Gold Price Prediction

Data collection:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [11]:
```
!pip install xlrd
```

```
Requirement already satisfied: xlrd in c:\users\client\anaconda3\lib\site-packages (2.0.1)
```

In [2]:
```
data= pd.read_excel("gold_price.xlsx")
data.head()
```

Out[2]:

	Date	SPX	GLD	USO	SLV	EUR/USD
0	2008-02-01 00:00:00	1447.160034	84.860001	78.470001	15.180	1.471692
1	2008-03-01 00:00:00	1447.160034	85.570000	78.370003	15.285	1.474491
2	2008-04-01 00:00:00	1411.630005	85.129997	77.309998	15.167	1.475492
3	2008-07-01 00:00:00	1416.180054	84.769997	75.500000	15.053	1.468299
4	2008-08-01 00:00:00	1390.189941	86.779999	76.059998	15.590	1.557099

SPX=stock market index USO= US oil fund GLD= gold price SLV= silver price

In [12]:
```
data.shape
```

Out[12]:
```
(2290, 6)
```

In [13]:
```
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 [14]:
```
data.dtypes
```

Out[14]:
```
Date        object
SPX        float64
GLD        float64
USO        float64
SLV        float64
EUR/USD    float64
dtype: object
```

1 categorical feature 4 numerical features

In [15]:
```
data.describe()
```

Out[15]:

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

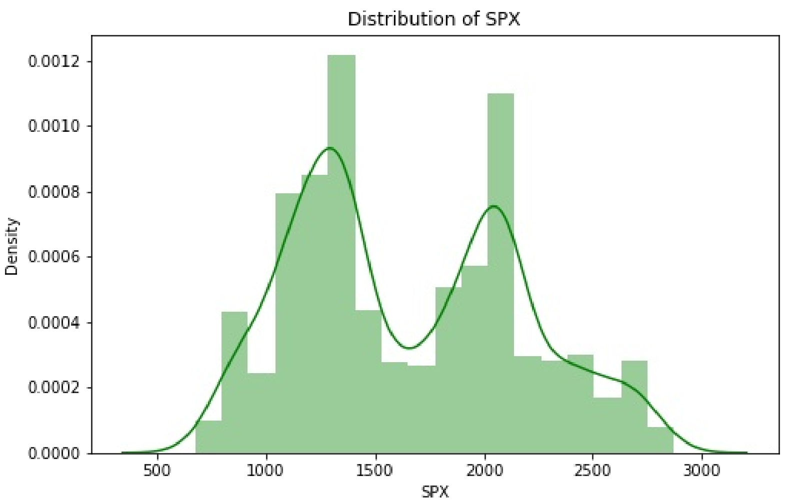
features: date, SPX, USO, SLV, EUR/USD target: GLD

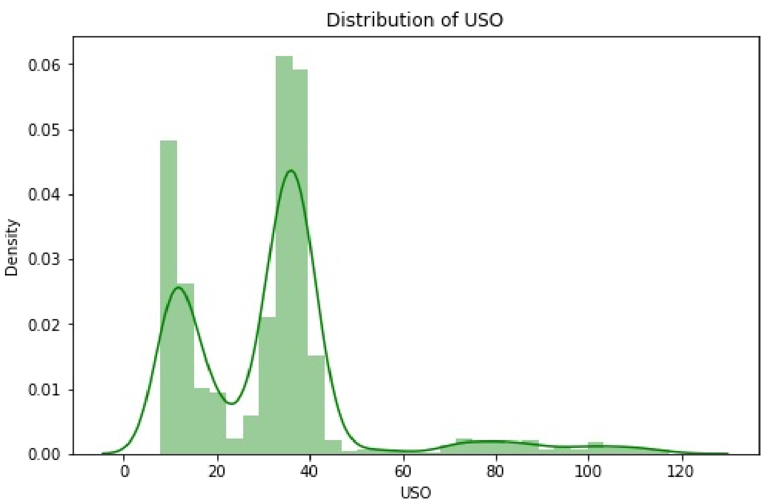
EDA:

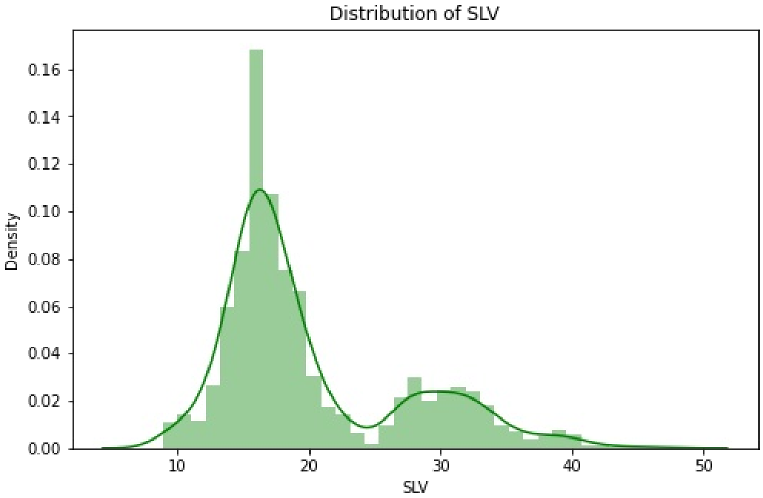
In [16]:
```
# checking null values:
data.isnull().sum()
```

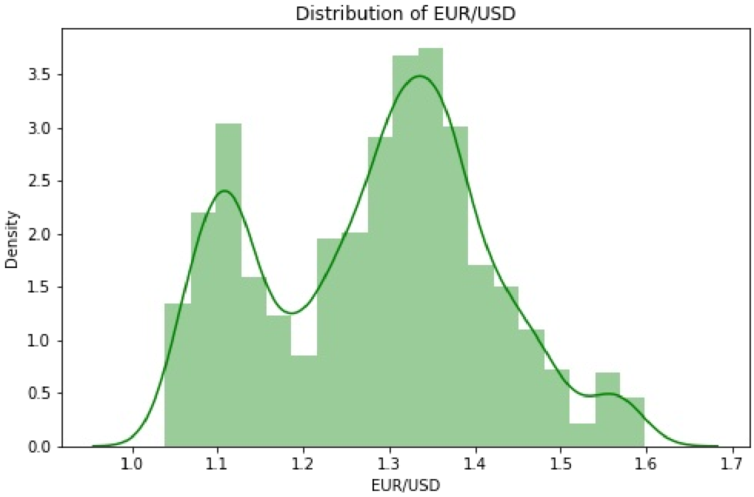
Out[16]:
```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

In [17]:
```
#distribution of numerical features:

features = ['SPX', 'USO', 'SLV', 'EUR/USD']

for feature in features:
    plt.figure(figsize=(8,5))
    sns.distplot(data[feature],kde=True, color='green')
    plt.xlabel(feature)
    plt.title(f'Distribution of {feature}')
    plt.show()
```



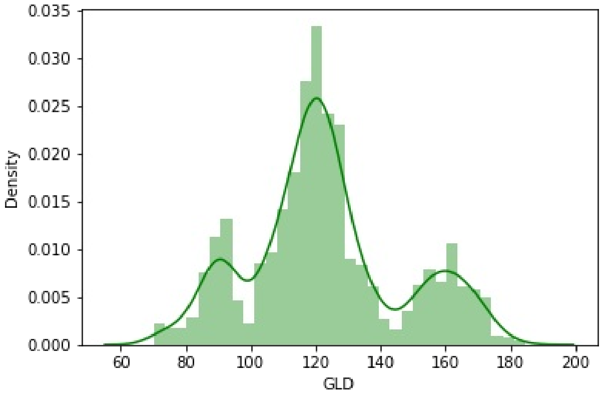






In [18]:
```
#ditribution of the target:
sns.distplot(data['GLD'],color='green')
plt.figure(figsize=(8,5))
```

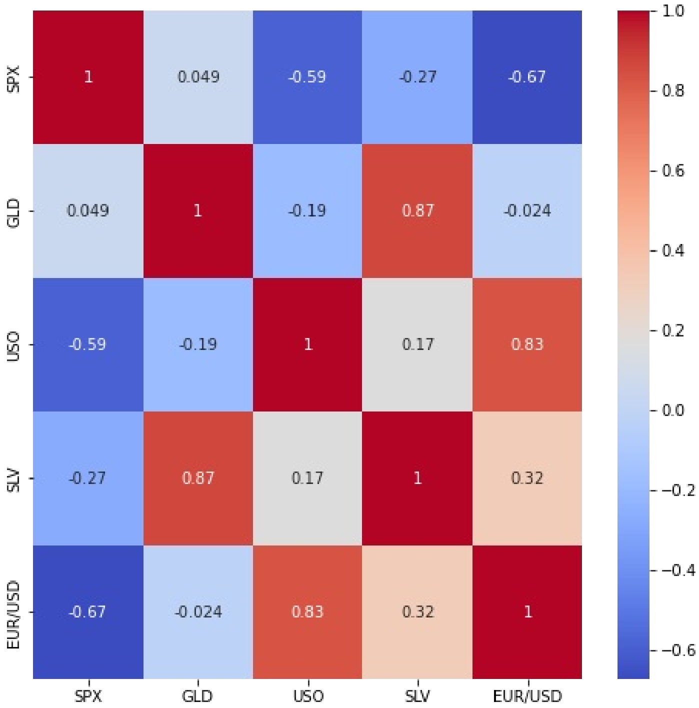
Out[18]: <Figure size 576x360 with 0 Axes>



<Figure size 576x360 with 0 Axes>

In [19]:
```
#correlation:
correlation=data.corr()
plt.figure(figsize=(8,8))
sns.heatmap(correlation, annot=True , cmap="coolwarm")
```

Out[19]: <AxesSubplot:>

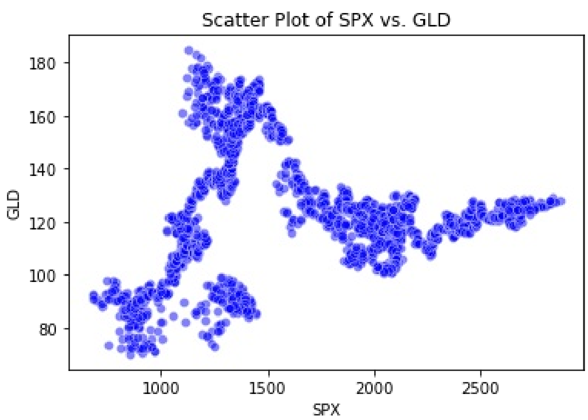


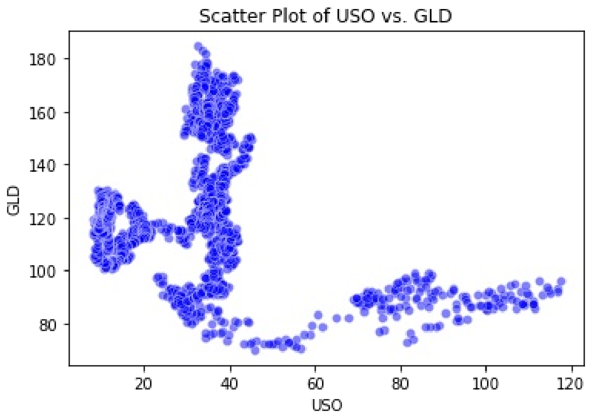
In [20]:
```
print(correlation['GLD'])
```

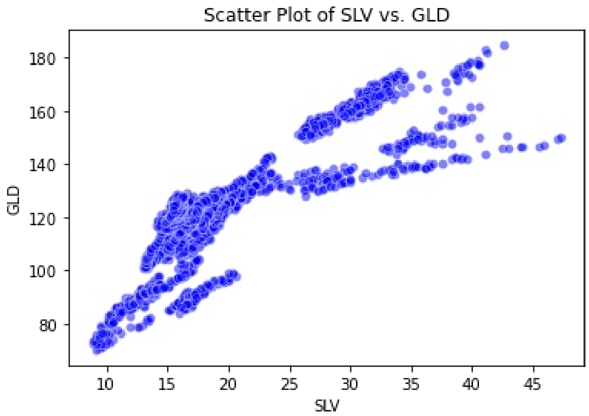
```
SPX        0.049345
GLD        1.000000
USO       -0.186360
SLV        0.866632
EUR/USD   -0.024375
Name: GLD, dtype: float64
```

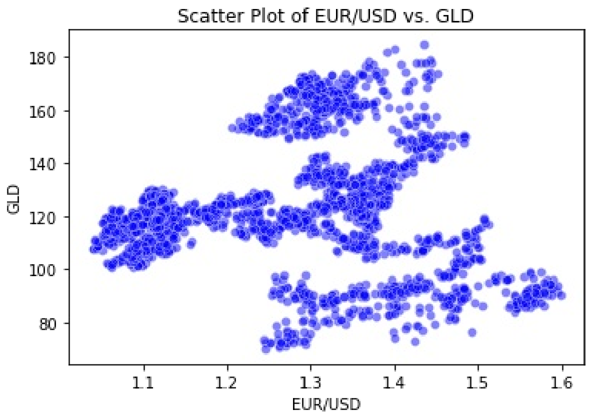
SLV is highly correlated with GLD

In [21]:
```
# relationship between GLD and numerical features:
features = ['SPX', 'USO', 'SLV', 'EUR/USD']
for feature in features:
    sns.scatterplot(x=data[feature], y=data['GLD'], color='blue', alpha=0.5)
    plt.xlabel(feature)
    plt.ylabel('GLD')
    plt.title(f'Scatter Plot of {feature} vs. GLD')
    plt.show()
```









Training data:

In [22]:
```
x=data.drop(['Date', 'GLD'], axis=1)
x
```

Out[22]:

	SPX	USO	SLV	EUR/USD
0	1447.160034	78.470001	15.1800	1.471692
1	1447.160034	78.370003	15.2850	1.474491
2	1411.630005	77.309998	15.1670	1.475492
3	1416.180054	75.500000	15.0530	1.468299
4	1390.189941	76.059998	15.5900	1.557099
...	...	...	...	...
2285	2671.919922	14.060000	15.5100	1.186789
2286	2697.790039	14.370000	15.5300	1.184722
2287	2723.070068	14.410000	15.7400	1.191753
2288	2730.129883	14.380000	15.5600	1.193118
2289	2725.780029	14.405800	15.4542	1.182033

2290 rows × 4 columns

In [ ]:

```
y=data['GLD']
```

In [5]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.1,random_state=42)
```

In [6]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
regressor = RandomForestRegressor(n_estimators=100)
regressor.fit(x_train,y_train)
```

Out[6]:

```
▾ RandomForestRegressor
RandomForestRegressor()
```

Prediction

In [7]:

```
test_data_prediction = regressor.predict(x_test)
print(test_data_prediction)
```

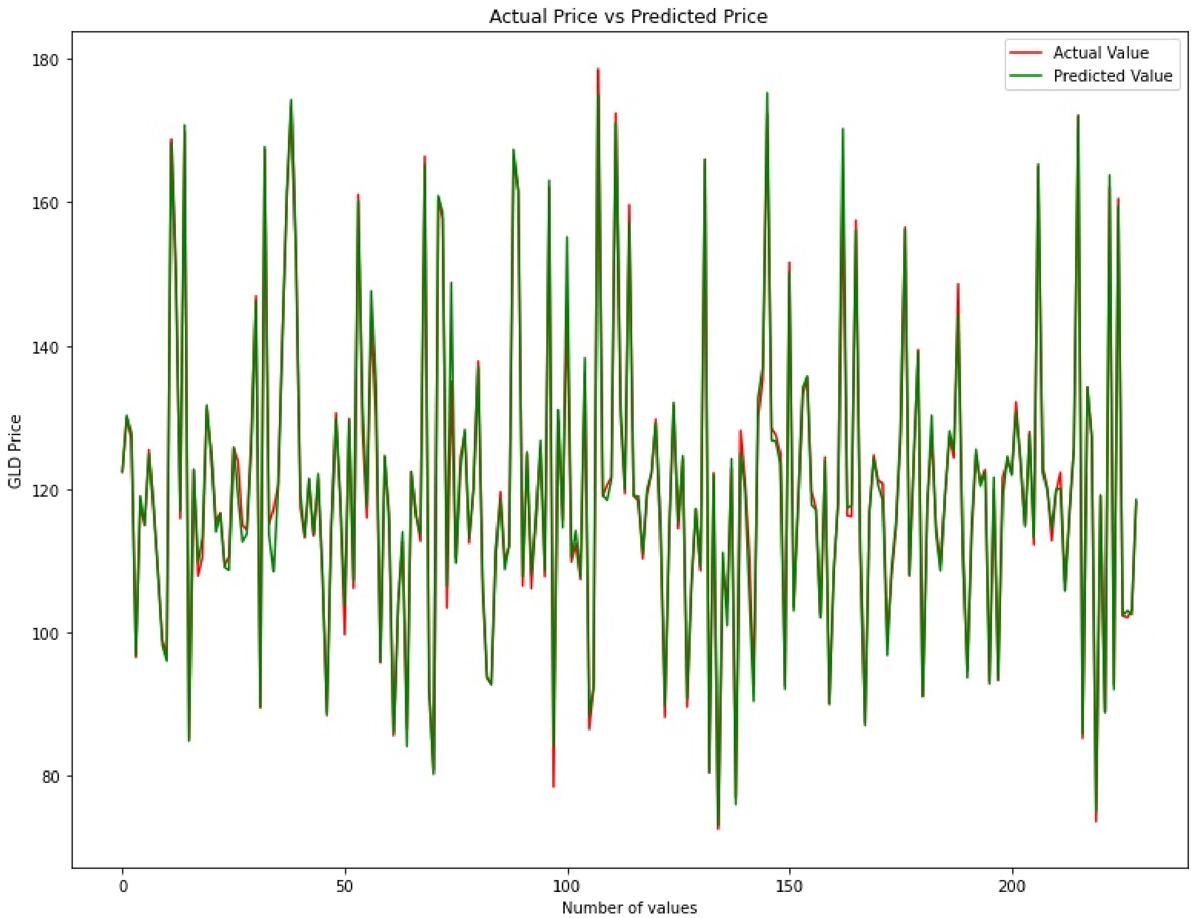
```
[122.46049952 130.21770053 127.75880025  96.74959813 118.99650056
 114.92250019 124.85210133 117.72389911 107.96360115  98.17039977
  95.96789868 168.19159885 148.67700134 116.91869982 170.72830087
  84.81440092 122.71879868 109.59359715 113.19680078 131.70060172
 124.4358987  113.99000123 116.58129966 109.07479951 108.6633019
 125.78249931 118.71749948 112.58709932 113.69930147 125.17939943
 146.25980211  89.49959944 167.69349909 113.55829926 108.47630133
 120.22730054 141.36979841 161.32560068 174.27229842 152.82720233
 119.46640096 113.38560056 121.41449935 113.73439927 122.08850042
 108.02900099  88.40559886 114.33529924 129.73640048 118.49409949
 103.79129984 129.60980042 107.26469863 160.14250149 131.67479977
 117.52740036 147.57380032 134.55000136  95.85510143 124.63420152
 115.15619862  86.01260091 104.28709924 113.99210067  84.06170003
 122.37380025 116.13309967 113.63660219 165.25270236  91.87199988
  80.23180094 160.91080112 158.52300198 106.44830009 148.76270177
 109.64099804 122.66070047 128.27180092 113.21679863 120.09010036
 136.94529685 107.34190086  93.81070064  92.67749877 111.2963003
 118.50040009 108.78339904 112.25619935 167.30229793 161.36039798
 107.73119884 125.11919974 108.14920024 115.79590164 126.71929874
 108.64459901 163.02490271  84.11159882 131.01460256 114.61339991
 155.13859964 110.35689845 114.16920017 107.7142996  138.24709905
  88.36929947  92.50419939 174.90080251 119.13400059 118.38180017
 121.3303     171.03979818 131.84960005 119.97040024 156.9854022
 119.0270994  118.96469956 110.9517997  120.00799894 122.22149994
 129.19859747 114.91400012  89.80469978 114.06260139 132.06829844
 115.44190152 124.60060016  90.69590029 106.98550076 117.21870115
 109.16539971 165.98290158  80.50950074 121.92499907  73.07280156
 111.11859936 100.90030125 124.15599997  75.96459997 124.91009955
 119.80490056 105.26949985  90.352199   132.5883002  136.93950239
 175.26540037 126.79239939 126.60839966 123.49509949  92.06889972
 150.13230129 102.99239907 117.56869993 134.18289804 135.68439736
 117.77420072 117.10580106 102.03079819 123.94229894  89.90629946
 108.35729906 117.847999   170.19110108 117.31480063 117.69319954
 156.08240078 111.11810061  86.96509909 116.65370125 124.28349932
 120.54580182 118.44579973  96.77989868 109.06920005 115.19939932
 127.74200102 156.21640135 108.044901   124.13429942 139.12880221
  91.00860057 117.87330083 130.23440057 113.88209956 108.58999927
 119.31410052 128.05219956 125.30810065 144.32680114 112.43440086
  93.64050018 115.1388006  125.49720016 120.41220146 122.33530017
  92.75930091 121.58079908  93.34680078 119.05110032 124.50519974
 121.89650034 130.76230019 124.32769897 114.77870158 127.59680125
 113.21640086 165.28289971 122.24869822 119.67600161 114.51979967
 119.90909978 120.07259954 105.7957014  117.18400013 125.74059929
 172.04379748  85.83159992 134.20419807 127.3215991   74.96330053
 119.11529992  88.81469977 163.77150204  92.0525999  159.39300116
 102.36759914 103.0177995  102.48479854 118.44709926]
```

In [8]:
```python
# R squared error
error_score = metrics.r2_score(y_test, test_data_prediction)
print(f"R squared error: {error_score} ")
```

```
R squared error: 0.991240221063502
```

In [9]:
```python
y_test = list(y_test)
```

In [27]:
```python
plt.figure(figsize=(13, 10))
plt.plot(y_test, color='red', label='Actual Value')
plt.plot(test_data_prediction, color='green', label='Predicted Value')
plt.title('Actual Price vs Predicted Price')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js