]

predictions = []

for t in range(len(test)):

model = ARIMA(history, order=(5,1,0))

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

output = model\_fit.forecast()

yhat = output[0]

predictions.append(yhat[0])

obs = test[t]

history.append(obs)

print('predicted=%f, expected=%f' % (yhat, obs))

error = mean\_squared\_error(test, predictions)

print('Test MSE: %.3f' % error)

df = pd.DataFrame(columns={'Test'})

df['Test'] = test

df['Predictions'] = predictions

# plot

print(df.head())

df.plot()

plt.show()

This function provides ARIMA model and plot of residuals.

### Function: olsModel

def olsModel(X, Y, y\_col):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.25)

sm.add\_constant(X\_train)

model = sm.OLS(y\_train, X\_train)

result = model.fit()

yPredict = result.predict(X\_test)

df = pd.DataFrame()

df[y\_col] = y\_test;

df[y\_col + '\_Predict'] = yPredict

print(result.summary())

df.plot()

plt.show()

To find linear relation between two commodities, this function predicts and plot the relation.

This function provides ARIMA model and plot of residuals.