In [39]:
```
# API lib

import requests
#  import base64
import json
import math
import numpy as np
from numpy import concatenate
import pandas as pd
# import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# from keras.models import load_model
# import seaborn as sns
# sns.set()
```

Init

In [64]:
```
# APIs
api_key = ''
api_header = {}
with open('api_header.json') as f:
    api_header = json.load(f)
f = open("api_key.txt", "r")
api_key = f.read()
api_header["X-IG-API-KEY"] = api_key


pd.options.display.max_columns = None


# start_session_init()
# Last checkout 2020-04-23T00:00:00
# date0 = '2019-05-01T00%3A00%3A00'
date1 = '2020-04-23T00%3A00%3A00'
date2 = '2020-04-27T20%3A00%3A00'
# resolution = 'HOUR_2'
resolution = 'MINUTE_30'
xau_epic = 'CS.D.CFDGOLD.CFDGC.IP'
usd_epic = 'CO.D.DX.FWS2.IP'
us500_epic = 'IX.D.SPTRD.IFD.IP'
us100_epic = 'IX.D.NASDAQ.IFD.IP'
usoil_epic = 'CC.D.CL.UNC.IP'
eur_epic = 'CS.D.EURUSD.CFD.IP'
ftse_epic = 'IX.D.FTSE.CFD.IP'
eurchn_epic = 'CS.D.EURCNH.CFD.IP'
usdchn_epic = 'CS.D.USDCNH.CFD.IP'

tables = ['xau','usd','us500','us100','usoil','eur','ftse','eurchn','usdchn']
```

Functions

In [44]:
```
def start_session():
    url = "https://demo-api.ig.com/gateway/deal"
    session = "/session/encryptionKey"
    m_url = url + session
    return  requests.get(m_url, headers=api_header)

def price_history(epic,resolution,date1,date2):
    m_url = "https://api.ig.com/gateway/deal/prices/{}?resolution={}&from={}&t
o={}&pageSize=0".format(epic,resolution,date1,date2)
     # "Version": "2"
    return  requests.get(m_url, headers=api_header)

def price_extractor(dfx,obj):
    # always have problem with str or not str. please convert to csv first
    suffix = '' # obj + '_'
    dfx[suffix +'openPrice'] = dfx['openPrice'].apply(lambda x: (eval(x)).get(
'ask'))
    dfx[suffix +'closePrice'] = dfx['closePrice'].apply(lambda x: (eval(x)).ge
t('ask'))
    dfx[suffix +'highPrice'] = dfx['highPrice'].apply(lambda x: (eval(x)).get(
'ask'))
    dfx[suffix +'lowPrice'] = dfx['lowPrice'].apply(lambda x: (eval(x)).get('a
sk'))

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

# scale train and test data to [0, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, yhat):
    new_row = [x for x in X] + [yhat]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

def adj_r2(x,r2):
    n = x.shape[0]
    p = x.shape[1]
    adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
    return adjusted_r2
```

Data Import from API. *[Do only when needed!]*

In [13]:
```
# Data Import
xau = price_history(xau_epic,resolution,date1,date2)
xau_list = json.loads(xau.text)['prices']
usd = price_history(usd_epic,resolution,date1,date2)
usd_list = json.loads(usd.text)['prices']
us500 = price_history(us500_epic,resolution,date1,date2)
us500_list = json.loads(us500.text)['prices']
us100 = price_history(us100_epic,resolution,date1,date2)
us100_list = json.loads(us100.text)['prices']
usoil = price_history(usoil_epic,resolution,date1,date2)
usoil_list = json.loads(usoil.text)['prices']
eur = price_history(eur_epic,resolution,date1,date2)
eur_list = json.loads(eur.text)['prices']
ftse = price_history(ftse_epic,resolution,date1,date2)
ftse_list = json.loads(ftse.text)['prices']
eurchn = price_history(eurchn_epic,resolution,date1,date2)
eurchn_list = json.loads(eurchn.text)['prices']
usdchn = price_history(usdchn_epic,resolution,date1,date2)
usdchn_list = json.loads(usdchn.text)['prices']
```

In [24]:
```
# DataFrame convert
new_xau = pd.DataFrame(xau_list)
new_usd = pd.DataFrame(usd_list)
new_us500 = pd.DataFrame(us500_list)
new_us100 = pd.DataFrame(us100_list)
new_eur = pd.DataFrame(eur_list)
new_ftse = pd.DataFrame(ftse_list)
new_usoil = pd.DataFrame(usoil_list)
new_eurchn = pd.DataFrame(eurchn_list)
new_usdchn = pd.DataFrame(usdchn_list)
```

Backup / Restore DataFrames

In [4]:
```
# Backup NEW Data ------------------------------------
# archive_path = 'data/new_'
# new_xau.to_csv(archive_path + 'df_xau'+ '_' + resolution +'.csv',index=Fals
e,header = True)
# new_usd.to_csv(archive_path + 'df_usd'+ '_' + resolution +'.csv',index=Fals
e,header = True)
# new_us500.to_csv(archive_path + 'df_us500'+ '_' + resolution +'.csv',index=F
alse,header = True)
# new_us100.to_csv(archive_path + 'df_us100'+ '_' + resolution +'.csv',index=F
alse,header = True)
# new_usoil.to_csv(archive_path + 'df_usoil'+ '_' + resolution +'.csv',index=F
alse,header = True)
# new_eur.to_csv(archive_path + 'df_eur'+ '_' + resolution +'.csv',index=Fals
e,header = True)
# new_ftse.to_csv(archive_path + 'df_ftse'+ '_' + resolution +'.csv',index=Fal
se,header = True)
# new_eurchn.to_csv(archive_path + 'df_eurchn'+ '_' + resolution +'.csv',index
=False,header = True)
# new_usdchn.to_csv(archive_path + 'df_usdchn'+ '_' + resolution +'.csv',index
=False,header = True)

archive_path = 'data/new_'
new_xau = pd.read_csv(archive_path + 'df_xau'+ '_' + resolution +'.csv', parse
_dates=['snapshotTime']) # Data Restore
new_usd = pd.read_csv(archive_path + 'df_usd'+ '_' + resolution +'.csv', parse
_dates=['snapshotTime']) # Data Restore
new_us500 = pd.read_csv(archive_path + 'df_us500'+ '_' + resolution +'.csv', p
arse_dates=['snapshotTime']) # Data Restore
new_us100 = pd.read_csv(archive_path + 'df_us100'+ '_' + resolution +'.csv', p
arse_dates=['snapshotTime']) # Data Restore
new_usoil = pd.read_csv(archive_path + 'df_usoil'+ '_' + resolution +'.csv', p
arse_dates=['snapshotTime']) # Data Restore
new_eur = pd.read_csv(archive_path + 'df_eur'+ '_' + resolution +'.csv', parse
_dates=['snapshotTime']) # Data Restore
new_ftse = pd.read_csv(archive_path + 'df_ftse'+ '_' + resolution +'.csv', par
se_dates=['snapshotTime']) # Data Restore
new_eurchn = pd.read_csv(archive_path + 'df_eurchn'+ '_' + resolution +'.csv',
parse_dates=['snapshotTime']) # Data Restore
new_usdchn = pd.read_csv(archive_path + 'df_usdchn'+ '_' + resolution +'.csv',
parse_dates=['snapshotTime']) # Data Restore

archive_path = 'data/'
df_xau = pd.read_csv(archive_path + 'df_xau'+ '_' + resolution +'.csv', parse_
dates=['snapshotTime']) # Data Restore
df_usd = pd.read_csv(archive_path + 'df_usd'+ '_' + resolution +'.csv', parse_
dates=['snapshotTime']) # Data Restore
df_us500 = pd.read_csv(archive_path + 'df_us500'+ '_' + resolution +'.csv', pa
rse_dates=['snapshotTime']) # Data Restore
df_us100 = pd.read_csv(archive_path + 'df_us100'+ '_' + resolution +'.csv', pa
rse_dates=['snapshotTime']) # Data Restore
df_usoil = pd.read_csv(archive_path + 'df_usoil'+ '_' + resolution +'.csv', pa
rse_dates=['snapshotTime']) # Data Restore
df_eur = pd.read_csv(archive_path + 'df_eur'+ '_' + resolution +'.csv', parse_
dates=['snapshotTime']) # Data Restore
df_ftse = pd.read_csv(archive_path + 'df_ftse'+ '_' + resolution +'.csv', pars
e_dates=['snapshotTime']) # Data Restore
```

```
# df_eurchn = pd.read_csv('df_eurchn'+ '_' + resolution +'.csv', parse_dates=
['snapshotTime']) # Data Restore
# df_usdchn = pd.read_csv('df_usdchn'+ '_' + resolution +'.csv', parse_dates=
['snapshotTime']) # Data Restore

# Backup/Restore Data ------------------------------------
# archive_path = 'data/'
# df_xau.to_csv('df_xau'+ '_' + resolution +'.csv',index=False,header = True)
# df_usd.to_csv('df_usd'+ '_' + resolution +'.csv',index=False,header = True)
# df_us500.to_csv('df_us500'+ '_' + resolution +'.csv',index=False,header = Tr
ue)
# df_us100.to_csv('df_us100'+ '_' + resolution +'.csv',index=False,header = Tr
ue)
# df_usoil.to_csv('df_usoil'+ '_' + resolution +'.csv',index=False,header = Tr
ue)
# df_eur.to_csv('df_eur'+ '_' + resolution +'.csv',index=False,header = True)
# df_ftse.to_csv('df_ftse'+ '_' + resolution +'.csv',index=False,header = Tru
e)
# df_eurchn.to_csv('df_eurchn'+ '_' + resolution +'.csv',index=False,header =
 True)
# df_usdchn.to_csv('df_usdchn'+ '_' + resolution +'.csv',index=False,header =
 True)
```

In [6]:
```
#       Dict extract openPrice  {'bid': 1275.64, 'ask': 1275.94, 'lastTraded':
None}
price_extractor(new_xau,'xau')
price_extractor(new_usd, 'usd')
price_extractor(new_us500, 'us500')
price_extractor(new_us100, 'us100')
price_extractor(new_usoil, 'usoil')
price_extractor(new_eur, 'eur')
price_extractor(new_ftse, 'ftse')
price_extractor(new_eurchn, 'eurchn')
price_extractor(new_usdchn, 'usdchn')


new_xau['price_change'] = new_xau['openPrice'] - new_xau['closePrice']
new_xau['price_maxmin'] = new_xau['highPrice'] - new_xau['lowPrice']
new_usd['price_change'] = new_usd['openPrice'] - new_usd['closePrice']
new_usd['price_maxmin'] = new_usd['highPrice'] - new_usd['lowPrice']
new_us500['price_change'] = new_us500['openPrice'] - new_us500['closePrice']
new_us500['price_maxmin'] = new_us500['highPrice'] - new_us500['lowPrice']
new_us100['price_change'] = new_us100['openPrice'] - new_us100['closePrice']
new_us100['price_maxmin'] = new_us100['highPrice'] - new_us100['lowPrice']
new_usoil['price_change'] = new_usoil['openPrice'] - new_usoil['closePrice']
new_usoil['price_maxmin'] = new_usoil['highPrice'] - new_usoil['lowPrice']
new_eur['price_change'] = new_eur['openPrice'] - new_eur['closePrice']
new_eur['price_maxmin'] = new_eur['highPrice'] - new_eur['lowPrice']
new_ftse['price_change'] = new_ftse['openPrice'] - new_ftse['closePrice']
new_ftse['price_maxmin'] = new_ftse['highPrice'] - new_ftse['lowPrice']
new_eurchn['price_change'] = new_eurchn['openPrice'] - new_eurchn['closePrice'
]
new_eurchn['price_maxmin'] = new_eurchn['highPrice'] - new_eurchn['lowPrice']
new_usdchn['price_change'] = new_usdchn['openPrice'] - new_usdchn['closePrice'
]
new_usdchn['price_maxmin'] = new_usdchn['highPrice'] - new_usdchn['lowPrice']

# zscore_fun_improved = lambda x: (x - x.rolling(window=200, min_periods=20).m
ean())\
# / x.rolling(window=200, min_periods=20).std()
# features['f10'] =prices.groupby(level='symbol').close.apply(zscore_fun_impro
ved)
# features.f10.unstack().plot.kde(title='Z-Scores (accurate)')
```

In [7]:
```
# Merge old and new
df_xau = df_xau.append(new_xau, ignore_index= True)
df_usd = df_usd.append(new_usd, ignore_index= True)
df_us500 = df_us500.append(new_us500, ignore_index= True)
df_us100 = df_us100.append(new_us100, ignore_index= True)
df_usoil = df_usoil.append(new_usoil, ignore_index= True)
df_eur = df_eur.append(new_eur, ignore_index= True)
df_ftse = df_ftse.append(new_ftse, ignore_index= True)
# df_eurchn = df_eurchn.append(new_eurchn, ignore_index= True)
# df_usdchn = df_usdchn.append(new_usdchn, ignore_index= True)
```

In [27]:
```
# Last record check
print(df_xau.iloc[-1,0])
print(df_usd.iloc[-1,0])
print(df_us500.iloc[-1,0])
print(df_us100.iloc[-1,0])
print(df_usoil.iloc[-1,0])
print(df_eur.iloc[-1,0])
print(df_ftse.iloc[-1,0])
```

```
2020-04-27 19:00:00
2020-04-27 19:00:00
2020-04-27 19:00:00
2020-04-27 19:00:00
2020-04-27 19:00:00
2020-04-27 19:00:00
2020-04-27 19:00:00
```

Feature Dataframe Prep

In [31]:
```
df_prices = pd.merge(df_xau, df_usd, on='snapshotTime', how = 'left', suffixes
=('','_usd'))
df_prices = pd.merge(df_prices, df_us500, on='snapshotTime', how = 'left', suf
fixes=('','_us500'))
df_prices = pd.merge(df_prices, df_us100, on='snapshotTime', how = 'left', suf
fixes=('','_us100'))
df_prices = pd.merge(df_prices, df_usoil, on='snapshotTime', how = 'left', suf
fixes=('','_usoil'))
df_prices = pd.merge(df_prices, df_eur, on='snapshotTime', how = 'left', suffi
xes=('','_eur'))
df_prices = pd.merge(df_prices, df_ftse, on='snapshotTime', how = 'left', suff
ixes=('','_ftse'))
# df_prices = pd.merge(df_prices, df_eurchn, on='snapshotTime', how = 'left',
 suffixes=('','_eurchn'))
# df_prices = pd.merge(df_prices, df_usdchn, on='snapshotTime', how = 'left',
 suffixes=('','_usdchn'))

df_prices = df_prices[[
                    'snapshotTime','openPrice','closePrice','price_change','pr
ice_maxmin','lastTradedVolume',
                    'openPrice_usd','closePrice_usd','price_change_usd','price
_maxmin_usd','lastTradedVolume_usd',
                    'openPrice_us500','closePrice_us500','price_change_us500',
'price_maxmin_us500','lastTradedVolume_us500',
                    'openPrice_us100','closePrice_us100','price_change_us100',
'price_maxmin_us100','lastTradedVolume_us100',
                    'openPrice_usoil','closePrice_usoil','price_change_usoil',
'price_maxmin_usoil','lastTradedVolume_usoil',
                    'openPrice_eur','closePrice_eur','price_change_eur','price
_maxmin_eur','lastTradedVolume_eur',
                    'openPrice_ftse','closePrice_ftse','price_change_ftse','pr
ice_maxmin_usd','lastTradedVolume_ftse',
                    # 'openPrice_eurchn','closePrice_eurchn','price_change_eur
chn','price_maxmin_eurchn','lastTradedVolume_eurchn',
                    # 'openPrice_usdchn','closePrice_usdchn','price_change_usd
chn','price_maxmin_usdchn','lastTradedVolume_usdchn',
                    ]]

# df_prices.reset_index(inplace = True)
# NaNs
# features_price = features_price.iloc[100:, :]
df_prices.fillna(method = 'ffill' ,axis = 0,inplace = True)
df_prices.fillna(method = 'bfill' ,axis = 0,inplace = True)
df_prices.sort_values('snapshotTime', inplace = True)
df_prices.reset_index(inplace = True)

archive_path = 'data/'
df_prices.to_csv(archive_path + 'gold_feature_price'+ '_' + resolution +'.csv'
,index=False,header = True)
# df_prices = pd.read_csv('gold_feature_price'+ '_' + resolution +'.csv', pars
e_dates=['snapshotTime']) # Data Restore
```

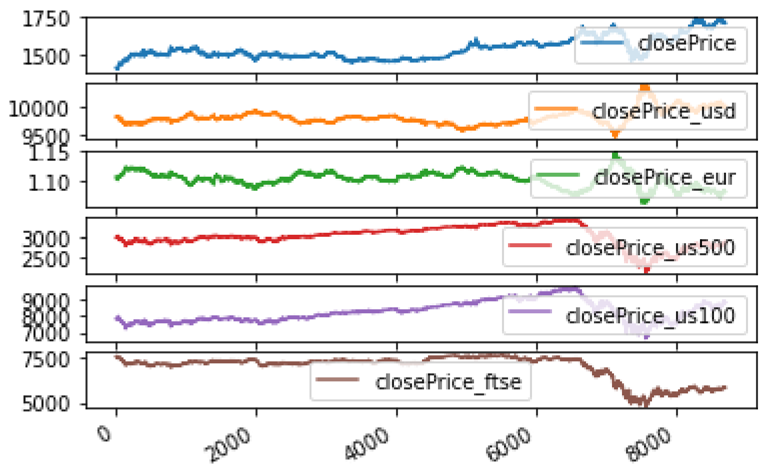
In [57]:
```
# feature prep
features_price = df_prices.copy()

# Feature Selection
datetime_price = features_price[['snapshotTime']]
features_price = features_price[['closePrice','closePrice_usd','closePrice_eu
r','closePrice_usoil','closePrice_us500','closePrice_us100','closePrice_ftse'
]] #,'closePrice_eurchn']
# features_price = features_price[['price_change','price_change_usd','price_ch
ange_us500','price_change_usoil', 'price_change_eur']]

# Feature Preview
features_price.plot(subplots=True)
# plt.show()
```

```
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:298: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().rowspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:298: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().colspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:304: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().rowspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:304: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().colspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:298: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().rowspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:298: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().colspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:304: MatplotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().rowspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\pandas\plotting\_matplot
lib\tools.py:304: MatplotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two
minor releases later. Use ax.get_subplotspec().colspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
```

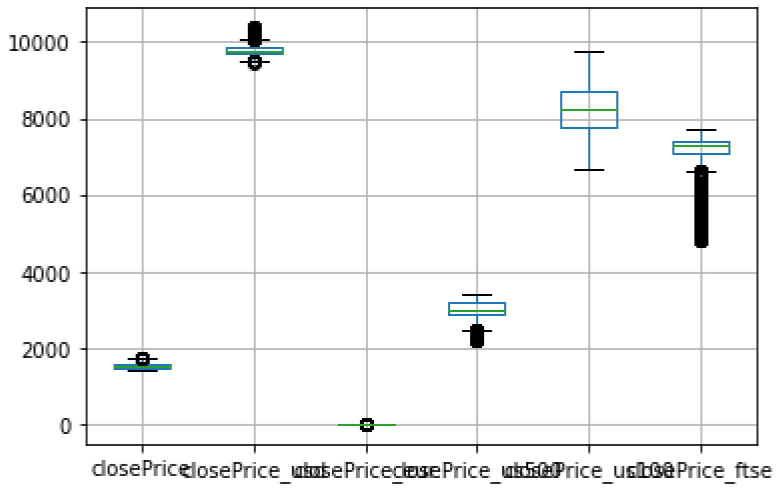
```
Out[57]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x00000184E2DB9EF0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x00000184E2DF8748>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x00000184E2E2A940>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x00000184E2E5DB70>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x00000184E2E90DA0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x00000184E2EC4FD0>],
               dtype=object)
```



Some Visualization

In [62]:
```
# sns.set_context("talk")
# plt.figure(figsize=(12, 6))
# g = sns.boxplot(x="timezone", y="delay_d", , data=df_prices, ) #  hue="origi
n", kind="swarm",  jitter=False, #
# g.set(xlabel='Timezone', ylabel='Departure Delay')
features_price.boxplot(list(features_price.columns))
```

Out[62]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x184e2fb9978>
```

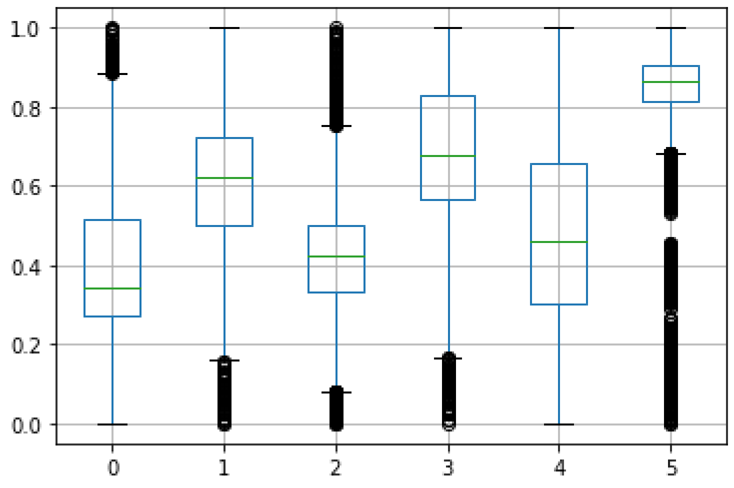


In [63]:
```
# Split test/training
counter = 7613
split_point1 = 7400
split_point2 = 7600
features_matrix = features_price.values.astype('float32')
train_price = features_matrix[:split_point1,:]
test_price = features_matrix[split_point1:,:]
# test_price = features_matrix[split_point2:,]
scaler, train_scaled, test_scaled = scale(train_price,test_price)

print(train_price.shape, test_price.shape)
df_train = pd.DataFrame(train_scaled)
df_train.boxplot(list(df_train.columns))
```

```
(7400, 6) (1307, 6)
```

Out[63]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x184e2b747b8>
```



Linear Regression Coefficients. 1 is bad

In [60]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
X_train, y_train = train_scaled[:,1:], train_scaled[:,0:1]
regressor.fit(X_train, y_train)
r2 = regressor.score(X_train,y_train)
print('accuracy:',r2,'Adj-R2:',adj_r2(X_train,r2))
```

```
accuracy: 0.7518317135408443 Adj-R2: 0.751663896198094
```

In [61]:
```
# Create a regression summary where we can compare them with one-another
reg_summary = pd.DataFrame(features_price.columns.values[1:], columns=['Featur
es'])
reg_summary['Coefs'] = regressor.coef_[0]
reg_summary['Weights^2'] = np.exp(np.abs(regressor.coef_[0]))
reg_summary.sort_values('Weights^2',ascending=False)
```

Out[61]:

	Features	Coefs	Weights^2
2	closePrice_us500	-3.032185	20.742508
3	closePrice_us100	2.048027	7.752591
1	closePrice_eur	-1.512359	4.537422
0	closePrice_usd	-1.342796	3.829737
4	closePrice_ftse	0.501982	1.651992