ML PROJECT

GOLD PRICE PREDICTION

In [4]:
```
#import libraries
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 [5]:
```
# loading the csv data to a Pandas DataFrame
gold_data = pd.read_csv('gld_price_data.csv')
```

In [6]:
```
# print first 5 rows in the dataframe
gold_data.head()
```

Out[6]:

	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 [7]:
```
# print last 5 rows of the dataframe
gold_data.tail()
```

Out[7]:

	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 [9]:
```
# data info
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 [10]:

```
# checking the number of missing values
gold_data.isnull().sum()
```

Out[10]:

```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

In [11]:

```
# getting the statistical measures of the data
gold_data.describe()
```

Out[11]:

	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 [12]:

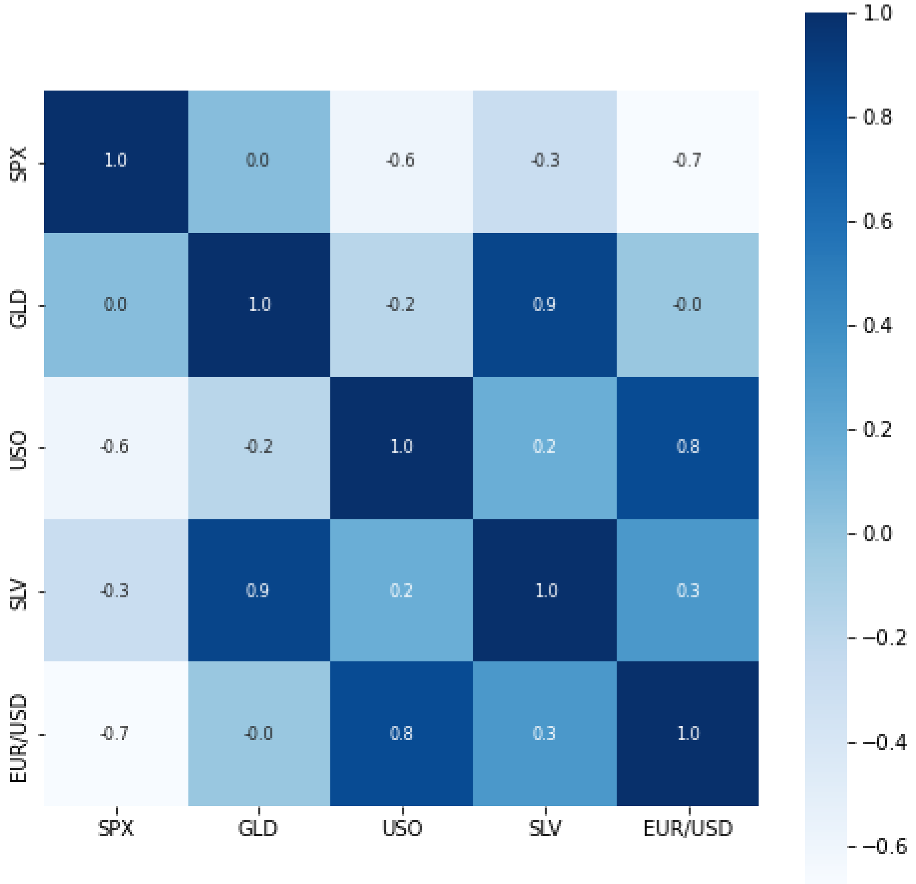
```
correlation = gold_data.corr()
```

In [9]:

```
# 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={'s
```

Out[9]:

```
<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
```

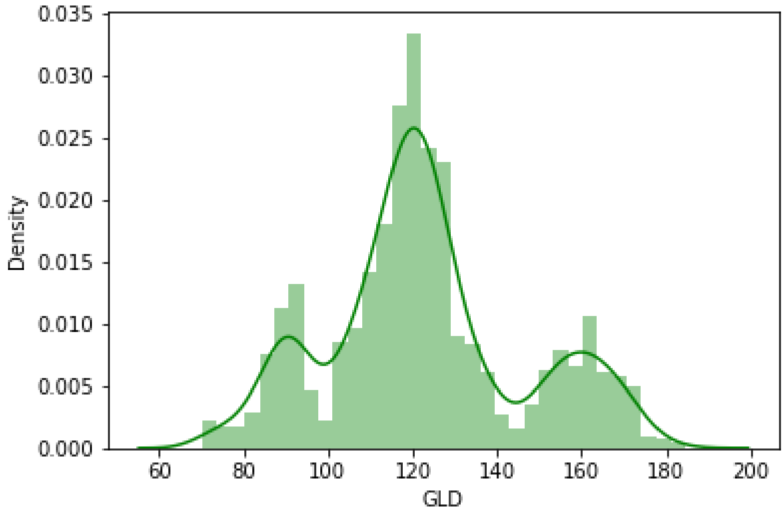
In [14]:

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

```
C:\Users\RANJAN KUMAR\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a future vers
ion. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) 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 [13]:
```
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 [16]:
```
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_st
```

In [40]:
```
#model training
model = RandomForestRegressor(n_estimators=100)
```

In [41]:

```
# training the model
model.fit(X_train,Y_train)
```

Out[41]:

```
RandomForestRegressor()
```

In [42]:

```
#model evaluation
# prediction on Test Data
test_data_prediction = model.predict(X_test)
```

In [43]:

```
print(test_data_prediction)
```

```
[168.80409977  81.94009991 116.10660017 127.68220091 120.66770154
 154.81749801 150.16559825 126.04690015 117.45569872 126.17130013
 116.46150123 172.32970076 142.00319816 167.92309816 115.21549999
 117.69660062 139.1975029  170.18820092 159.74720358 155.87299913
 155.11159997 125.22839961 176.22159953 157.16760332 125.19140041
  93.6242998   77.87380022 120.52559995 119.23669973 167.44709929
  88.09580016 125.21260004  91.14780101 117.70110025 121.1137992
 136.00810127 115.56970096 115.0534006  148.62120024 107.18870106
 104.24800252  87.10129796 126.47480087 118.2459001  152.20909881
 119.54290018 108.42839975 108.28899811  93.16600039 127.23529738
  74.66270035 113.60559932 121.53030016 111.15589935 118.9142988
 120.82529931 159.32530112 167.61670164 147.11959685  85.79799866
  94.27860037  86.65579876  90.49159991 118.69070086 126.41650036
 127.65759993 169.91709973 122.31399901 117.38639874  98.75200042
 167.52110172 142.83949838 132.13590262 120.96670222 121.14269962
 119.82990061 114.50270172 118.18440064 107.1760009  128.08130093
 114.00239949 107.65719995 116.8442007  119.585899    89.24730097
  88.28159869 146.56000167 127.21490006 113.35840009 110.14399854
 108.16979907  77.41889907 170.09940257 114.12989928 121.73619899
 127.95980152 155.13839864  91.75349942 136.10410076 158.83240337
 125.67660036 125.06510082 130.50020146 114.90390127 119.83250009
  92.24110012 110.31399887 167.75409968 156.74839858 114.19799952
 106.72400138  79.42820003 113.06820035 125.77370073 107.30709931
 119.07140112 155.66620261 159.52359844 119.99810004 134.64080222
 101.38379974 117.44999807 119.19000018 112.97060061 102.7795993
 159.62799755  98.91050037 147.92049946 125.47740133 169.69369879
 125.72359944 127.46779713 127.36430187 113.77819931 112.57360063
 123.74619926 102.21869906  89.24659999 124.54469977 101.71649939
 107.1554991  113.29770063 117.38400095  99.33459955 121.74450021
 163.67959907  87.32639885 106.82329995 117.23650044 127.78360103
 124.06320053  80.75839934 120.25150056 158.10609773  87.89379966
 110.34739909 118.86829921 172.54369865 103.03319897 105.79500047
 122.46180058 158.57199719  87.62559829  93.36430037 112.63670048
 177.44649922 114.43419964 119.23150043  94.55700106 125.94260034
 165.93060118 114.84160076 116.69760105  88.28079854 148.78730085
 120.26509984  89.46560018 112.35589996 117.73269996 118.75700122
  88.3814997   94.12669983 116.74030052 118.47820213 120.19170011
 126.72509844 122.01539956 149.60540006 165.60660095 118.60299923
 120.21960145 151.05110026 118.55009913 172.75129874 105.54079928
 104.94270119 149.34160107 113.57870075 124.7265009  147.37819982
 119.41520135 115.51300061 112.4852     113.36520238 141.66250144
 117.86079771 102.86940053 115.82710092 103.73150167  98.96700068
 117.35170073  90.85440002  91.66350052 153.40779874 102.68829978
 155.27500099 114.38020191 138.5083011   90.0682986  115.48909919
 114.6349998  123.30950023 121.83200032 165.15730114  92.94589949
 135.77180101 121.38649894 120.831401   105.00530018 141.00760272
 121.46259918 116.58440035 113.30200095 127.02299762 122.90109943
 125.68309925 121.22850049  86.88079911 132.55240178 144.75240191
  92.65019962 157.48299915 159.10900305 126.29159923 165.09609955
```

```
 108.95119921 109.77740053 103.72239833  94.39800084 127.87900314
 107.28440043 160.70339942 121.78940012 132.08500011 130.55370152
 160.13680007  90.13939885 175.43270183 128.24270085 126.82129832
  86.61529946 124.65629959 150.43509758  89.56810043 106.83669991
 109.14179989  84.4864987  136.13780051 154.9285024  139.6760033
  73.95830049 151.9841014  126.01379995 126.70869973 127.46779923
 108.66269915 156.4808998  114.54280101 116.98110121 125.07559957
 154.04790109 121.4272997  156.44579835  93.06420082 125.48690137
 125.77050058  87.6828004   92.1789992  126.25749944 128.17790296
 113.32370084 117.65599736 120.95090042 127.15909804 119.83770092
 136.51360113  93.8883993  119.77890036 113.181501    94.19229945
 108.94959985  87.26539904 108.98849952  89.66819961  92.34110001
 131.14710288 162.36580012  89.4627997  119.60300083 133.35220164
 123.80310005 128.42140213 102.01909849  88.87209868 132.10620031
 120.18880007 108.36189973 167.75840062 115.19830039  86.65249905
 118.74740048  90.97359936 161.58670038 116.54720063 121.64370002
 159.9101977  120.09119927 112.58659952 108.41849886 126.51860019
  75.91570048 103.01659985 127.69190297 121.79849879  92.65560017
 132.14310015 118.10480104 116.13349949 154.69410276 159.46690095
 110.16099936 154.895198   119.32320104 160.71390109 118.49280014
 158.39409981 115.0466999  116.75360037 148.66949904 114.83040084
 125.64449859 165.40409943 117.69820021 125.0170995  153.16640375
 153.41640287 132.2155003  114.92120047 121.06720197 125.03750096
  89.75960045 123.24799987 154.65240115 111.77430051 106.83339975
 161.1231008  118.31069976 165.80390056 133.81570068 114.91659947
 153.08399962 168.66330001 115.44319987 114.00300133 157.93129856
  85.18349891 127.21200004 127.9843006  128.6684002  124.17810057
 123.91330083  90.44960034 152.98140018  96.98659986 137.00179966
  88.92079899 107.63270006 115.00410035 112.36340082 124.17579908
  91.3535989  125.35370126 162.34229915 119.80649921 165.12090068
 126.72529801 112.28710008 127.51769922  94.89309902  91.03489988
 103.39819904 120.69459988  83.3100995  126.48600025 160.631605
 117.21870099 118.18250001 120.01379972 122.84529974 120.16490132
 121.64809969 118.53150097 107.17119991 148.36090022 126.24119888
 115.925201    74.01829997 127.81680094 153.55770107 122.05099985
 125.63790042  88.94359998 103.30489845 124.44800058 120.30190039
  73.41140069 151.47050036 121.36680021 104.69680005  86.35849788
 115.12819888 172.21769836 119.8695004  160.26359788 113.17789973
 121.17360039 118.63670123  96.0411999  118.98800012 126.15330041
 118.62049965  96.14080077 153.71960179 122.05820041 147.1926995
 159.46250262 114.05810039 122.60309947 150.35629796 127.27180059
 165.91959979 134.86700041 119.96999977 167.47229819 108.40889898
 121.57689862 139.83020112 107.00379902]
```

In [44]:

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

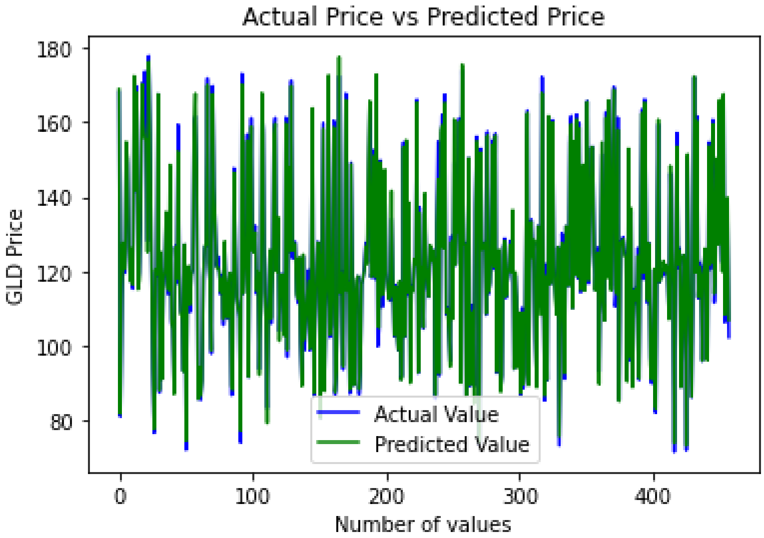
```
R squared error :  0.9888392523838178
```

In [47]:

```
#Compare the Actual Values and Predicted Values in a Plot
Y_test = list(Y_test)
```

In [48]:

```
# visualisation of Actual price vs Predicted price
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 [49]:

```
#Testing scores
testing_test_data_prediction = model.score(X_test, Y_test)
print("Model Score/Performance on Testing data",testing_test_data_prediction)
```

```
Model Score/Performance on Testing data 0.9888392523838178
```

In [50]:

```
training__test_data_prediction = model.score(X_train, Y_train)
print("Model Score/Performance on Training data",training__test_data_prediction)
```

```
Model Score/Performance on Training data 0.9983963812102966
```

In [51]:

```
# Checking working of the model
input_data=(1447.160034 , 78.470001 , 15.1800 , 1.471692)
input_array=np.asarray(input_data)
reshape_data =input_array.reshape(1,-1)
new_pred =model.predict(reshape_data)
print('Price of GOLD predicted by the model : = ')
print(new_pred)
```

```
Price of GOLD predicted by the model : =
[84.97980009]
```

THANK YOU