Gold Price Prediction

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/dock
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list a

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
# You can also write temporary files to /kaggle/temp/, but they won't be saved ou
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings(action='ignore')
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
```

In [3]:
```
gold_data=pd.read_csv('gld_price_data.csv')
```

In [6]:
```
gold_data = pd.read_csv
```

In [15]:
```
gold_data.head(20)
```

Out[15]:

	Date	SPX	GLD	USO	SLV	EUR/USD
0	1/2/2008	1447.160034	84.860001	78.470001	15.180000	1.471692
1	1/3/2008	1447.160034	85.570000	78.370003	15.285000	1.474491
2	1/4/2008	1411.630005	85.129997	77.309998	15.167000	1.475492
3	1/7/2008	1416.180054	84.769997	75.500000	15.053000	1.468299
4	1/8/2008	1390.189941	86.779999	76.059998	15.590000	1.557099
5	1/9/2008	1409.130005	86.550003	75.250000	15.520000	1.466405
6	1/10/2008	1420.329956	88.250000	74.019997	16.061001	1.480100
7	1/11/2008	1401.020020	88.580002	73.089996	16.077000	1.479006
8	1/14/2008	1416.250000	89.540001	74.250000	16.280001	1.486900
9	1/15/2008	1380.949951	87.989998	72.779999	15.834000	1.480210
10	1/16/2008	1373.199951	86.699997	71.849998	15.654000	1.466405
11	1/17/2008	1333.250000	86.500000	71.029999	15.717000	1.464000
12	1/18/2008	1325.189941	87.419998	71.540001	16.030001	1.461796
13	1/22/2008	1310.500000	88.169998	70.550003	15.902000	1.464794
14	1/23/2008	1338.599976	87.889999	69.500000	15.900000	1.463208
15	1/24/2008	1352.069946	90.080002	70.930000	16.299999	1.477410
16	1/25/2008	1330.609985	90.300003	71.910004	16.298000	1.467502
17	1/28/2008	1353.959961	91.750000	72.349998	16.549999	1.478809
18	1/29/2008	1362.300049	91.150002	72.980003	16.534000	1.477192
19	1/30/2008	1355.810059	92.059998	73.080002	16.674999	1.483107

In [16]:
```
gold_data.shape
```

Out[16]:
```
(2290, 6)
```

In [17]:
```
gold_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Date     2290 non-null   object 
 1   SPX      2290 non-null   float64
 2   GLD      2290 non-null   float64
 3   USO      2290 non-null   float64
 4   SLV      2290 non-null   float64
 5   EUR/USD  2290 non-null   float64
dtypes: float64(5), object(1)
memory usage: 107.5+ KB
```

In [19]:
```
gold_data['Date'] = pd.to_datetime(gold_data['Date'])
gold_data['Date'] = gold_data['Date'].apply(lambda x:x.date())
```

In [20]:
```
gold_data.describe(include='all')
```

Out[20]:

	Date	SPX	GLD	USO	SLV	EUR/USD
count	2290	2290.000000	2290.000000	2290.000000	2290.000000	2290.000000
unique	2290	NaN	NaN	NaN	NaN	NaN
top	2015-10-12	NaN	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN	NaN
mean	NaN	1654.315776	122.732875	31.842221	20.084997	1.283653
std	NaN	519.111540	23.283346	19.523517	7.092566	0.131547
min	NaN	676.530029	70.000000	7.960000	8.850000	1.039047
25%	NaN	1239.874969	109.725000	14.380000	15.570000	1.171313
50%	NaN	1551.434998	120.580002	33.869999	17.268500	1.303297
75%	NaN	2073.010070	132.840004	37.827501	22.882500	1.369971
max	NaN	2872.870117	184.589996	117.480003	47.259998	1.598798

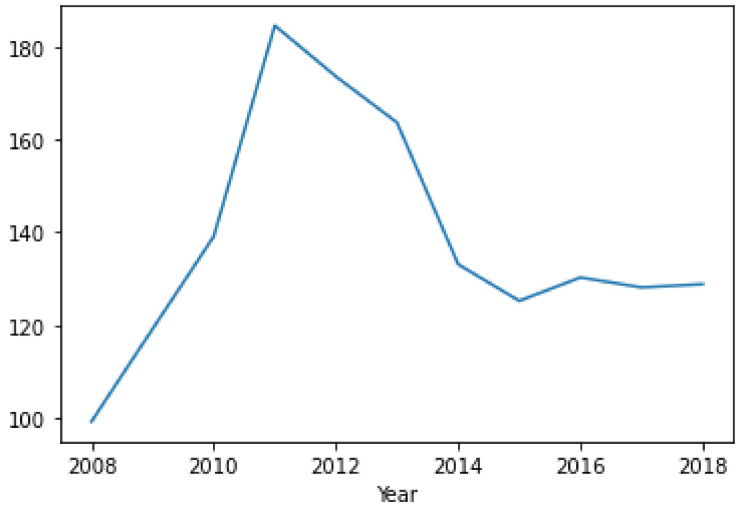
In [21]:
```
gold_data['Year'] = gold_data['Date'].apply(lambda x: x.year)
```

In [22]:
```
gold_data['Year'].value_counts()
```

Out[22]:
```
2014    224
2009    224
2015    223
2010    222
2011    222
2016    221
2013    221
2012    219
2017    218
2008    209
2018     87
Name: Year, dtype: int64
```

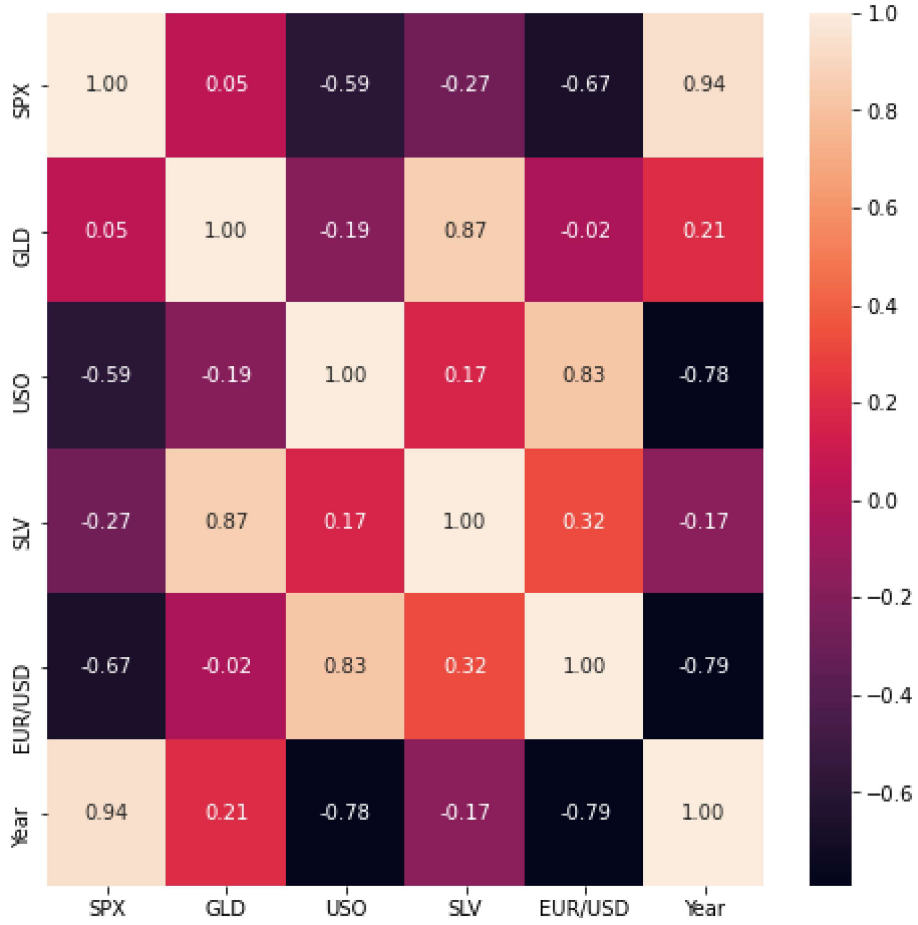
In [23]:
```
gold_data.groupby('Year').max()['GLD'].plot()
```

Out[23]:
```
<AxesSubplot:xlabel='Year'>
```



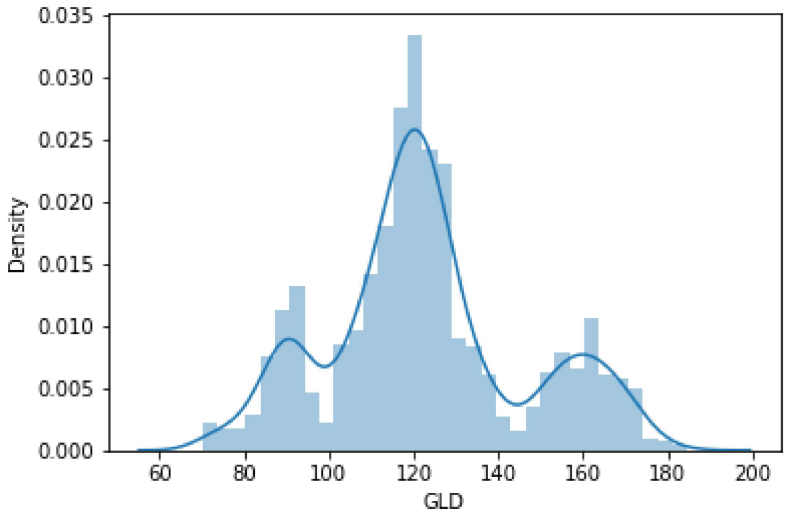
In [25]:
```
plt.figure(figsize=(8,8))
sns.heatmap(gold_data.corr(),annot=True,fmt='.2f')
```

Out[25]: <AxesSubplot:>



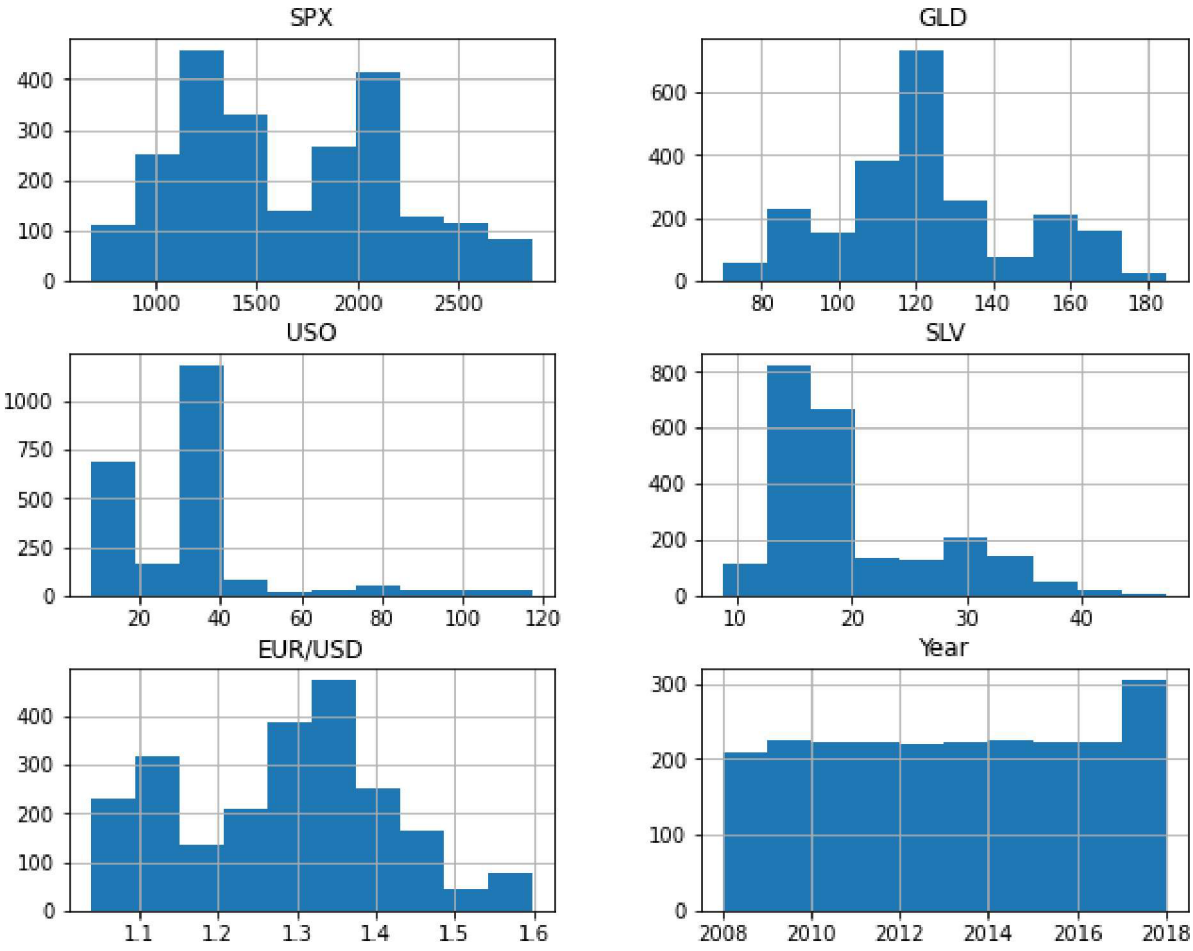
In [26]:
```
sns.distplot(gold_data['GLD'])
```

Out[26]:
```
<AxesSubplot:xlabel='GLD', ylabel='Density'>
```



In [27]:
```
gold_data.hist(figsize=(10,8))
```

Out[27]:
```
array([[<AxesSubplot:title={'center':'SPX'}>,
        <AxesSubplot:title={'center':'GLD'}>],
       [<AxesSubplot:title={'center':'USO'}>,
        <AxesSubplot:title={'center':'SLV'}>],
       [<AxesSubplot:title={'center':'EUR/USD'}>,
        <AxesSubplot:title={'center':'Year'}>]], dtype=object)
```



In [28]:
```
x = gold_data.drop(['Date','Year','GLD'],axis=1)
y = gold_data['GLD']
```

In [29]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_stat
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[29]:
```
((1832, 4), (1832,), (458, 4), (458,))
```

In [30]:
```
regressor = RandomForestRegressor(n_estimators=100)
```

In [31]:
```
regressor.fit(x_train,y_train)
```

Out[31]:
```
RandomForestRegressor()
```

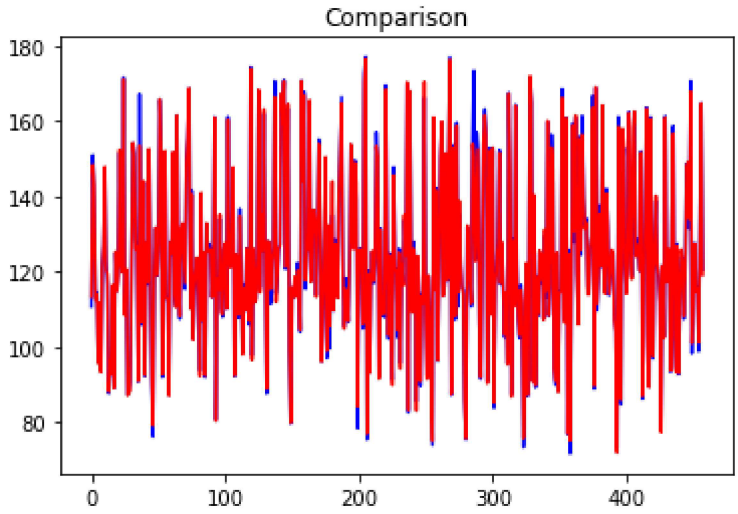
In [32]:
```
y_train_pred = regressor.predict(x_train)
y_test_pred = regressor.predict(x_test)
```

In [33]:
```
error_score = metrics.r2_score(y_test,y_test_pred)
print(error_score)
```

```
0.9874068933318638
```

In [34]:
```
#error_score2 = metrics.r2_score(y_train,y_train_pred)
#print(error_score2)
```

In [35]:
```
#Comparing actual and predicted values with a plot
y_test = list(y_test)
plt.plot(y_test, color='blue',label='Actual')
plt.plot(y_test_pred, color='red',label="Predicted")
plt.title("Comparison")
plt.show()
#We can see that the values are almost overlapping
#Hence our model predicted very well
```



In [ ]: