In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
```

In [4]:
```
# loading the csv data to a Pandas DataFrame
gold_data = pd.read_csv('gld_price_data.csv')
```

In [5]:
```
# print first 5 rows in the dataframe
gold_data.head()
```

Out[5]:

	Date	SPX	GLD	USO	SLV	EUR/USD
0	1/2/2008	1447.160034	84.860001	78.470001	15.180	1.471692
1	1/3/2008	1447.160034	85.570000	78.370003	15.285	1.474491
2	1/4/2008	1411.630005	85.129997	77.309998	15.167	1.475492
3	1/7/2008	1416.180054	84.769997	75.500000	15.053	1.468299
4	1/8/2008	1390.189941	86.779999	76.059998	15.590	1.557099

In [6]:
```
# print last 5 rows of the dataframe
gold_data.tail()
```

Out[6]:

	Date	SPX	GLD	USO	SLV	EUR/USD
2285	5/8/2018	2671.919922	124.589996	14.0600	15.5100	1.186789
2286	5/9/2018	2697.790039	124.330002	14.3700	15.5300	1.184722
2287	5/10/2018	2723.070068	125.180000	14.4100	15.7400	1.191753
2288	5/14/2018	2730.129883	124.489998	14.3800	15.5600	1.193118
2289	5/16/2018	2725.780029	122.543800	14.4058	15.4542	1.182033

In [7]:
```
# number of rows and columns
gold_data.shape
```

Out[7]:
```
(2290, 6)
```

In [8]:
```
# getting some basic informations about the data
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 [9]:
```
# checking the number of missing values
gold_data.isnull().sum()
```

Out[9]:
```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

In [10]:
```
# getting the statistical measures of the data
gold_data.describe()
```

Out[10]:

	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

In [11]:

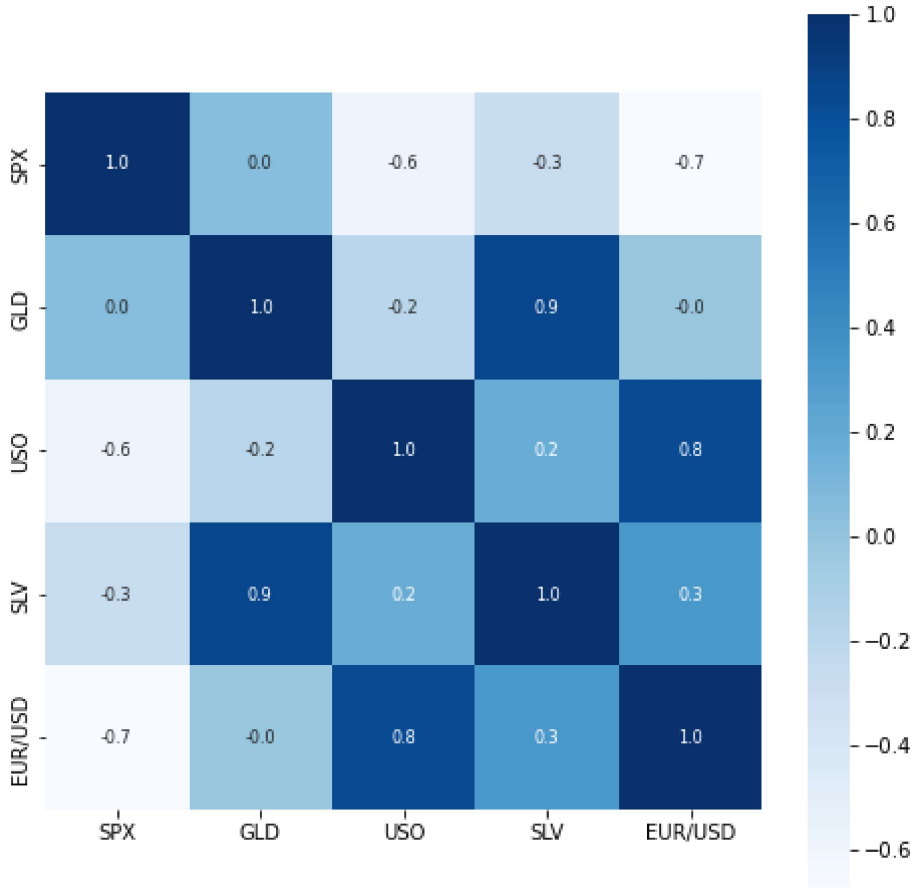
```
#Correlation
correlation = gold_data.corr()
```

In [12]:

```
# constructing a heatmap to understand the correlatiom
plt.figure(figsize = (8,8))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f',annot=True, annot_kws={'size':8}, cmap='Blues')
```

Out[12]:

```
<AxesSubplot:>
```



In [13]:
```
# correlation values of GLD
print(correlation['GLD'])
```

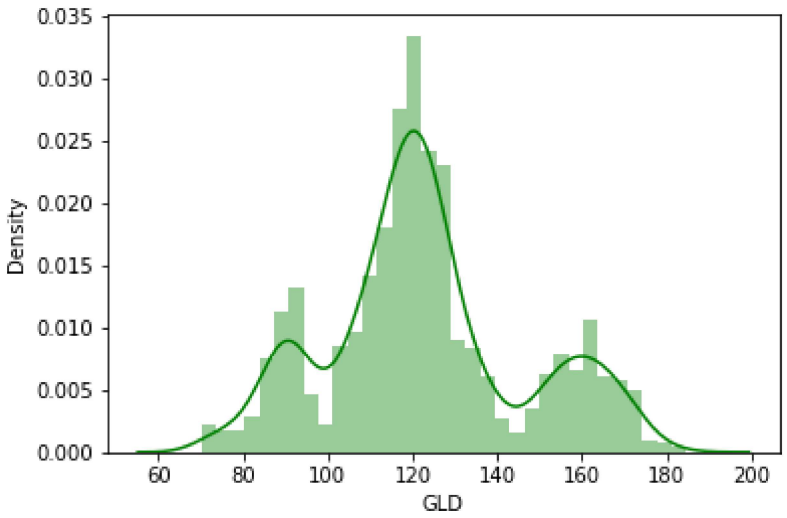
```
SPX        0.049345
GLD        1.000000
USO       -0.186360
SLV        0.866632
EUR/USD   -0.024375
Name: GLD, dtype: float64
```

In [14]:
```
# checking the distribution of the GLD Price
sns.distplot(gold_data['GLD'],color='green')
```

```
C:\Users\ACER\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and
will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexib
ility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[14]:

```
<AxesSubplot:xlabel='GLD', ylabel='Density'>
```



In [15]:

```
#Splitting the Features and Target
X = gold_data.drop(['Date','GLD'],axis=1)
Y = gold_data['GLD']
```

In [16]:

```
print(X)
```

```
              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 x 4 columns]
```

In [17]:
```
print(Y)
```

```
0        84.860001
1        85.570000
2        85.129997
3        84.769997
4        86.779999
           ...
2285    124.589996
2286    124.330002
2287    125.180000
2288    124.489998
2289    122.543800
Name: GLD, Length: 2290, dtype: float64
```

In [19]:
```
#Splitting into Training data and Test Data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=2)
```

In [20]:
```
#Model Training: Random Forest Regressor
regressor = RandomForestRegressor(n_estimators=100)
```

In [21]:
```
# training the model
regressor.fit(X_train,Y_train)
```

Out[21]:
```
RandomForestRegressor()
```

Model Evaluation

In [22]:
```
# prediction on Test Data
test_data_prediction = regressor.predict(X_test)
```

In [23]:
```
print(test_data_prediction)
```

```
[168.52249998  81.87799985 116.17480029 127.66370059 120.6764013
 154.72059752 150.79839864 125.94770051 117.41029875 125.90300087
 116.83540099 172.2186006  141.37609834 167.67439876 115.13280006
 117.51670066 138.24240326 170.09640109 159.35400293 159.36530003
 155.14260034 125.24110012 175.48599999 157.31330399 125.19000028
  93.59399965  77.86650014 120.28449995 119.09979912 167.56519912
  88.17200035 125.15530028  91.13950056 117.67370026 121.10019913
 136.38430058 115.53800095 114.82420065 147.2666997  107.23260104
 104.6367026   87.16839788 126.46150092 117.62190025 154.58429918
 119.55010021 108.32299956 108.31959839  93.38210082 127.24349762
  75.00850038 113.68519904 121.07140037 111.3801992  118.81439866
 120.3660993  160.39769905 168.0338011  147.13259642  86.14199861
  94.32030044  86.87719875  90.37099987 118.98460105 126.42060061
 127.5131     167.40120023 122.2327993  117.23119911  98.7751003
 168.43710086 142.96609721 132.04630212 121.33580234 121.21409937
 119.82860066 114.37240193 118.51140045 107.02650094 127.87050031
 113.64939973 108.01229994 116.71100082 119.584099    88.78600057
  88.18589848 146.11610229 127.54100001 113.39580026 110.38959848
 108.13769913  77.59309882 170.13300183 113.98389911 121.57189934
 127.82650148 154.98439818  91.62879951 136.64570107 159.41420312
 125.68580034 125.48760038 130.32840241 114.89280127 119.82800008
  92.10809976 110.52739899 165.56259992 157.41179947 114.14839939
 106.59550143  79.65709949 113.28290032 125.77350078 107.34309887
 119.27770074 155.576303   160.13159954 120.44059986 134.70190317
 100.94270059 117.84719789 119.36490039 113.0004006  102.78059878
 160.28699837  99.17010036 147.48479891 125.67740134 169.75219897
 125.58479892 127.35069719 127.28920161 113.78639941 112.81910078
 123.45739911 102.2145992   89.19010001 125.01689939 101.6932995
 107.16729904 113.81100091 117.24120048  99.48869964 121.59050067
 163.43349841  87.46009831 106.80629968 117.39350074 127.67730118
 124.07950069  80.82949944 120.2137005  157.62559823  87.8628998
 110.51589886 118.87499914 172.39089881 103.04189914 105.28560051
 122.49110026 157.61889793  87.2924985   93.16020053 112.9902004
 177.10309969 114.45769995 119.33050016  94.65050104 125.73850037
 165.84600079 114.91480072 116.82430121  88.25909858 148.39449962
 120.10709947  89.68560008 111.71220017 116.77030024 118.78750139
  87.97439948  94.16650021 116.98190059 118.46720189 120.2784
 126.63359836 121.94469977 149.75740065 165.64510085 118.64349933
 120.46470157 150.92100012 118.22539933 173.08029883 105.20979924
 104.93020119 148.80279995 113.64440066 124.88150107 147.43379959
 119.51790105 115.23940051 112.53419986 113.38950203 142.01970133
 117.84829774 102.85469982 115.85850095 103.92490168  99.14060053
 117.57620061  90.69509996  91.58020024 153.33999925 102.68699969
 154.87210077 114.26300184 138.82430087  90.22649845 115.47979944
```

114.99179972	123.0743001	121.7986002	165.34720195	92.96839917
135.78220121	121.35109918	120.57360135	104.67420011	144.14450237
121.7281992	116.68770042	113.62330062	127.15539727	122.57739967
125.84889987	121.17160045	86.94679898	132.17600149	143.81940198
92.59469966	159.14939976	159.57300237	126.31229879	165.11589907
108.91449959	109.90160088	103.71659847	94.26730126	127.86440263
106.99630075	161.30819992	121.61070057	131.95109999	130.30780215
160.90569882	90.22279909	174.99870171	127.86060005	126.76499854
86.57759949	124.48989923	150.35089702	89.7120001	107.2369995
109.06869985	84.05179911	136.00039988	155.04220188	139.17100329
73.71060046	152.56350098	126.10919994	126.69170006	127.46139899
108.69149925	156.13510048	114.50620133	116.9274013	125.26999938
154.12900176	121.39260003	156.3713991	92.90450067	125.54730163
125.92340012	88.02190073	92.1388996	126.1830992	128.13270293
113.20860062	117.73439757	120.82820022	127.03159827	119.69290113
136.16700072	93.75649937	120.00700068	113.12250099	94.25269911
108.62180028	87.28079943	109.17049959	89.56629977	92.39480019
131.47360323	162.40390009	89.39909971	119.85260055	133.46430209
123.91180017	128.04540137	101.92699865	88.76969926	131.88970067
120.00170046	108.84689977	167.57990125	115.3320005	86.58039886
118.85160056	91.10939957	161.30160026	116.56600026	121.76350056
160.31639785	120.14369906	112.73349938	108.37609866	126.42069987
76.1349003	103.00439985	127.57500233	121.7888992	92.62219985
132.31750026	118.14520151	115.92869989	154.56480294	159.84700072
109.99279937	152.21999778	119.27350088	160.42130116	118.49380017
157.54449913	115.27389958	116.34630032	149.73729936	114.74810087
125.9910992	165.44749954	117.62719987	125.20529955	153.32130375
153.24840267	132.1338003	114.76540033	121.2913023	124.82690096
89.84340036	123.40949958	154.97250166	111.75470038	106.82329976
162.42940145	118.36129975	165.67240018	134.08180117	114.77369995
152.84389827	168.60819946	115.59460048	114.22680116	158.02889895
85.37289857	127.13360064	127.93150013	128.82020025	124.37300081
123.84680076	90.48820134	153.00489989	97.14219974	137.00769962
89.20399912	107.88289999	115.0490005	112.59940075	123.82149922
91.44539899	125.51820176	162.32679855	120.02189881	165.22510037
126.64489833	112.42970031	127.65319925	94.92939918	91.13349995
102.92339915	120.81280008	82.62949946	126.44679984	160.35970483
117.3638009	118.24939971	119.96779974	122.80259972	120.07220128
121.35370012	118.45160037	107.05059983	147.93169883	126.17809891
115.76770083	73.89429984	127.83140102	152.84310062	122.40639987
125.68550053	88.99670012	103.04569872	124.23500071	120.27079986
73.43420048	151.87159999	120.97820049	104.5992	86.36019758
115.16199903	172.21889833	119.88260012	160.35869788	113.15479969
121.51220035	118.28300122	95.97529977	118.55599982	126.07400046

```
 118.38609932  96.22160067 153.97050176 122.07910016 147.97499993
 159.58010286 113.84530034 122.48769921 150.05949819 127.39630025
 165.87300113 135.32350056 120.11859875 167.37249885 108.46129925
 121.74749856 138.37760088 107.1956988 ]
```

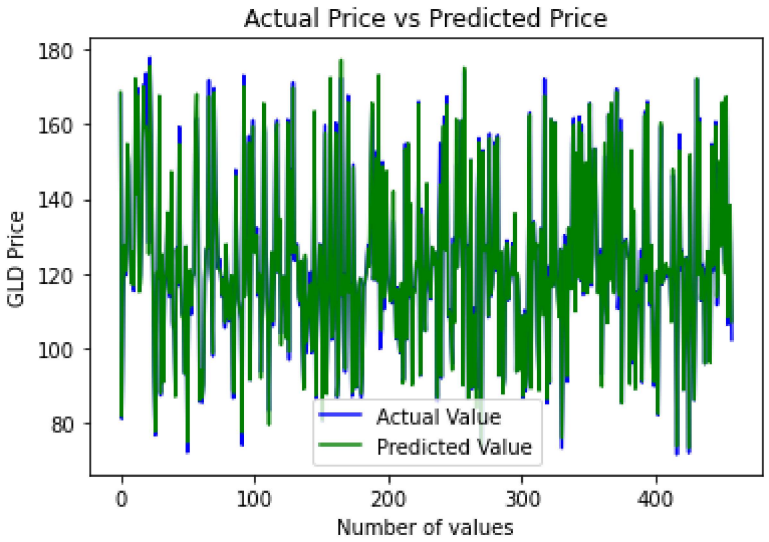
In [24]:
```
# R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error : ", error_score)
```

```
R squared error :  0.9894530675081028
```

Compare the Actual Values and Predicted Values in a Plot

In [25]:
```
Y_test = list(Y_test)
```

In [26]:
```
plt.plot(Y_test, color='blue', 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()
```



In [ ]: