In [1]:
```
# API lib

import requests
import base64
import json
from math import sqrt
import numpy as np
from numpy import concatenate
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
```

```
Using TensorFlow backend.
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:493: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:494: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:495: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:496: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:497: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\mjvaf\Anaconda3\envs\TFG1\lib\site-packages\tensorflow\python\framew
ork\dtypes.py:502: FutureWarning: Passing (type, 1) or '1type' as a synonym o
f type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [2]:
```
# Test GPU
from tensorflow.python.client import device_lib
import tensorflow as tf
print(device_lib.list_local_devices())
print(tf.test.is_built_with_cuda())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 17899341883119177496
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 3231462195
locality {
  bus_id: 1
}
incarnation: 6778468820053523769
physical_device_desc: "device: 0, name: GeForce GTX 950M, pci bus id: 0000:0
1:00.0, compute capability: 5.0"
]
True
```

Init

In [3]:
```
# 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

# from Crypto.PublicKey import RSA
# from Crypto.Cipher import PKCS1_v1_5

# login_url = "https://demo-api.ig.com/gateway/deal/"

# data=json.dumps({
# 'encryptedPassword': False,
# 'identifier': identifier,
# 'password': password
# })


# x = requests.post(login_url, data = data)
# print(x.text)


# m_data = r.json()

# decoded = base64.b64decode(m_data['encryptionKey'])
# rsakey = RSA.importKey(decoded)
# message = password + '|' + str(long(m_data['timeStamp']))
# input = base64.b64encode(message)
# encryptedPassword = base64.b64encode(PKCS1_v1_5.new(rsakey).encrypt(input))

# session = "/session"
# m_url = url + session
# headers = { "Content-Type": "application/json; charset=utf-8",
# "Accept": "application/json; charset=utf-8",
# "X-IG-API-KEY": m_apiKey,
# "Version": "2"
# }

# payload = json.dumps({ "identifier": identifier,
# "password": encryptedPassword,
# "encryptedPassword": True
# })

# # In[]
# r = requests.post(m_url, data=payload, headers=headers)
# r.status_code
# print r.status_code
# print r.text

pd.options.display.max_columns = None


# start_session_init()
```

```
date0 = '2019-05-01T00%3A00%3A00'
date1 = '2019-08-01T00%3A00%3A00'
date2 = '2020-04-22T23%3A59%3A59'
# 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'
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'
usoil_epic = 'CC.D.CL.UNC.IP'
```

Functions

In [4]:
```
def start_session():
    url = "https://demo-api.ig.com/gateway/deal"
    session = "/session/encryptionKey"
    m_url = url + session
    return  requests.get(m_url, headers=headers)

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=headers)

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'))

def ts(new_data, look_back = 100, pred_col = 1):
    t = new_data.copy()
    t['id'] = range(1, len(t)+1)
    t = t.iloc[:-look_back,:]
    t.set_index('id', inplace= True)
    pred_value = new_data.copy()
    pred_value = pred_value.iloc[look_back:, pred_col]
    pred_value.columns = ['Pred']
    pred_value = pd.DataFrame(pred_value)
    pred_value['id'] = range(1,len(pred_value)+1)
    pred_value.set_index('id', inplace= True)
    final_df= pd.concat([t,pred_value],axis=1)
    return final_df



# 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)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
        return yhat + history[-interval]

# scale train and test data to [-1, 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]

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
        n_vars = 1 if type(data) is list else data.shape[1]
        df = pd.DataFrame(data)
        cols, names = list(), list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
                names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df.shift(-i))
                if i == 0:
                        names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                else:
                        names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_
vars)]
        # put it all together
        agg = pd.concat(cols, axis=1)
        agg.columns = names
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg

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 [ ]:
```
# 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']
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']
usoil = price_history(usoil_epic,resolution,date1,date2)
usoil_list = json.loads(usoil.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']
# DataFrame convert
df_xau = pd.DataFrame(xau_list)
df_usd = pd.DataFrame(usd_list)
df_us500 = pd.DataFrame(us500_list)
df_us100 = pd.DataFrame(us100_list)
df_eur = pd.DataFrame(eur_list)
df_ftse = pd.DataFrame(ftse_list)
df_usoil = pd.DataFrame(usoil_list)
# df_eurchn = pd.DataFrame(eurchn_list)
# df_usdchn = pd.DataFrame(usdchn_list)


# Give it a name
# df_xau.name = 'xau'
# df_eur.name = 'usd'
# df_us500.name = 'us500'
# df_us100.name = 'us100'
# df_eur.name = 'eur'
tables = ['xau','usd','us500','us100','usoil','eur','ftse','eurchn','usdchn']

#       Dict extract openPrice  {'bid': 1275.64, 'ask': 1275.94, 'lastTraded': None}
price_extractor(df_xau,'xau')
price_extractor(df_usd, 'usd')
price_extractor(df_us500, 'us500')
price_extractor(df_us100, 'us100')
price_extractor(df_usoil, 'usoil')
price_extractor(df_eur, 'eur')
price_extractor(df_ftse, 'ftse')
# price_extractor(df_eurchn, 'eurchn')
# price_extractor(df_usdchn, 'usdchn')


df_xau['price_change'] = df_xau['openPrice'] - df_xau['closePrice']
df_xau['price_maxmin'] = df_xau['highPrice'] - df_xau['lowPrice']
df_usd['price_change'] = df_usd['openPrice'] - df_usd['closePrice']
df_usd['price_maxmin'] = df_usd['highPrice'] - df_usd['lowPrice']
```

```
df_us500['price_change'] = df_us500['openPrice'] - df_us500['closePrice']
df_us500['price_maxmin'] = df_us500['highPrice'] - df_us500['lowPrice']
df_us100['price_change'] = df_us100['openPrice'] - df_us100['closePrice']
df_us100['price_maxmin'] = df_us100['highPrice'] - df_us100['lowPrice']
df_usoil['price_change'] = df_usoil['openPrice'] - df_usoil['closePrice']
df_usoil['price_maxmin'] = df_usoil['highPrice'] - df_usoil['lowPrice']
df_eur['price_change'] = df_eur['openPrice'] - df_eur['closePrice']
df_eur['price_maxmin'] = df_eur['highPrice'] - df_eur['lowPrice']
df_ftse['price_change'] = df_ftse['openPrice'] - df_ftse['closePrice']
df_ftse['price_maxmin'] = df_ftse['highPrice'] - df_ftse['lowPrice']
# df_eurchn['price_change'] = df_eurchn['openPrice'] - df_eurchn['closePrice']
# df_eurchn['price_maxmin'] = df_eurchn['highPrice'] - df_eurchn['lowPrice']
# df_usdchn['price_change'] = df_usdchn['openPrice'] - df_usdchn['closePrice']
# df_usdchn['price_maxmin'] = df_usdchn['highPrice'] - df_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)')
```

Backup / Restore DataFrames

In [5]:
```
# Backup/Restore 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.csv',index=False,header = True)
# df_usdchn.to_csv('df_usdchn.csv',index=False,header = True)

df_xau = pd.read_csv('df_xau'+ '_' + resolution +'.csv', parse_dates=['snapsho
tTime']) # Data Restore
df_usd = pd.read_csv('df_usd'+ '_' + resolution +'.csv', parse_dates=['snapsho
tTime']) # Data Restore
df_us500 = pd.read_csv('df_us500'+ '_' + resolution +'.csv', parse_dates=['sna
pshotTime']) # Data Restore
df_us100 = pd.read_csv('df_us100'+ '_' + resolution +'.csv', parse_dates=['sna
pshotTime']) # Data Restore
df_usoil = pd.read_csv('df_usoil'+ '_' + resolution +'.csv', parse_dates=['sna
pshotTime']) # Data Restore
df_eur = pd.read_csv('df_eur'+ '_' + resolution +'.csv', parse_dates=['snapsho
tTime']) # Data Restore
df_ftse = pd.read_csv('df_ftse'+ '_' + resolution +'.csv', parse_dates=['snaps
hotTime']) # 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
```

Feature Matrix Prep

In [6]:
```
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)
df_prices.to_csv('gold_feature_price'+ '_' + resolution +'.csv',index=False,he
ader = True)
# df_prices = pd.read_csv('gold_feature_price'+ '_' + resolution +'.csv', pars
e_dates=['snapshotTime']) # Data Restore
```

In [7]:
```
# feature prep
features_price = df_prices.copy()

# NaNs
# features_price = features_price.iloc[100:, :]
features_price.dropna(axis = 0, inplace = True)
features_price = features_price.sort_values('snapshotTime')
features_price.reset_index(inplace = True)

#

features_price = features_price[['closePrice','closePrice_usd','closePrice_eu
r','closePrice_usoil']] #,'closePrice_us500','closePrice_ftse','closePrice_fts
e','closePrice_eurchn']
# features_price = features_price[['price_change','price_change_usd','price_ch
ange_us500','price_change_usoil', 'price_change_eur']]

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

Out[7]:
```
(7613, 4)
```

In [29]:
```
# 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)
```

```
(7400, 4) (213, 4)
```

Linear Regression Coefficients. 1 is bad

In [30]:
```
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: 0.731890779168325 Adj-R2: 0.7317820274562516
```

In [10]:
```
# 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[10]:

	Features	Coefs	Weights^2
0	closePrice_usd	-2.470518	11.828568
1	closePrice_eur	-1.771792	5.881385
2	closePrice_usoil	-0.992817	2.698825

RNN Matrix prep

In [31]:
```
# The Scale
rnn_time_steps = 48
n_features = train_price.shape[1]
print(n_features)
scaler, train_scaled, test_scaled = scale(train_price,test_price)
print(train_scaled.shape)
```

```
4
(7400, 4)
```

Matrix Reformation methods

In [32]:
```
# One feature method
# gold_price_scaled = scaled_price #[:,0:1]
# rnn_size = 300
# X_train = list()
# y_train = list()
# for i in range(rnn_size, spliting_point):
#     X_train.append(gold_price_scaled[i-rnn_size:i, 0])
#     y_train.append(gold_price_scaled[i, 0])
# X_train, y_train = np.array(X_train), np.array(y_train)

# Blog method
# train_scaled = pd.DataFrame(train_scaled)
# train_scaled = ts(train_scaled,rnn_depth,pred_col=-1)
# train_scaled = train_scaled.values

# series_to_supervised
# series_to_supervised(scaled, n_hours, 1)
train_scaled = series_to_supervised(train_scaled,n_in=rnn_time_steps,n_out=1)
train_scaled = train_scaled.values

test_scaled = series_to_supervised(test_scaled,n_in=rnn_time_steps,n_out=1)
test_scaled = test_scaled.values

# Keras
# from keras.preprocessing import TimeseriesGenerator
# seq = TimeseriesGenerator(data, targets, length, sampling_rate=1, stride=1,
 start_index=0, end_index=None, shuffle=False, reverse=False, batch_size=128)
train_scaled.shape
```

Out[32]:
```
(7352, 196)
```

In [33]:
```
# Split into Input and output, X & Y
# One feature method
# x_train, y_train = train_scaled[:,1:], train_scaled[:,0:1]
# x_test, y_test = test_scaled[:,1:], test_scaled[:,0:1]

# split into input and outputs
n_obs = rnn_time_steps * n_features
x_train, y_train = train_scaled[:, :n_obs], train_scaled[:, -n_features]
x_test, y_test = test_scaled[:, :n_obs], test_scaled[:, -n_features]

print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
```

```
(7352, 192) (7352,) (165, 192) (165,)
```

In [34]:
```
# # Reshaping (batch_size, timesteps, input_dim) # reshape input to be 3D [sam
ples, timesteps, features]
# train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))

X_train = x_train.reshape(x_train.shape[0], rnn_time_steps, n_features)
X_test = x_test.reshape(x_test.shape[0], rnn_time_steps, n_features)
print(X_train.shape,X_test.shape)
```

```
(7352, 48, 4) (165, 48, 4)
```

LSTM Model

In [15]:
```
#  RNN Model libs
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

# Init RNN
regressor = Sequential()

# 1st LSTM Layer + dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train
.shape[1], X_train.shape[2])))
regressor.add(Dropout(0.2))

# 2nd LSTM Layer + dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# # 3rd LSTM Layer + dropout regularisation
# regressor.add(LSTM(units = 20, return_sequences = True))
# regressor.add(Dropout(0.2))

# 4th LSTM Layer+ dropout regularisation
regressor.add(LSTM(units = 20))
regressor.add(Dropout(0.2))

# Output layer
regressor.add(Dense(units = 1))

# Compile the model
regressor.compile(  optimizer = 'adam', loss = 'mean_squared_error') #, metric
s = ['accuracy'])

# Fitting the RNN to the training set
regressor.fit(X_train,y_train, epochs = 30, batch_size=48, validation_data=(X_
test, y_test), verbose=2, shuffle=False)
regressor.save('reg_lstm_v1_3.h5')
```

```
Train on 7352 samples, validate on 0 samples
Epoch 1/30
 - 33s - loss: 0.0116
Epoch 2/30
 - 32s - loss: 0.0261
Epoch 3/30
 - 33s - loss: 0.0170
Epoch 4/30
 - 35s - loss: 0.0081
Epoch 5/30
 - 35s - loss: 0.0043
Epoch 6/30
 - 35s - loss: 0.0046
Epoch 7/30
 - 34s - loss: 0.0049
Epoch 8/30
 - 39s - loss: 0.0026
Epoch 9/30
 - 40s - loss: 0.0026
Epoch 10/30
 - 39s - loss: 0.0020
Epoch 11/30
 - 40s - loss: 0.0021
Epoch 12/30
 - 41s - loss: 0.0023
Epoch 13/30
 - 41s - loss: 0.0019
Epoch 14/30
 - 40s - loss: 0.0020
Epoch 15/30
 - 39s - loss: 0.0023
Epoch 16/30
 - 41s - loss: 0.0016
Epoch 17/30
 - 39s - loss: 0.0017
Epoch 18/30
 - 40s - loss: 0.0018
Epoch 19/30
 - 39s - loss: 0.0016
Epoch 20/30
 - 43s - loss: 0.0017
Epoch 21/30
 - 40s - loss: 0.0016
Epoch 22/30
 - 43s - loss: 0.0013
Epoch 23/30
 - 42s - loss: 0.0013
Epoch 24/30
 - 44s - loss: 0.0015
Epoch 25/30
 - 40s - loss: 0.0012
Epoch 26/30
 - 41s - loss: 0.0013
Epoch 27/30
 - 46s - loss: 0.0011
Epoch 28/30
 - 41s - loss: 0.0011
```

```
Epoch 29/30
 - 40s - loss: 0.0019
Epoch 30/30
 - 41s - loss: 0.0013
```

In [35]:
```
# Restore a Model to aviod test set conflicts
# regressor.save('reg_lstm4.h5')
regressor = load_model('reg_lstm_v1_3.h5')
```

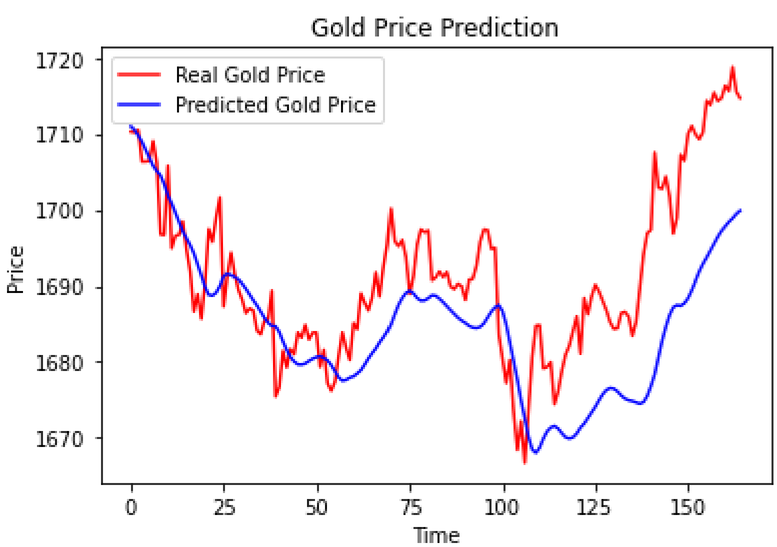
Test the Model

In [36]:
```
# Test the RNN
# dataset_test = test_price #['closePrice']
# real_gold_price = dataset_test.iloc[:,0:1].values

# # dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), a
xis = 0)
# dataset_total = features_price #['closePrice']
# inputs = dataset_total[len(dataset_total) - len(dataset_test) - 200:].values
# inputs = inputs.reshape(-1,1)
# gold_test_scaled = sc.transform(inputs[:,0:1])
# # gold_test_scaled =

# # RNN Data Structure
# X_test = list()
# for i in range(200, 277):
#     X_test.append(gold_test_scaled[i-200:i, 0])
# X_test = np.array(X_test)

# # Reshaping
# X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Prediction
y_pred = regressor.predict(X_test)
# Hint: test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))
X_test = X_test.reshape((X_test.shape[0], rnn_time_steps*n_features))
# invert scaling for forecast
inv_yhat = concatenate((y_pred, X_test[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
y_test = y_test.reshape((len(y_test), 1))
inv_y = concatenate((y_test, X_test[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]


# predicted_gold_price = invert_scale(scaler, X_test, y_pred)
# predicted_gold_price = scaler.inverse_transform(predicted_gold_price)
# predicted_gold_price = predicted_gold_price[:,0]
# predicted_gold_price = predicted_gold_price.reshape(-1,1)
```

In [37]:
```
# calculate RMSE
from sklearn.metrics import mean_squared_error
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))

print(inv_y.shape, inv_yhat.shape)
print('Test RMSE: %.3f' % rmse)
```

```
(165,) (165,)
Test RMSE: 10.370
```

In [39]:
```
# Visualization the results
plot_size = None # Max 165
plt.plot(inv_y[plot_size:], color= 'red', label = 'Real Gold Price')
plt.plot(inv_yhat[plot_size:], color= 'blue', label = 'Predicted Gold Price')
plt.title('Gold Price Prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
# plt.savefig('gold_multi_feature_test.png' , dpi=150)
plt.show()
```



In [ ]:
```
y_pred
```

In [40]:
```
X_test.shape, y_pred.shape
```

Out[40]:
```
((165, 192), (165, 1))
```