In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [2]:

```
df = pd.read_csv("gold_price_data.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	Date	Value
0	1970-01-01	35.2
1	1970-04-01	35.1
2	1970-07-01	35.4
3	1970-10-01	36.2
4	1971-01-01	37.4

In [4]:

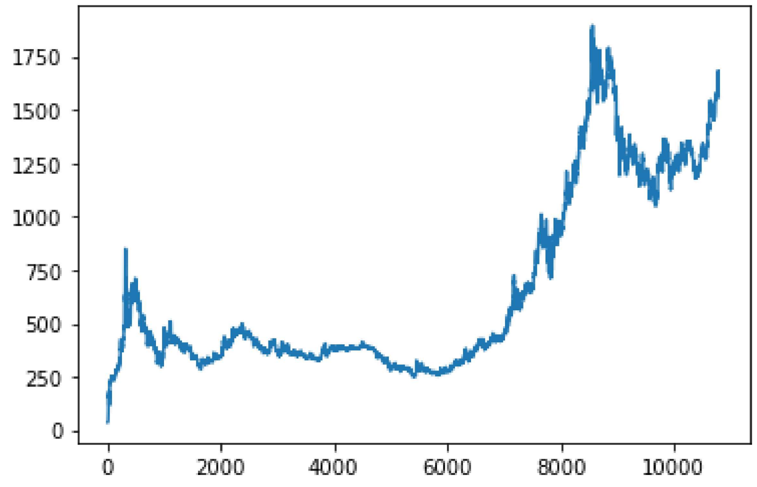
```
df.tail()
```

Out[4]:

	Date	Value
10782	2020-03-09	1672.50
10783	2020-03-10	1655.70
10784	2020-03-11	1653.75
10785	2020-03-12	1570.70
10786	2020-03-13	1562.80

In [5]:

```
plt.figure(figsize=(6,4))
plt.plot(df["Value"])
plt.show()
```

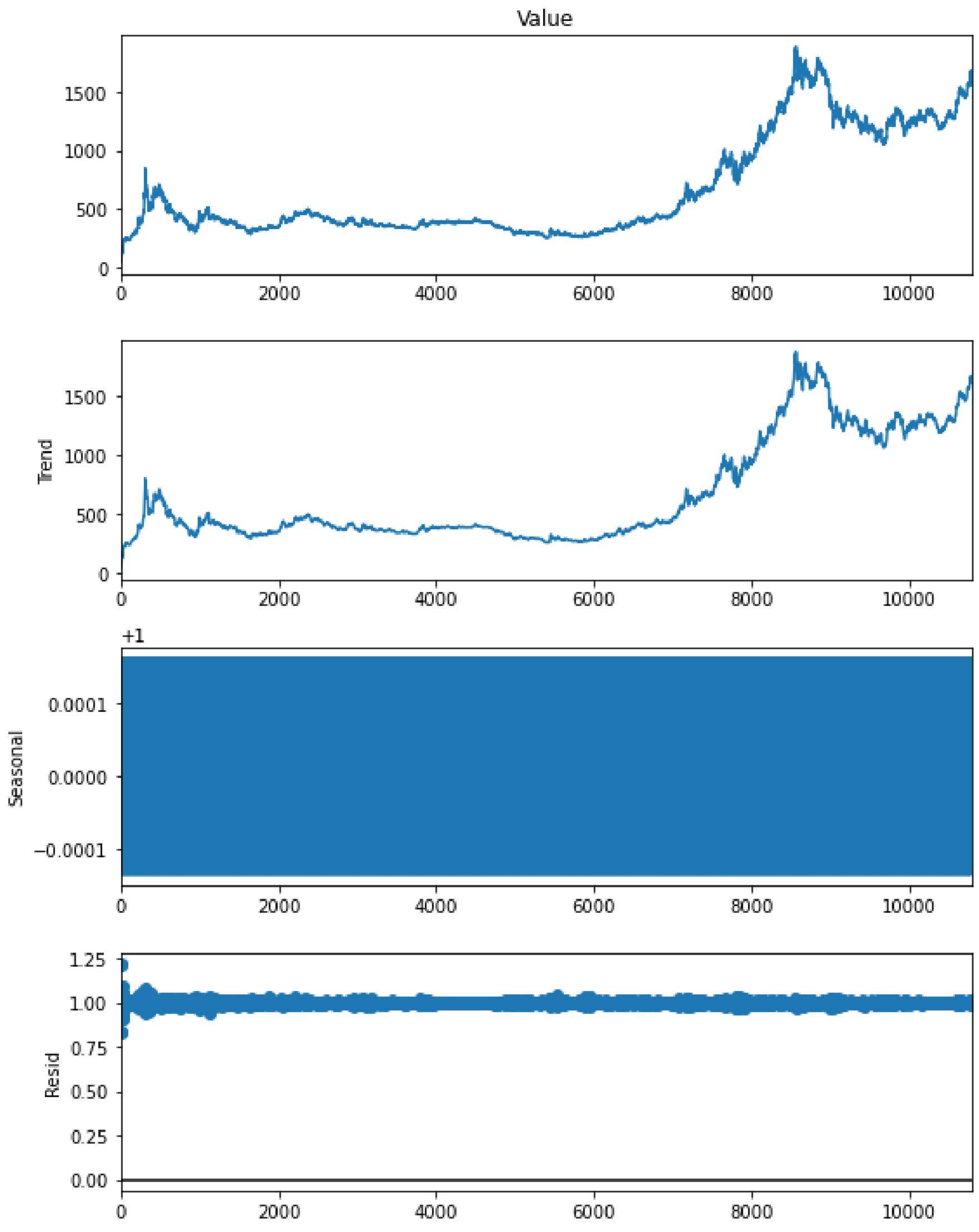


In [6]:

```
result = seasonal_decompose(x=df["Value"], model ="multiplicative", period=4)
```

In [7]:

```
plt.rcParams.update({'figure.figsize': (8,10)})
result.plot()
plt.show()
```



There is Trend and Possible Sesonality we want to remove seasonality

In [8]:

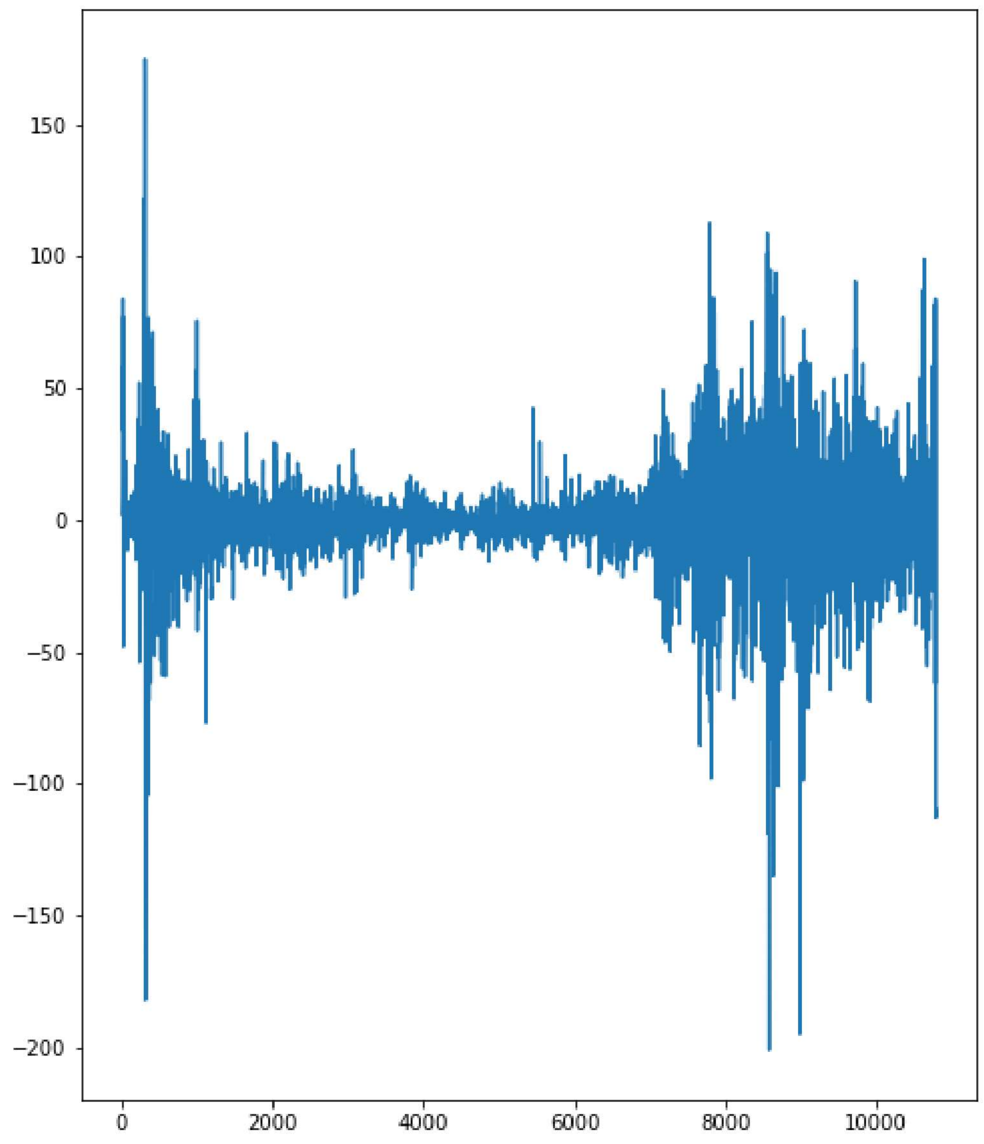
```
difference = df["Value"] - df["Value"].shift(4)
```

In [9]:

```
plt.plot(difference.dropna())
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x1f2fa41b040>]
```



In [10]:

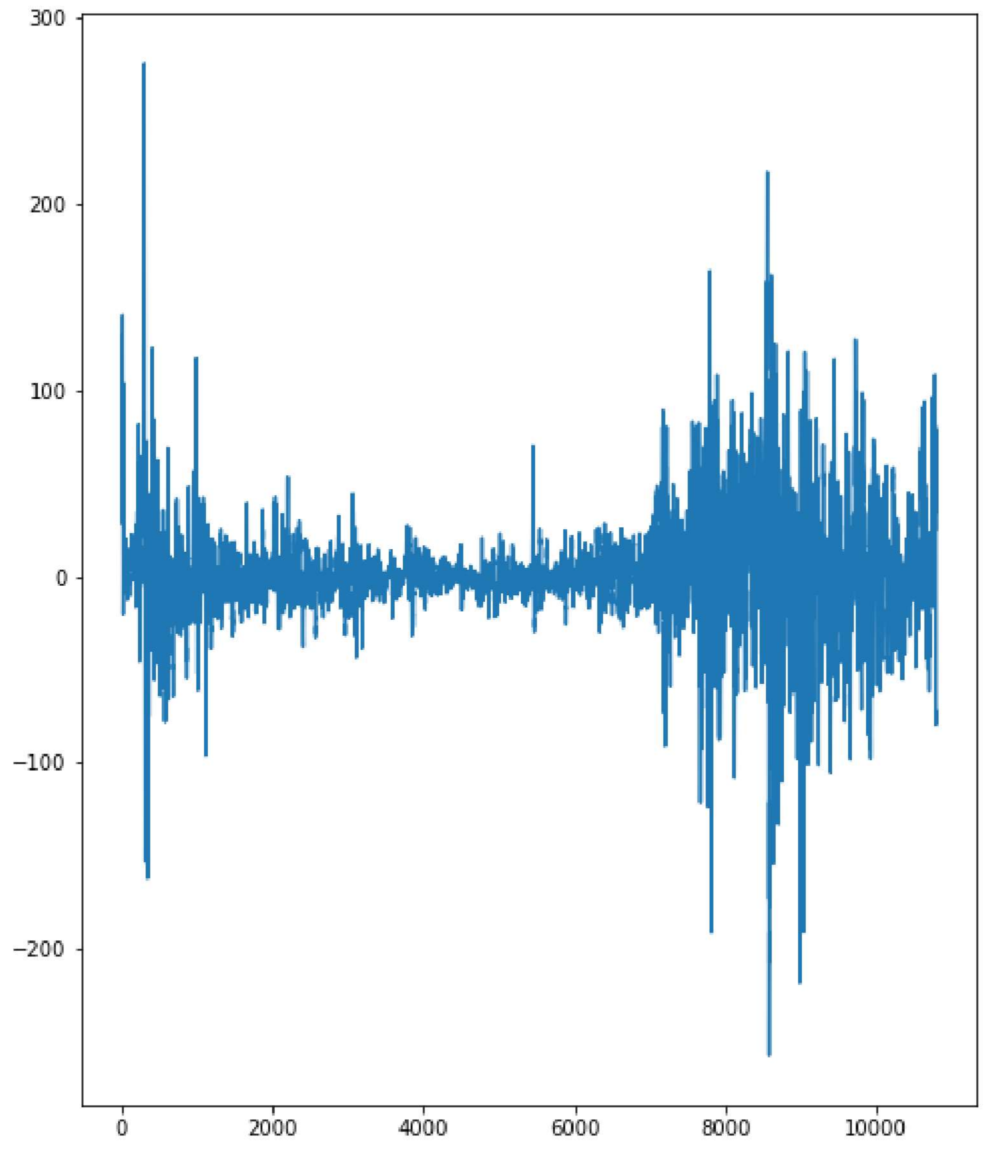
```
seasonal_difference = df["Value"] - df["Value"].shift(12)
```

In [11]:

```
plt.plot(seasonal_difference.dropna())
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x1f2fa4865b0>]
```



In [12]:

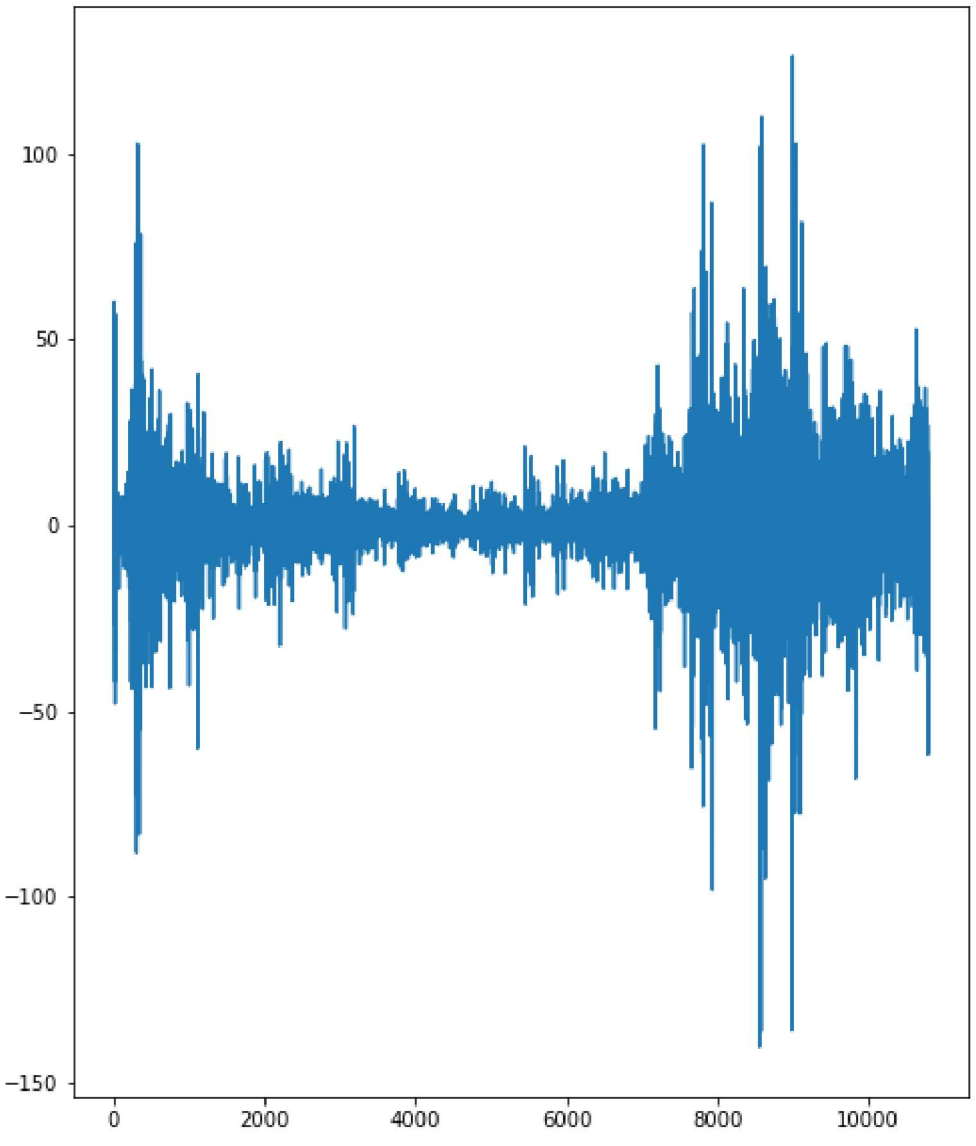
```
double_difference = seasonal_difference - seasonal_difference.shift(1)
```

In [13]:

```
plt.plot(double_difference.dropna())
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x1f2fa4f2310>]
```

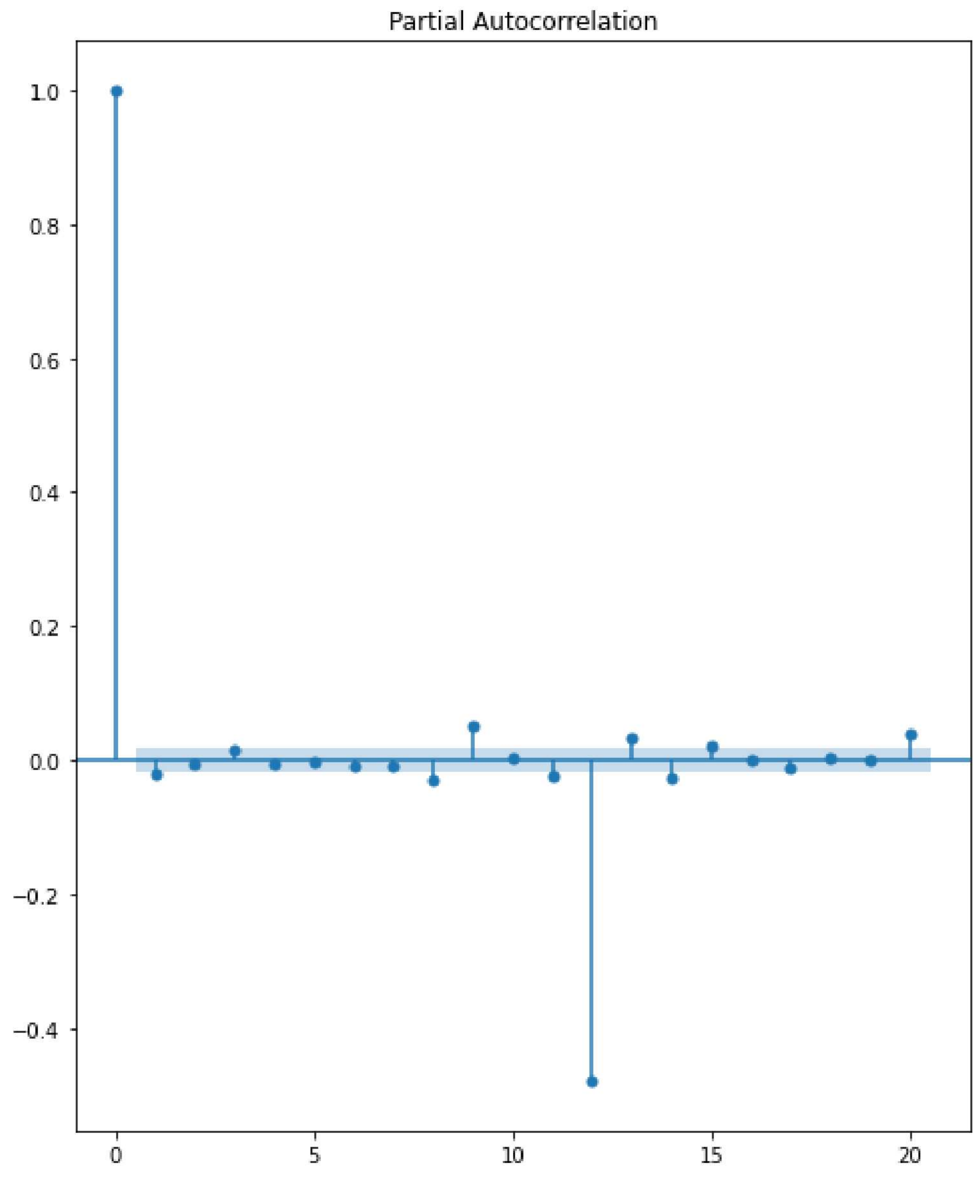


In [14]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

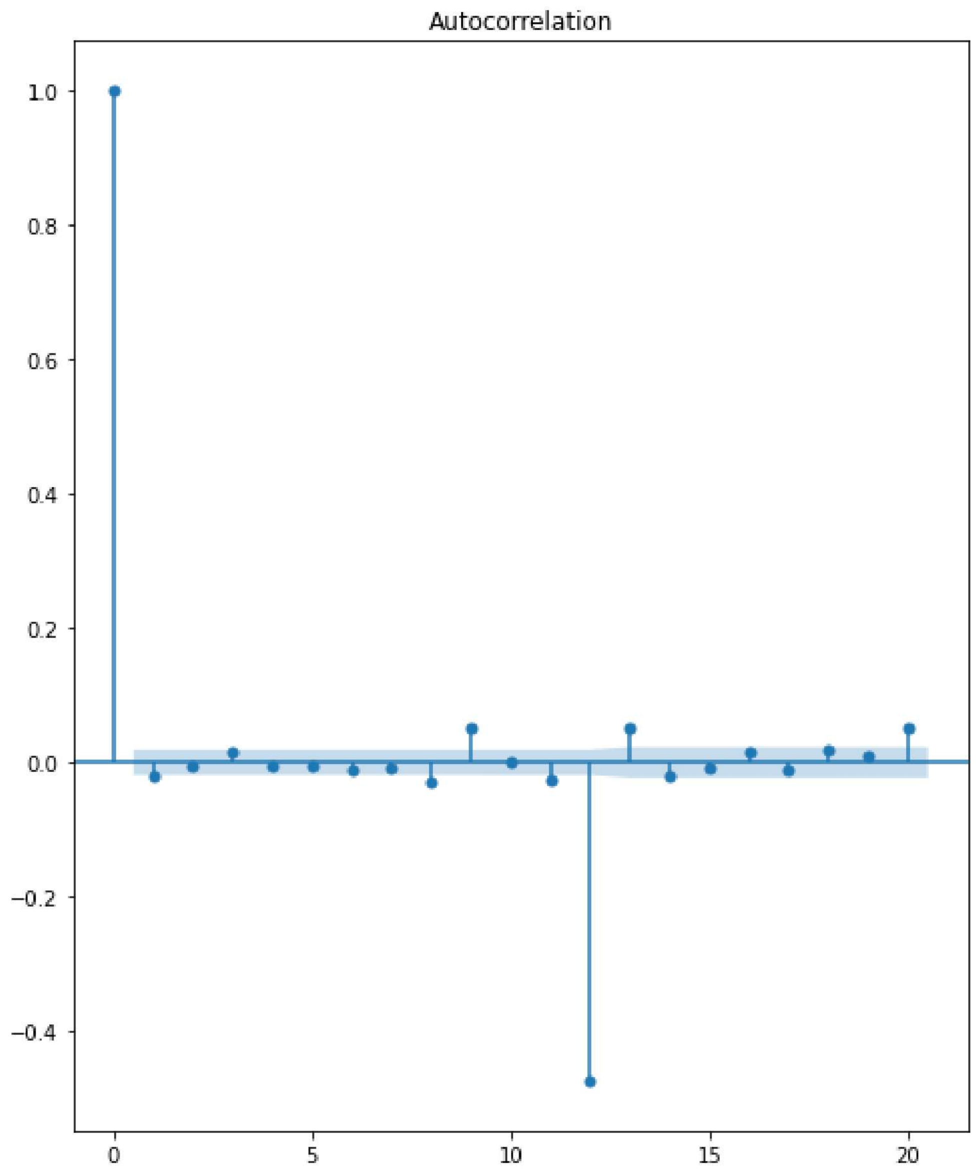
In [15]:

```
plot_pacf(double_difference.dropna(), lags=20)
plt.show()
```



In [16]:

```
plot_acf(double_difference.dropna(), lags=20)
plt.show()
```



In [17]:

```
df["Value"].shape
```

Out[17]:

```
(10787,)
```

Implementation of ARIMA

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration

In [18]:

```
from statsmodels.tsa.arima_model import ARIMA
import warnings
warnings.filterwarnings("ignore")
```

In [19]:

```
model=ARIMA(df["Value"],order=(1,2,1))
model=model.fit()
```

In [20]:

```
df["forecast"] = model.predict(start=7550, end=10786)
```

In [21]:

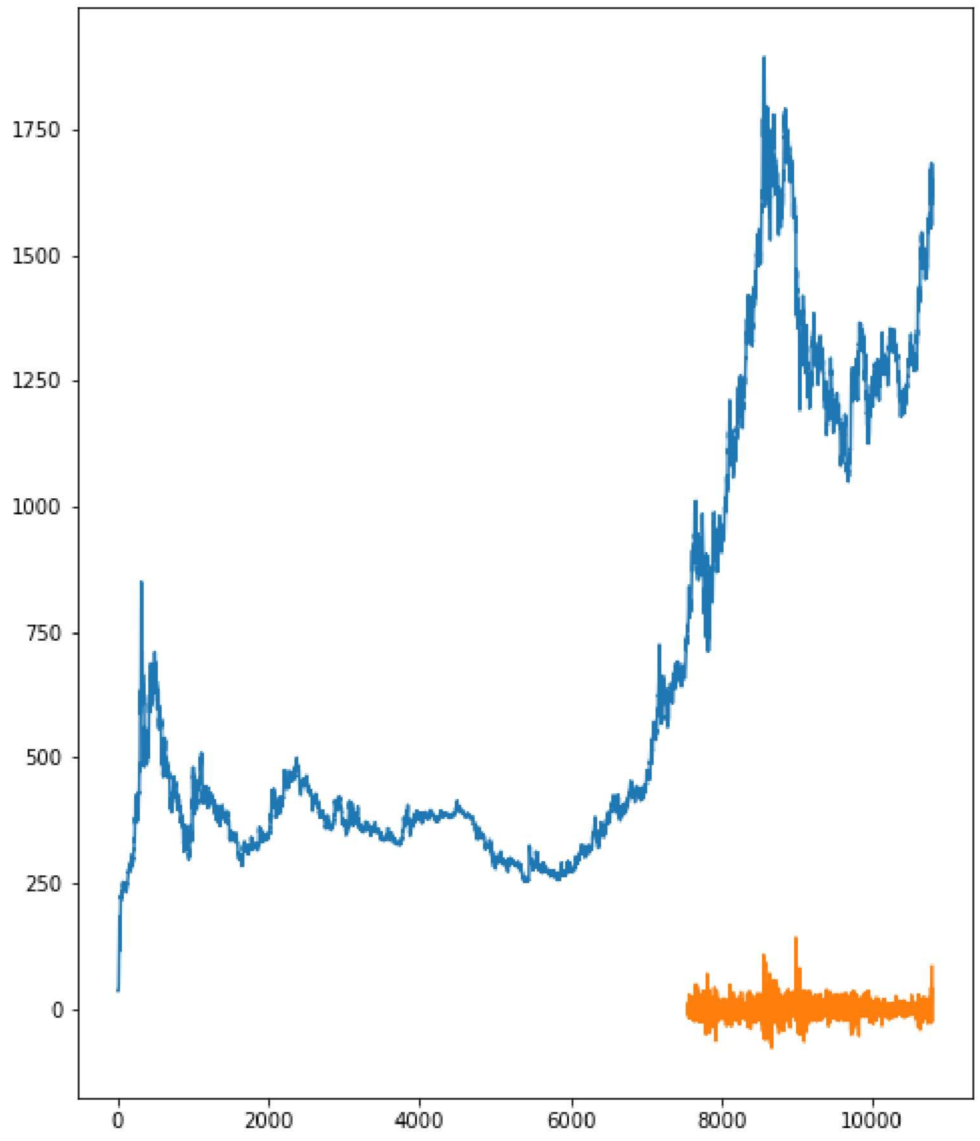
```
df.tail()
```

Out[21]:

	Date	Value	forecast
10782	2020-03-09	1672.50	-24.109570
10783	2020-03-10	1655.70	11.522923
10784	2020-03-11	1653.75	17.240928
10785	2020-03-12	1570.70	2.207836
10786	2020-03-13	1562.80	84.298915

In [22]:

```
plt.figure()
plt.plot(df[["Value","forecast"]])
plt.show()
```



here you can see prediction is not good so we try to make good prediction using SARIMAX model

SARIMAX

In [23]:

```
import statsmodels.api as sm
```

In [24]:

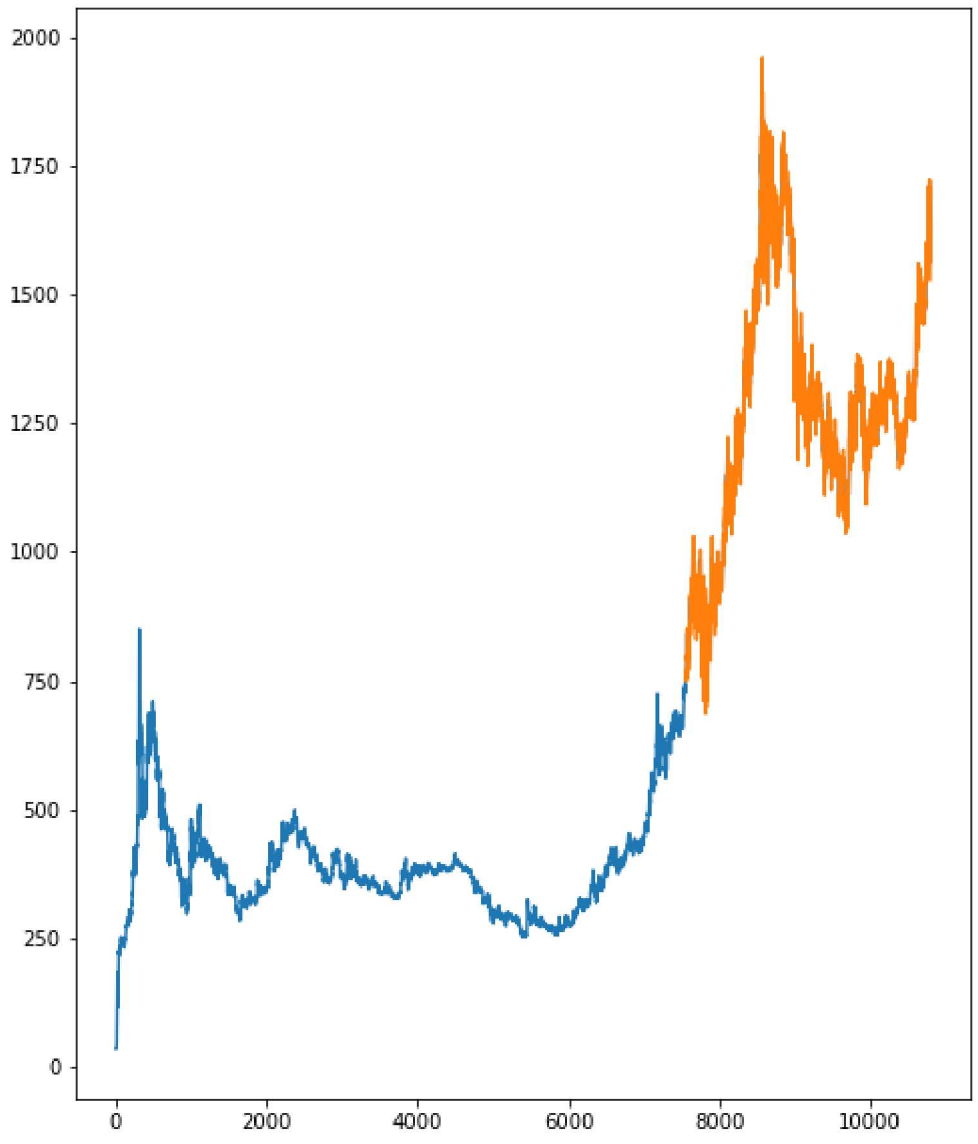
```
model = sm.tsa.statespace.SARIMAX(df['Value'],order=(1, 2, 1),seasonal_order=(1,2,1,4), tre
model = model.fit()
```

In [25]:

```
df['forecast2']=model.predict(start=7550,end=10786)
```

In [26]:

```
plt.figure()
plt.plot(df[["Value","forecast2"]])
plt.show()
```



#simple ANN

In [27]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,SimpleRNN, LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error

import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np
```

In [28]:

```
df = pd.read_csv("gold_price_data.csv")
```

In [29]:

```
df.shape
```

Out[29]:

```
(10787, 2)
```

In [30]:

```
train = df["Value"][:7550]
test = df["Value"][7550:]
```

In [31]:

```
train.shape
```

Out[31]:

```
(7550,)
```

In [32]:

```
test.shape
```

Out[32]:

```
(3237,)
```

In [33]:

```
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(np.array(train).reshape(-1,1))
test_scaled = scaler.transform(np.array(test).reshape(-1,1))
```

In [34]:

```
#convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
    X, y = [], []
    for i in range(len(dataset)-time_step-1):
        sample = dataset[i:(i+time_step), 0]
        X.append(sample)
        y.append(dataset[i + time_step, 0])
    return np.array(X), np.array(y)
```

In [35]:

```
time_step = 50
X_train, y_train = create_dataset(train_scaled, time_step)
X_test, y_test = create_dataset(test_scaled, time_step)
```

In [36]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(7499, 50)
(3186, 50)
```

In [37]:

```
# reshape into (samples, time steps, features)
X_train = X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test  = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
```

In [38]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(7499, 50, 1)
(3186, 50, 1)
```

In [39]:

```
model = Sequential()
model.add(SimpleRNN(32, return_sequences=True, input_shape=(time_step,1)))
model.add(SimpleRNN(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [40]:

```
model.fit(X_train,y_train, epochs=20, batch_size=32)
```

```
Epoch 1/20
235/235 [==============================] - 5s 15ms/step - loss: 0.0040
Epoch 2/20
235/235 [==============================] - 3s 13ms/step - loss: 1.7341e-04
Epoch 3/20
235/235 [==============================] - 3s 13ms/step - loss: 1.3838e-04
Epoch 4/20
235/235 [==============================] - 3s 14ms/step - loss: 1.2086e-04
Epoch 5/20
235/235 [==============================] - 3s 12ms/step - loss: 1.0499e-04
Epoch 6/20
235/235 [==============================] - 3s 14ms/step - loss: 1.0275e-04
Epoch 7/20
235/235 [==============================] - 3s 13ms/step - loss: 9.5803e-05
Epoch 8/20
235/235 [==============================] - 3s 13ms/step - loss: 9.9652e-05
Epoch 9/20
235/235 [==============================] - 3s 13ms/step - loss: 8.8190e-05
Epoch 10/20
235/235 [==============================] - 3s 14ms/step - loss: 9.6903e-05
Epoch 11/20
235/235 [==============================] - 3s 14ms/step - loss: 8.7371e-05
Epoch 12/20
235/235 [==============================] - 3s 12ms/step - loss: 7.9699e-05
Epoch 13/20
235/235 [==============================] - 3s 11ms/step - loss: 1.0784e-04
Epoch 14/20
235/235 [==============================] - 3s 12ms/step - loss: 7.6306e-05
Epoch 15/20
235/235 [==============================] - 3s 11ms/step - loss: 7.6166e-05
Epoch 16/20
235/235 [==============================] - 3s 12ms/step - loss: 7.2140e-05
Epoch 17/20
235/235 [==============================] - 3s 12ms/step - loss: 8.5792e-05
Epoch 18/20
235/235 [==============================] - 3s 11ms/step - loss: 8.3440e-05
Epoch 19/20
235/235 [==============================] - 3s 11ms/step - loss: 6.9444e-05
Epoch 20/20
235/235 [==============================] - 3s 12ms/step - loss: 9.3106e-05
```

Out[40]:

```
<keras.callbacks.History at 0x1f28578d160>
```

In [41]:

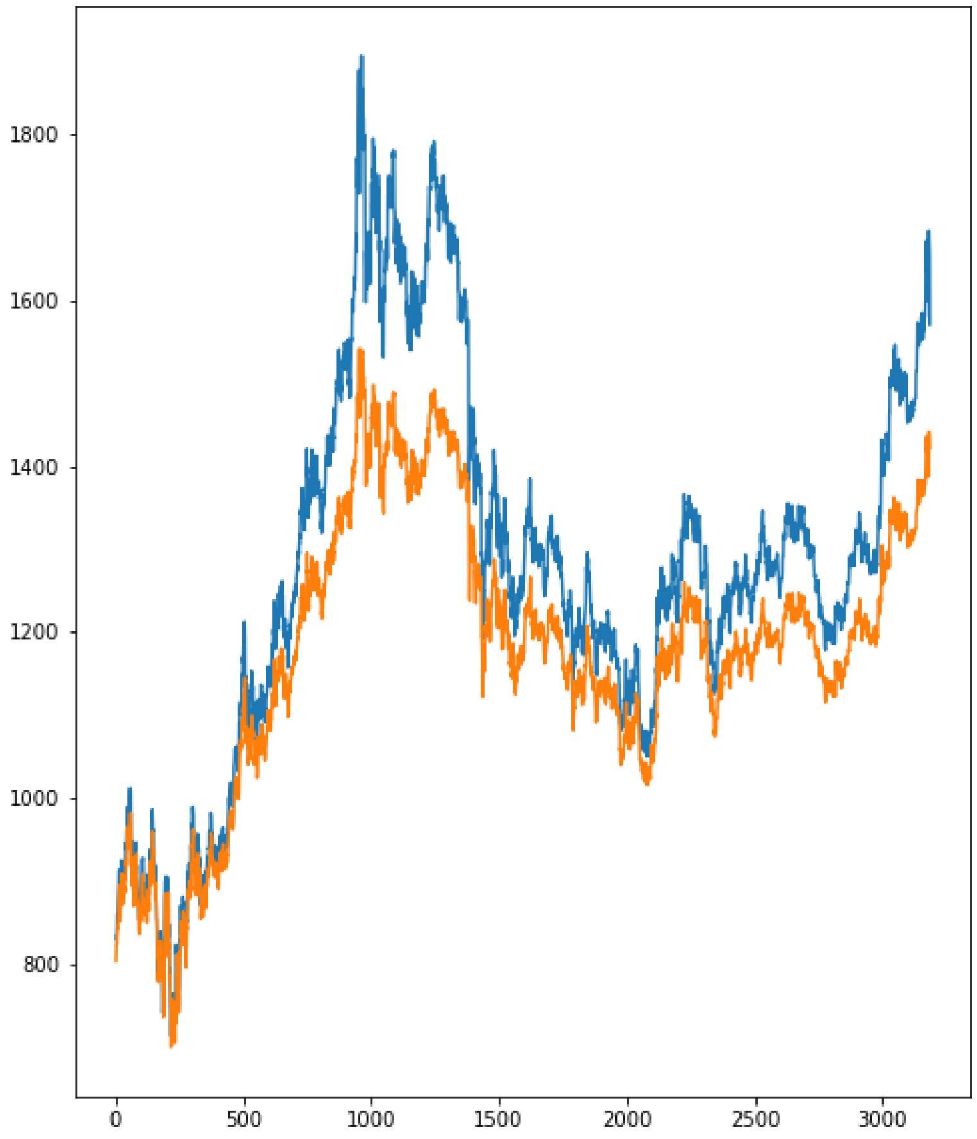
```
y_pred = model.predict(X_test)
mean_squared_error(y_test,y_pred)
```

Out[41]:

```
0.023601914835312054
```

In [42]:

```
plt.figure()
plt.plot(scaler.inverse_transform(np.array(y_test).reshape(-1,1)))
plt.plot(scaler.inverse_transform(y_pred))
plt.show()
```



Lstm

In [43]:

```
model = Sequential()
model.add(LSTM(32, return_sequences=True, input_shape=(time_step,1)))
model.add(LSTM(32, return_sequences=True))
model.add(LSTM(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [44]:

```
model.fit(X_train,y_train, epochs=50, batch_size=32)
```

```
Epoch 1/50
235/235 [==============================] - 14s 39ms/step - loss: 0.0056
Epoch 2/50
235/235 [==============================] - 9s 39ms/step - loss: 4.3055e-04
Epoch 3/50
235/235 [==============================] - 12s 49ms/step - loss: 3.8653e-04
Epoch 4/50
235/235 [==============================] - 11s 45ms/step - loss: 3.0674e-04
Epoch 5/50
235/235 [==============================] - 9s 37ms/step - loss: 2.7207e-04
Epoch 6/50
235/235 [==============================] - 10s 43ms/step - loss: 2.4524e-04
Epoch 7/50
235/235 [==============================] - 10s 41ms/step - loss: 2.1201e-04
Epoch 8/50
235/235 [==============================] - 11s 49ms/step - loss: 1.9286e-04
Epoch 9/50
235/235 [==============================] - 13s 54ms/step - loss: 1.6556e-04
Epoch 10/50
235/235 [==============================] - 10s 42ms/step - loss: 1.5978e-04
Epoch 11/50
235/235 [==============================] - 10s 42ms/step - loss: 1.1866e-04
Epoch 12/50
235/235 [==============================] - 9s 40ms/step - loss: 1.1172e-04
Epoch 13/50
235/235 [==============================] - 10s 41ms/step - loss: 9.8706e-05
Epoch 14/50
235/235 [==============================] - 10s 41ms/step - loss: 9.7084e-05
Epoch 15/50
235/235 [==============================] - 10s 41ms/step - loss: 9.4640e-05
Epoch 16/50
235/235 [==============================] - 10s 41ms/step - loss: 9.1448e-05
Epoch 17/50
235/235 [==============================] - 9s 40ms/step - loss: 9.8959e-05
Epoch 18/50
235/235 [==============================] - 9s 40ms/step - loss: 9.3067e-05
Epoch 19/50
235/235 [==============================] - 9s 39ms/step - loss: 8.8636e-05
Epoch 20/50
235/235 [==============================] - 10s 41ms/step - loss: 8.6646e-05
Epoch 21/50
235/235 [==============================] - 10s 41ms/step - loss: 7.2776e-05
Epoch 22/50
235/235 [==============================] - 12s 50ms/step - loss: 7.8231e-05
Epoch 23/50
235/235 [==============================] - 10s 41ms/step - loss: 7.8281e-05
Epoch 24/50
235/235 [==============================] - 10s 42ms/step - loss: 8.8569e-05
Epoch 25/50
235/235 [==============================] - 9s 38ms/step - loss: 7.2223e-05
Epoch 26/50
235/235 [==============================] - 10s 41ms/step - loss: 7.0950e-05
Epoch 27/50
235/235 [==============================] - 9s 39ms/step - loss: 6.7636e-05
Epoch 28/50
235/235 [==============================] - 9s 38ms/step - loss: 6.5756e-05
```

```
Epoch 29/50
235/235 [==============================] - 9s 38ms/step - loss: 8.2342e-05
Epoch 30/50
235/235 [==============================] - 9s 39ms/step - loss: 7.2135e-05
Epoch 31/50
235/235 [==============================] - 9s 39ms/step - loss: 6.6160e-05
Epoch 32/50
235/235 [==============================] - 9s 39ms/step - loss: 8.0016e-05
Epoch 33/50
235/235 [==============================] - 9s 39ms/step - loss: 7.4974e-05
Epoch 34/50
235/235 [==============================] - 9s 38ms/step - loss: 6.5439e-05
Epoch 35/50
235/235 [==============================] - 9s 40ms/step - loss: 6.4746e-05
Epoch 36/50
235/235 [==============================] - 9s 38ms/step - loss: 6.7422e-05
Epoch 37/50
235/235 [==============================] - 9s 39ms/step - loss: 6.7494e-05
Epoch 38/50
235/235 [==============================] - 9s 39ms/step - loss: 6.5822e-05
Epoch 39/50
235/235 [==============================] - 9s 39ms/step - loss: 7.0463e-05
Epoch 40/50
235/235 [==============================] - 9s 40ms/step - loss: 6.6668e-05
Epoch 41/50
235/235 [==============================] - 9s 39ms/step - loss: 6.5559e-05
Epoch 42/50
235/235 [==============================] - 9s 39ms/step - loss: 6.6939e-05
Epoch 43/50
235/235 [==============================] - 9s 40ms/step - loss: 6.5834e-05
Epoch 44/50
235/235 [==============================] - 9s 37ms/step - loss: 6.0422e-05
Epoch 45/50
235/235 [==============================] - 9s 37ms/step - loss: 7.3393e-05
Epoch 46/50
235/235 [==============================] - 9s 37ms/step - loss: 6.2809e-05
Epoch 47/50
235/235 [==============================] - 9s 37ms/step - loss: 6.3733e-05
Epoch 48/50
235/235 [==============================] - 10s 42ms/step - loss: 6.6080e-05
Epoch 49/50
235/235 [==============================] - 9s 38ms/step - loss: 6.6318e-05
Epoch 50/50
235/235 [==============================] - 9s 37ms/step - loss: 6.3409e-05
```

Out[44]:

```
<keras.callbacks.History at 0x1f28b804250>
```

In [45]:

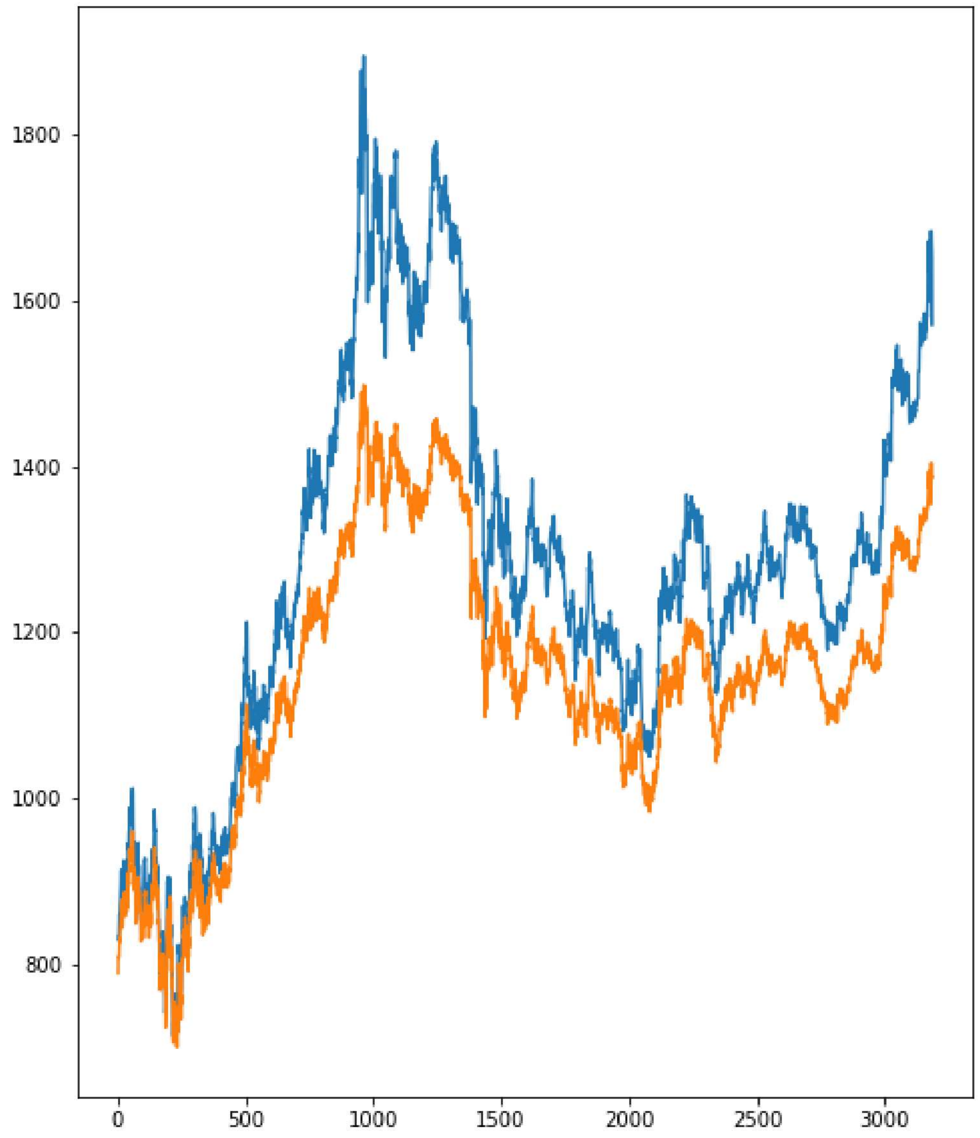
```
y_pred = model.predict(X_test)
mean_squared_error(y_test,y_pred)
```

Out[45]:

```
0.034674674916339215
```

In [46]:

```
plt.figure()
plt.plot(scaler.inverse_transform(np.array(y_test).reshape(-1,1)))
plt.plot(scaler.inverse_transform(y_pred))
plt.show()
```



In [47]:

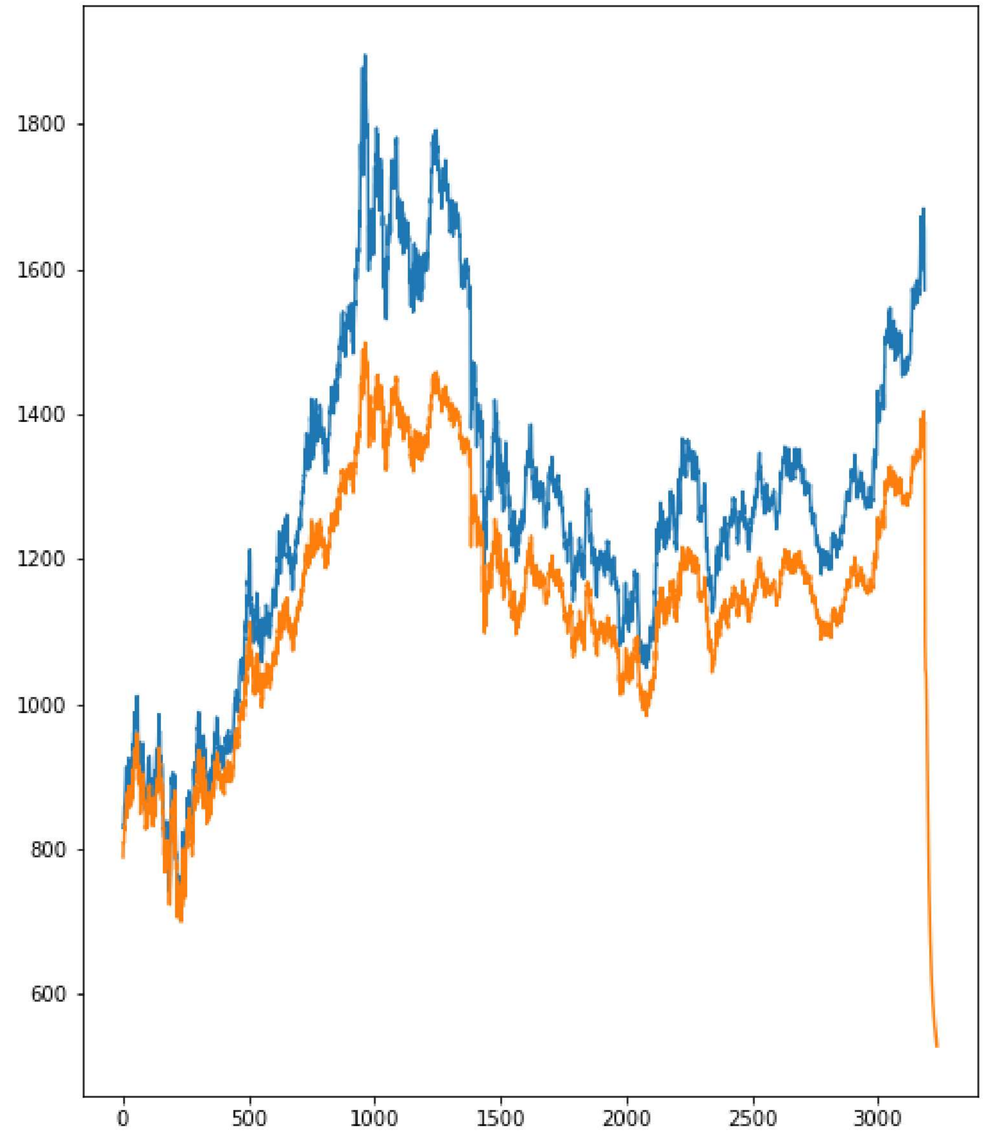
```
# next 50 days

days = 50
last_input = X_test[-1]
last_output = y_pred[-1]
y_forecast = []

for i in range(1,days+1):
    last_input = np.append(last_input[1:], last_output)
    last_output = model.predict(last_input.reshape(1,50,1))
    y_forecast.append(last_output[0][0])
```

In [48]:

```
all_predictions = np.append(y_pred,y_forecast)
```

In [49]:

```
plt.figure()
plt.plot(scaler.inverse_transform(np.array(y_test).reshape(-1,1)))
plt.plot(scaler.inverse_transform(all_predictions.reshape(-1,1)))
plt.show()
```



In [ ]:

```

```