In [56]:

```
import matplotlib.pyplot as plt

import numpy as np
import time
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import numpy

np.set_printoptions(suppress=True)
%matplotlib inline

from sklearn.preprocessing import MinMaxScaler
from pylab import *
mpl.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
import seaborn as sns
sns.set_palette("husl") #设置所有图的颜色，使用hls色彩空间
import numpy as np
from sklearn import metrics
from sklearn.metrics import mean_squared_error #均方误差
from sklearn.metrics import mean_absolute_error #平方绝对误差

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.svm import SVR
```

In [2]:

```
# 读取BCHAIN-MKPRU.csv文件
df_BM = pd.read_csv('BCHAIN-MKPRU.csv', encoding='utf-8')
# 读取LBMA-GOLD.csv文件
df_GOLD = pd.read_csv('LBMA-GOLD.csv', encoding='utf-8')

# 通过简单的观察，采用Date字段合并数据，采用merge连接
df = pd.merge(df_BM, df_GOLD, how='outer', on='Date')
# 格式化时间 文件的时间格式是 月/日/年 -> 20xx/月/日
df["Date"] = df["Date"].apply(lambda x: time.strftime('%Y/%m/%d %H:%M:%S', time.strptime(x, '%m/%d/
# 排序Date字段
df = df.sort_values("Date")
df
```

Out[2]:

	Date	Value	USD (PM)
0	2016/09/11 00:00:00	621.65	NaN
1	2016/09/12 00:00:00	609.67	1324.60
2	2016/09/13 00:00:00	610.92	1323.65
3	2016/09/14 00:00:00	608.82	1321.75
4	2016/09/15 00:00:00	610.38	1310.80
...	...	...	...
1821	2021/09/06 00:00:00	51769.06	1821.60
1822	2021/09/07 00:00:00	52677.40	1802.15
1823	2021/09/08 00:00:00	46809.17	1786.00
1824	2021/09/09 00:00:00	46078.38	1788.25
1825	2021/09/10 00:00:00	46368.69	1794.60

1826 rows × 3 columns

In [3]:

```
#查看缺失值
#比特币可以每天交易，但黄金仅在开市日交易
df.isnull().sum()
```

Out[3]:

```
Date           0
Value          0
USD (PM)     571
dtype: int64
```

In [4]:

```
df.columns=['Date','BCHAIN-MKPRU','GOLD']
```

In [5]:

```
#计算日收益率
df['GOLD_fillna']=df['GOLD'].fillna(method='bfill')
```

In [6]:

```
# 计算每日收益率
stock_data = df[['BCHAIN-MKPRU','GOLD_fillna']].pct_change()
# 打印前5行数据
stock_data.head()
```

Out[6]:

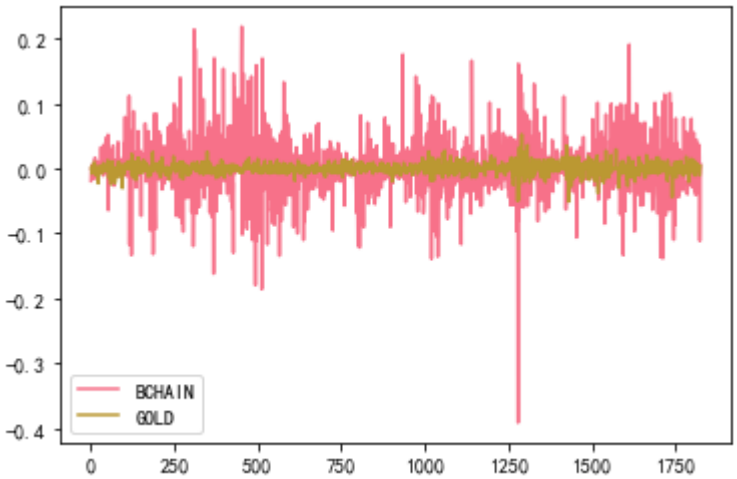
	BCHAIN-MKPRU	GOLD_fillna
0	NaN	NaN
1	-0.019271	0.000000
2	0.002050	-0.000717
3	-0.003437	-0.001435
4	0.002562	-0.008284

In [7]:

```
df['BCHAIN日收益率']=stock_data['BCHAIN-MKPRU']
df['GOLD日收益率']=stock_data['GOLD_fillna']
```

In [8]:

```
df['BCHAIN日收益率'].plot(label='BCHAIN')
df['GOLD日收益率'].plot(label='GOLD')
plt.legend()
plt.show()
```



In [9]:

```
df
```

Out[9]:

	Date	BCHAIN-MKPRU	GOLD	GOLD_fillna	BCHAIN日收益率	GOLD日收益率
0	2016/09/11 00:00:00	621.65	NaN	1324.60	NaN	NaN
1	2016/09/12 00:00:00	609.67	1324.60	1324.60	-0.019271	0.000000
2	2016/09/13 00:00:00	610.92	1323.65	1323.65	0.002050	-0.000717
3	2016/09/14 00:00:00	608.82	1321.75	1321.75	-0.003437	-0.001435
4	2016/09/15 00:00:00	610.38	1310.80	1310.80	0.002562	-0.008284
...	...	...	...	...	...	...
1821	2021/09/06 00:00:00	51769.06	1821.60	1821.60	0.036472	0.000000
1822	2021/09/07 00:00:00	52677.40	1802.15	1802.15	0.017546	-0.010677
1823	2021/09/08 00:00:00	46809.17	1786.00	1786.00	-0.111399	-0.008962
1824	2021/09/09 00:00:00	46078.38	1788.25	1788.25	-0.015612	0.001260
1825	2021/09/10 00:00:00	46368.69	1794.60	1794.60	0.006300	0.003551

1826 rows × 6 columns

In [10]:

```
#时序数据滑窗转换用于将时间序列数据转为回归数据，简单地说，就是把一个单序列的数据变为X->Y的回归数据。

#步阶为2代表2个X（步阶多少就有多少个X），一个Y（这个不会变的），

#简单地说，就是用第1，2天的数据预测第3天，用第2，3天的数据预测第4天，以此类推。
def create_dataset(dataset, look_back):

    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(dataset[i + look_back])
        dddd=pd.concat([pd.DataFrame(np.array(dataY)),pd.DataFrame(np.array(dataX))],axis=1)
        dddd.columns=['Y']+['shiftX_'+str(i) for i in range(len(dddd.columns)-1)]
    return dddd
data1=create_dataset(df_BM['Value'],look_back=1)
data1
```

Out[10]:

	Y	shiftX_0
0	609.67	621.65
1	610.92	609.67
2	608.82	610.92
3	610.38	608.82
4	609.11	610.38
...	...	...
1819	49947.38	50035.33
1820	51769.06	49947.38
1821	52677.40	51769.06
1822	46809.17	52677.40
1823	46078.38	46809.17

1824 rows × 2 columns

In [11]:

```
'''
随机森林回归
'''

from sklearn.metrics import r2_score


def score(y_true, y_pre):
    # MSE
    print("MSE :")
    print(metrics.mean_squared_error(y_true, y_pre))
    # RMSE
    print("RMSE :")
    print(np.sqrt(metrics.mean_squared_error(y_true, y_pre)))
    # MAE
    print("MAE :")
    print(metrics.mean_absolute_error(y_true, y_pre))
    # R2
    print("R2 :")
    print(r2_score(y_true,y_pre))
```

In [12]:

```
from sklearn.model_selection import train_test_split
X=data1['shiftX_0'].values.reshape(-1, 1)
Y=data1['Y'].values.reshape(-1, 1)

train_X,test_X,train_y,test_y = train_test_split(X,Y,test_size=0.3,random_state=5)


model_rf = RandomForestRegressor()
model_rf.fit(train_X,train_y)

score(test_y, model_rf.predict(test_X))
```

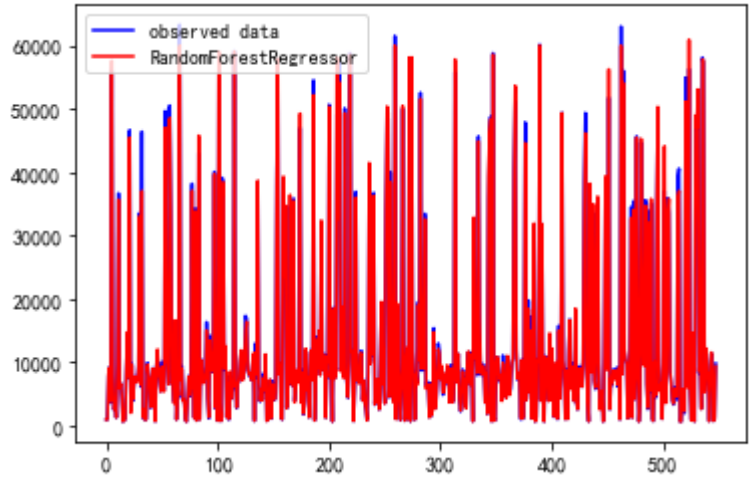
```
MSE :
1185882.4862585713
RMSE :
1088.9823167795569
MAE :
501.48320266886157
R2 :
0.9946835496818754

c:\users\stream\appdata\local\programs\python\python37\lib\site-packages\ipykernel_l
auncher.py:9: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples,), for example using ravel().
  if __name__ == '__main__':
```

In [13]:

```
plt.plot(test_y,color='blue', label='observed data')
plt.plot(model_rf.predict(test_X), color='red', label='RandomForestRegressor')

plt.legend() # 显示图例
plt.show()
```



In [15]:

```
df_BM['Value']=pd.DataFrame(model_rf.predict(df_BM['Value'].values.reshape(-1, 1)),columns=['Value']
```

In [17]:

```
data1=create_dataset(df_GOLD['USD (PM)'],look_back=1)
data1.dropna(inplace=True)
data1
```

Out[17]:

	Y	shiftX_0
0	1323.65	1324.60
1	1321.75	1323.65
2	1310.80	1321.75
3	1308.35	1310.80
4	1314.85	1308.35
...	...	...
1258	1823.70	1812.55
1259	1821.60	1823.70
1260	1802.15	1821.60
1261	1786.00	1802.15
1262	1788.25	1786.00

1243 rows × 2 columns

In [18]:

```
X=data1['shiftX_0'].values.reshape(-1, 1)
Y=data1['Y'].values.reshape(-1, 1)

train_X,test_X,train_y,test_y = train_test_split(X,Y,test_size=0.3,random_state=5)


model_rf = RandomForestRegressor()
model_rf.fit(train_X,train_y)

score(test_y, model_rf.predict(test_X))
```

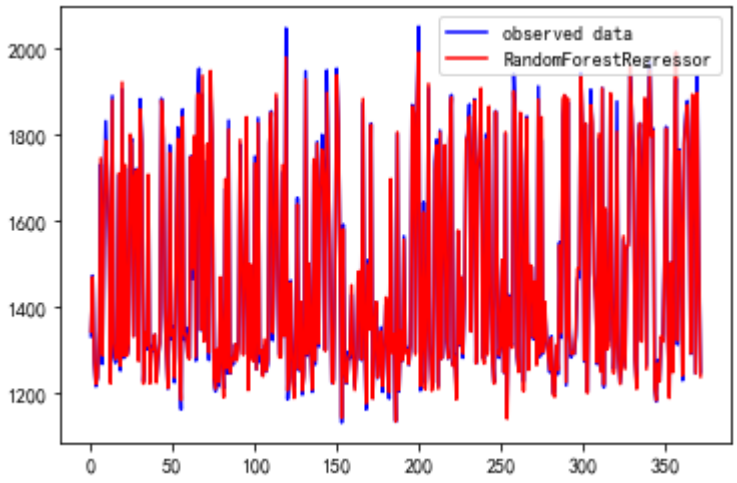
```
MSE :
250.96990976091644
RMSE :
15.842029849767247
MAE :
11.162902052214955
R2 :
0.9960219789992713

c:\users\stream\appdata\local\programs\python\python37\lib\site-packages\ipykernel_l
auncher.py:8: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples,), for example using ravel().
```

In [19]:

```
plt.plot(test_y,color='blue', label='observed data')
plt.plot(model_rf.predict(test_X), color='red', label='RandomForestRegressor')

plt.legend() # 显示图例
plt.show()
```



In [21]:

```
df_GOLD['USD (PM)']=pd.DataFrame(model_rf.predict(df_GOLD['USD (PM)'].fillna(method='bfill').values.
```

In [22]:

```
# 通过简单的观察，采用Date字段合并数据，采用merge连接
df = pd.merge(df_BM, df_GOLD, how='outer', on='Date')
# 格式化时间 文件的时间格式是 月/日/年 -> 20xx/月/日
df["Date"] = df["Date"].apply(lambda x: time.strftime('%Y/%m/%d %H:%M:%S', time.strptime(x, '%m/%d/
# 排序Date字段
df = df.sort_values("Date")
df.fillna(method='bfill',inplace=True)
df
```

Out[22]:

	Date	Value	USD (PM)
0	2016/09/11 00:00:00	612.1401	1323.030000
1	2016/09/12 00:00:00	611.3704	1323.030000
2	2016/09/13 00:00:00	609.1856	1322.171500
3	2016/09/14 00:00:00	603.1386	1314.841000
4	2016/09/15 00:00:00	609.0992	1318.835133
...	...	...	...
1821	2021/09/06 00:00:00	53440.6227	1833.983500
1822	2021/09/07 00:00:00	48720.4905	1795.723000
1823	2021/09/08 00:00:00	46338.0745	1799.757500
1824	2021/09/09 00:00:00	45160.0773	1789.106000
1825	2021/09/10 00:00:00	45419.3164	1791.323500

1826 rows × 3 columns

In [23]:

```
from sko.operators import ranking, selection, crossover, mutation
from sko.GA import GA
```

In [24]:

```
# 黄金持有
z1 = 500
# 比特币持有
z2 = 500
# m1 今日黄金价格
m1 = 1324.2576

# m2 今日比特币价格
m2 = 773.88040
# n1 明日黄金价格
n1 = 1323.3617
# n2 明日比特币价格
n2 = 774.83980
```

In [25]:

```
def model_1_ga(z1,z2,m1,m2,n1,n2):
    demo_func = lambda x: -((n1 / m1) * (z1 + x[0]) + (n2 / m2) * (z2 + x[1]))
    constraint_ueq = [
    lambda x: x[0] + x[1] - (z1 + z2),
    lambda x: (-z1 - z2) - (x[0] + x[1]),
    lambda x: 0.01 - ((n1 / m1) - 1),
    lambda x: 0.02 - ((n2 / m2) - 1)
    ]
    ga = GA(func=demo_func, n_dim=2, size_pop=100, max_iter=500, prob_mut=0.001,
    lb=[-z1, -z2], ub=[z1, z2], precision=[1e-7, 1e-7], constraint_ueq=constraint_ueq)
    ga.register(operator_name='selection', operator=selection.selection_roulette_2)
    ga.register(operator_name='ranking', operator=ranking.ranking). \
    register(operator_name='crossover', operator=crossover.crossover_2point_bit). \
    register(operator_name='mutation', operator=mutation.mutation)
    best_x, func = ga.run()

    return -func[0],best_x[0],best_x[1]
model_1_ga(z1,z2,m1,m2,n1,n2)
```

Out[25]:

```
(2000.132708889963, 499.7572267893561, 499.81235404264953)
```

In [26]:

```
%%time


def model_2_ga(z1,z2,n1,n2):

    demo_func = lambda x: -(z1 + (n2 / m2) * (z2 + x[0]))


    constraint_ueq = [
        lambda x: x[0] - z2,
        lambda x: - z2 - x[0],
        lambda x: 0.02  - ((n2 / m2) - 1)
    ]
    ga = GA(func=demo_func, n_dim=1, size_pop=100, max_iter=500, prob_mut=0.001,
            lb=[-z1], ub=[z1], precision=[1e-7], constraint_ueq=constraint_ueq)

    ga.register(operator_name='selection', operator=selection.selection_roulette_2)
    ga.register(operator_name='ranking', operator=ranking.ranking). \
        register(operator_name='crossover', operator=crossover.crossover_2point_bit). \
        register(operator_name='mutation', operator=mutation.mutation)
    best_x, func = ga.run()
    return -func[0],best_x[0]

model_2_ga(z1,z2,n1,n2)
```

```
Wall time: 790 ms
```

Out[26]:

```
(1501.2394551561126, 499.99972904333845)
```

In [28]:

```
# 读取BCHAIN-MKPRU.csv文件
df_BM = pd.read_csv('BCHAIN-MKPRU.csv', encoding='utf-8')
# 读取LBMA-GOLD.csv文件
df_GOLD = pd.read_csv('LBMA-GOLD.csv', encoding='utf-8')

# 通过简单的观察，采用Date字段合并数据，采用merge连接
newdf = pd.merge(df_BM, df_GOLD, how='outer', on='Date')
# 格式化时间 文件的时间格式是 月/日/年 -> 20xx/月/日
newdf["Date"] = newdf["Date"].apply(lambda x: time.strftime('%Y/%m/%d %H:%M:%S', time.strptime(x, '
# 排序Date字段
newdf = newdf.sort_values("Date")
newdf

# 通过简单的观察，采用Date字段合并数据，采用merge连接
df___ = pd.merge(df, newdf, how='left', on='Date')

df___
```

Out[28]:

	Date	Value_x	USD (PM)_x	Value_y	USD (PM)_y
0	2016/09/11 00:00:00	612.1401	1323.030000	621.65	NaN
1	2016/09/12 00:00:00	611.3704	1323.030000	609.67	1324.60
2	2016/09/13 00:00:00	609.1856	1322.171500	610.92	1323.65
3	2016/09/14 00:00:00	603.1386	1314.841000	608.82	1321.75
4	2016/09/15 00:00:00	609.0992	1318.835133	610.38	1310.80
...	...	...	...	...	...
1821	2021/09/06 00:00:00	53440.6227	1833.983500	51769.06	1821.60
1822	2021/09/07 00:00:00	48720.4905	1795.723000	52677.40	1802.15
1823	2021/09/08 00:00:00	46338.0745	1799.757500	46809.17	1786.00
1824	2021/09/09 00:00:00	45160.0773	1789.106000	46078.38	1788.25
1825	2021/09/10 00:00:00	45419.3164	1791.323500	46368.69	1794.60

1826 rows × 5 columns

In [ ]:

```
# 黄金持有
z1 = 500
# 比特币持有
z2 = 1000-z1
alldata=[]
for i in range(df.shape[0]):
    try:
        if i== df.shape[0]-1:
            break
        # m1 真实的今日黄金价格
        m1=df___['USD (PM)_y'].iloc[i]
        # m2 真实的今日比特币价格
        m2=df___['Value_y'].iloc[i]
        # n1 预测的明日黄金价格
        n1= df___['USD (PM)_x'].iloc[i+1]
        # n2 预测的明日比特币价格
        n2= df___['Value_x'].iloc[i+1]
         # n1 真实的明日黄金价格
        N1= df___['USD (PM)_y'].iloc[i+1]
        # n2 真实的明日比特币价格
        N2= df___['Value_y'].iloc[i+1]

        if (np.isnan(m1)==True) or (np.isnan(n1)==True):
            ojb,x_bit=model_2_ga(z1,z2,m2,n2)
            z2=N2 / m2 * (z2 + x_bit)
            alldata.append([df___['Date'].iloc[i],m1,m2,n1,n2,N1,N2,z1,z2,np.nan,x_bit,ojb])
        else:
            ojb,x_hj,x_bit=model_1_ga(z1,z2,m1,m2,n1,n2)
            z1=N1 / m1 * (z1 + x_hj)
            z2=N2 / m2 * (z2 + x_bit)
            alldata.append([df___['Date'].iloc[i],m1,m2,n1,n2,N1,N2,z1,z2,x_hj,x_bit,ojb])
        if z1==0 or np.isnan(z1)==True :
            z1=0.0001

    except:
        print(df___['Date'].iloc[i])
        break
```

In [48]:

```
columns1=['交易日期','真实的今日黄金价格','真实的今日比特币价格','预测的明日黄金价格','预测的明日比特
alldata=pd.DataFrame(alldata,columns=columns1)
```

In [176]:

```
alldata.to_excel('alldata.xlsx',index=None)
```

In [ ]:

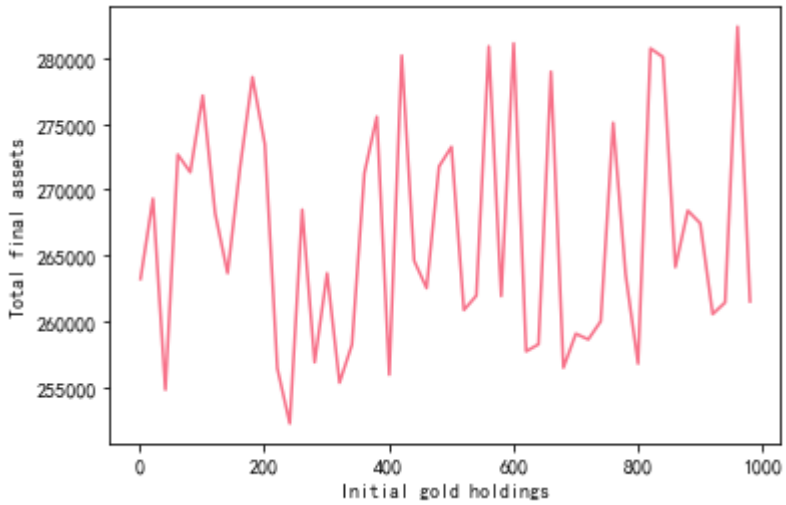
```
#迭代以上流程
x=[]
y=[]
for i in range(1,1001,20):
    x.append(i)
    # 黄金持有
    z1 = i
    # 比特币持有
    z2 = 1000-z1
    alldata=[]
    for i in range(df.shape[0]):
        try:
            if i== df.shape[0]-1:
                break
            # m1 真实的今日黄金价格
            m1=df___['USD (PM)_y'].iloc[i]
            # m2 真实的今日比特币价格
            m2=df___['Value_y'].iloc[i]
            # n1 预测的明日黄金价格
            n1= df___['USD (PM)_x'].iloc[i+1]
            # n2 预测的明日比特币价格
            n2= df___['Value_x'].iloc[i+1]
             # n1 真实的明日黄金价格
            N1= df___['USD (PM)_y'].iloc[i+1]
            # n2 真实的明日比特币价格
            N2= df___['Value_y'].iloc[i+1]

            if (np.isnan(m1)==True) or (np.isnan(n1)==True):
                ojb,x_bit=model_2_ga(z1,z2,m2,n2)
                z2=N2 / m2 * (z2 + x_bit)

            else:
                ojb,x_hj,x_bit=model_1_ga(z1,z2,m1,m2,n1,n2)
                z1=N1 / m1 * (z1 + x_hj)
                z2=N2 / m2 * (z2 + x_bit)
            if z1==0 or np.isnan(z1)==True :
                z1=0.0001
        except:
            print(df___['Date'].iloc[i])
            break
        y.append(ojb)
```

In [64]:

```
plt.plot(x,y)
plt.xlabel('Initial gold holdings')
plt.ylabel('Total final assets')
plt.show()
```



In [55]:

```

```