Installing Dependencies ¶

In [1]:

```
#!pip install numpy
```

In [2]:

```
#!pip install pandas
```

In [3]:

```
#!pip install seaborn
```

In [4]:

```
#!pip install sklearn
```

In [5]:

```
#!pip install matplotlib
```

Importing the Libraries

In [6]:

```
# will hide errors like outdated verisions
import warnings
warnings.filterwarnings('ignore')
```

In [7]:

```
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
```

Data Collection and Processing

In [8]:

```
# loading the csv data to a Pandas DataFrame
data = pd.read_csv('gold_price_data.csv')
```

In [9]:

```
df = data.copy() # creating a copy for further use
```

In [10]:

```
# print first 5 rows in the dataframe
data.head()
```

Out[10]:

	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 [11]:

```
# print last 5 rows of the dataframe
data.tail()
```

Out[11]:

	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 [12]:

```
# number of rows and columns
data.shape
```

Out[12]:

```
(2290, 6)
```

In [13]:

```
# getting some basic informations about the data
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]:

```
# checking the number of missing values in Gold Data
data.isnull().sum()
```

Out[14]:

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

In [15]:

```
data.duplicated().sum() # checking the duplicate values in Gold Data
```

Out[15]:

```
0
```

In [16]:

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

Out[16]:

	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

Correlation:

1. Positive Correlation -> Rising values in one variable align with increasing values in another
2. Negative Correlation -> Rising values in one variable align with decreasing values in another

In [17]:

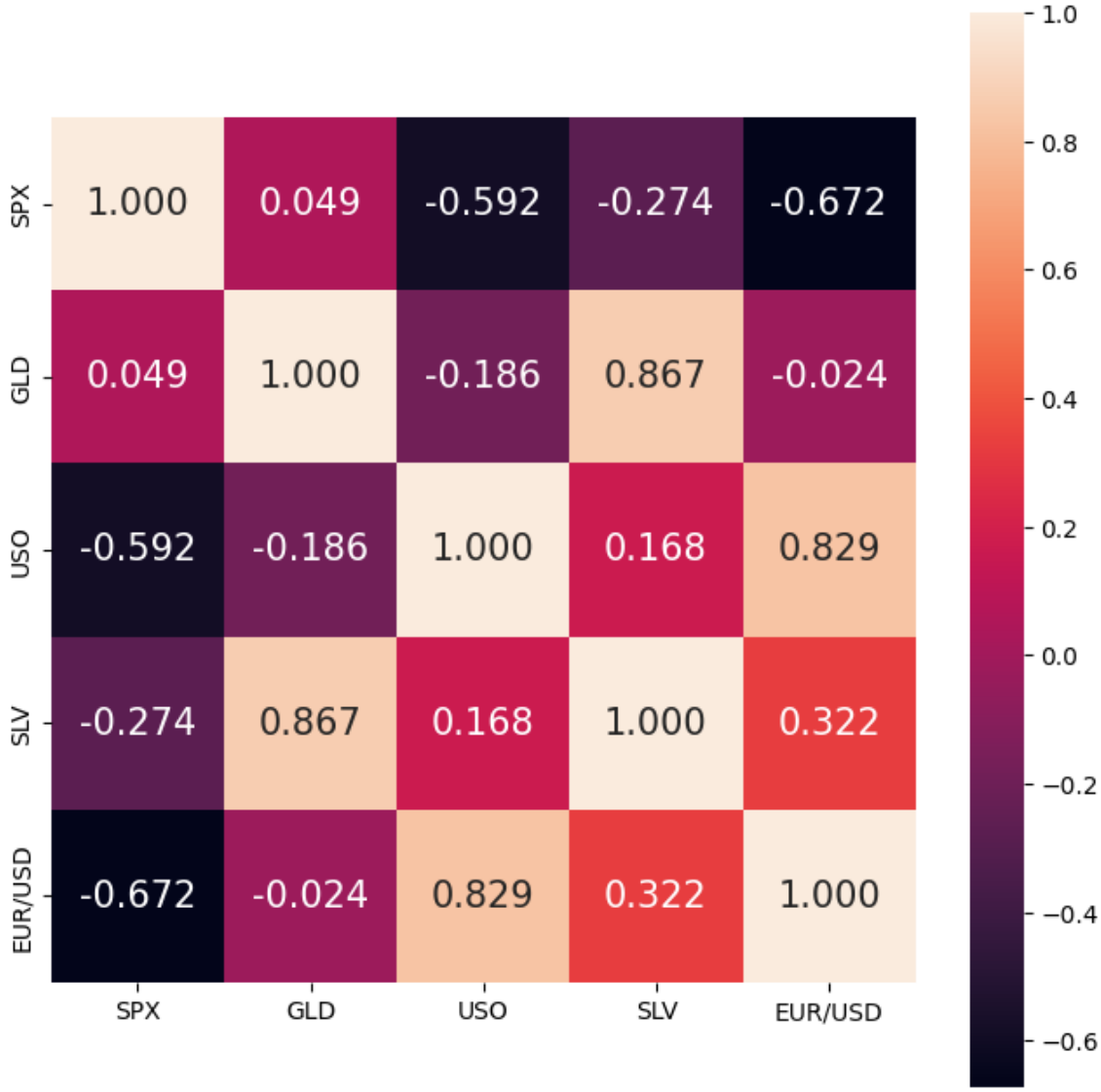
```
data = data.drop(["Date"], axis=1)
correlation = data.corr()
```

In [18]:

```
# constructing a heatmap to understand the correlatiom
plt.figure(figsize = (8,8))
sns.heatmap(correlation, cbar=True, square=True, fmt='.3f',annot=True, annot_kws=
```

Out[18]:

```
<Axes: >
```



In [19]:

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

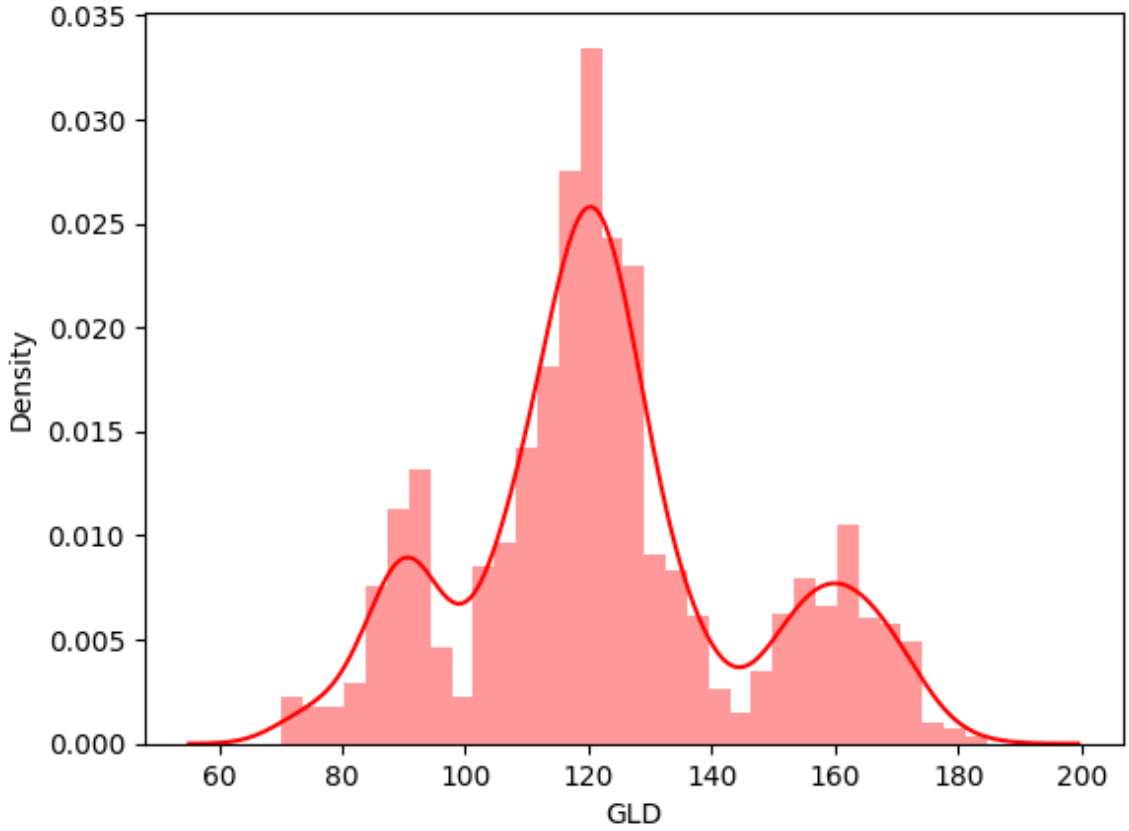
```
SPX        0.049345
GLD        1.000000
USO       -0.186360
SLV        0.866632
EUR/USD   -0.024375
Name: GLD, dtype: float64
```

In [20]:

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

Out[20]:

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



Splitting the Features and Target

In [21]:

```
# df is copy of data we created above
A = df.drop(['Date','GLD'],axis=1)
B = df['GLD']
```

In [22]:

```
print(A)
```

```
              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 [23]:

```
print(B)
```

```
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
```

Splitting into Training data and Test Data

In [24]:

```
X_train, X_test, Y_train, Y_test = train_test_split(A, B, test_size = 0.2, random_
```

Model Training: Random Forest Regressor

(n_estimators) represents number of trees in forest. Usually the higher number of trees better to learn data.

In [25]:

```
regressor = RandomForestRegressor(n_estimators=500)
```

In [26]:

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

Out[26]:

```
RandomForestRegressor
RandomForestRegressor(n_estimators=500)
```

Model Evaluation

In [27]:

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

In [28]:

```
print(test_data_prediction)
```

```
[122.42880716 130.53518274 127.57043968  96.61733736 118.91102075
 114.51123932 124.80924133 117.70951933 107.99598126  98.21913969
  95.54207935 167.83277817 149.34120095 116.17660066 170.76428147
  85.09599975 123.07791875 109.04711713 113.29302068 131.60998281
 124.29581878 113.63996074 115.89326079 108.68905979 108.48402125
 125.78461925 118.78991974 112.30945922 113.61394151 125.65411886
 146.07198155  89.4813399  167.79053963 113.78349929 108.45302111
 120.2143211  141.94849873 161.21500135 174.14803795 153.19468128
 119.38704106 113.54356043 121.34077954 113.7061394  122.07340764
 107.98574081  88.58035895 114.42721923 129.79484227 117.93816112
 103.74922028 129.79766245 107.20327847 160.7044236  131.54978029
 118.14789964 146.98146068 135.17472211  95.59900102 124.64984137
 115.13545869  86.16204094 104.25319911 113.83184067  84.22531923
 122.31490777 116.45729901 113.54124195 165.37986286  92.23638025
  80.37042085 160.98010107 158.47422166 106.64742008 149.47082118
 109.70149749 122.85212058 128.53248103 113.25465871 120.16084068
 135.80397733 107.41760083  93.82862094  92.64155867 111.38858053
 118.5411399  108.6560193  112.2437999  168.19839914 160.97253863
 107.57591864 125.2574602  108.19688031 115.27052187 126.69885816
 108.36367929 162.22796261  84.17155883 131.460783   114.62191994
 154.13734026 110.32453834 113.8362399  107.61663995 138.48452023
  88.22877948  92.38465909 175.47456113 119.01744063 118.67656012
 121.21978075 171.46663777 131.65220023 119.78994055 157.6522422
 118.84945847 119.29787983 110.65887944 119.90439923 122.0583001
 129.02915847 114.77816007  89.69517944 114.35408123 131.86527917
 115.43010114 124.95015994  90.9505003  106.81236058 117.1526213
 109.61433934 166.07866192  80.51798075 121.88677429  72.88146158
 111.3391393  100.29426095 124.14390005  75.96761998 125.1696592
 119.8217406  105.08493993  90.60213914 132.04925971 137.03540207
 176.11193958 126.64335929 126.61331903 123.88227993  92.0101188
 150.09786131 102.87735893 117.42038003 134.82489826 135.6059992
 118.2027403  116.98846153 102.17249789 124.01971885  89.61703985
 108.24433915 117.50966042 167.92208047 116.99490063 117.81691964
 155.60084121 111.32238032  87.17755908 116.58492134 124.25113943
 120.61058222 118.2519203   96.46211864 109.3379799  115.02817919
 127.60542093 156.06236086 107.94460091 124.017939   139.60002217
  90.99522057 118.28128085 130.41346036 114.08353915 108.39915937
 119.30086037 127.91500033 125.65850024 145.40220064 112.42916079
  93.65329985 115.10256011 125.54756067 120.41036145 122.26646081
  92.74898078 121.36805874  93.33316054 119.13114022 124.37872021
 121.91828025 131.49276042 124.3780394  114.99784137 127.44670072
 113.23452079 165.31437963 122.30405789 119.55560199 114.06136018
 120.10467986 120.18943975 105.74864105 116.63828038 125.79627895
 172.12883742  85.69118006 134.77849804 127.83101876  74.01048058
 119.22299978  88.77851979 162.67988227  92.0536396  158.92026154
 102.40487828 102.90621941 102.52335874 118.49914699 165.39964072
 120.33634105 135.46561854  96.55161834 113.09823941 132.30932117
 145.67984013 125.79596026 101.57059998 125.35728093 159.92012104
 119.84034092 126.45790077 127.54500116 115.44831943 156.87246195
 128.83366009 114.14535957 176.96405885 119.86496192 119.19738113
 102.77301876 160.80511974 114.72588091 118.38907967 125.63331937
 116.70904103 114.68275959  90.84089967 101.39399996 132.00632083
 118.82620217 167.88231853 108.04990101  86.27640059  91.75019952
 155.91413922 158.07804027 153.05981917  72.8517798  120.87230017
 117.34452033 158.4409795  135.41195882 111.84687968 113.85211995
 160.40788152 125.5281796  119.76848085 117.91367979 158.43994238
 104.04453977  89.44117947  83.45841933  90.19319944 115.54118029
 113.56037993 119.44916107 119.57356085  79.28842005  90.7181406
 153.59542352 119.50694061 131.91444018 126.05002086 113.56970112
  82.55756057 118.16777872  89.74284022 118.05825958 163.12518253
 121.48252072 110.59284033 125.40647861 114.39146029 135.89438002
```

```
 80.19448096 163.79067969 132.53446119 164.0456604  127.65357926
 91.64839911 108.42073927 114.19489952 127.68468156 119.26530187
 92.49569926 132.31129933 162.56722065  72.24702173 112.08920025
108.73511966 113.65101813 120.2525811  111.96355967 120.61591967
118.53894194 126.28186069 125.7805613  109.25371987 167.56063984
166.25799903 112.28773928 169.48673745 112.05758035 161.59202118
127.37211869 167.6175792  135.13896159 109.27939855 167.18591972
116.65176142  72.68970116 113.57006008  93.47929975  87.85172032
104.09739894 125.81340064 123.39735766 167.80135869 121.47644071
 87.33193866 131.76917821 121.6845603  107.71853946 167.69368009
126.11279767 127.10578128 113.60144087 134.86192068 125.12118117
143.89003919 123.34363937 118.40228017 120.79653999 167.02039977
 71.60582054 163.14161934 166.17497873 118.3958804  103.64105826
127.94265867 154.25198032 172.05676048 135.08125745 126.89768006
124.07878037 153.32849776  87.91709992 131.08688188 111.03344081
164.89804082 156.51503926 166.95442005 121.5853399   90.10920031
132.22978188  99.73965966 127.55625815 127.87439825 108.69701932
 91.01679895 153.22320114  95.09789881  87.88081924 124.99163952
 87.1912179   94.57348095 113.56913959 156.40888341 147.89248049
105.20738021 166.42749741 111.31716012 128.42180015  90.79473985
109.69321958  76.27048088 110.74529991 163.47863921 155.10667886
152.57972152 161.92559981  92.12309879 117.89584164  93.39304119
129.72406027 117.49634032 117.32920036 124.22102042 120.8194475
 97.88277936 167.08106113 145.97626186 124.54951859 168.89979825
 84.16248013 166.87687793 130.35754258 119.86634185  88.43971995
119.99695877  83.63293881 118.78080106 113.87769866 116.64317924
154.553318   134.86290301 118.27754134 118.82665831 123.09981859
115.63268102 118.56534016 122.25974047 145.79804065 150.4525408
168.5144209   98.14259922 160.01435967  93.11218021 140.56665947
121.41086082  84.0259988  106.42335988 123.68127974 169.21051713
 93.50265887  96.56540072 152.98496029]
```

In [29]:

```
# R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error : ", error_score * 100)
```

```
R squared error :  99.0172005495421
```

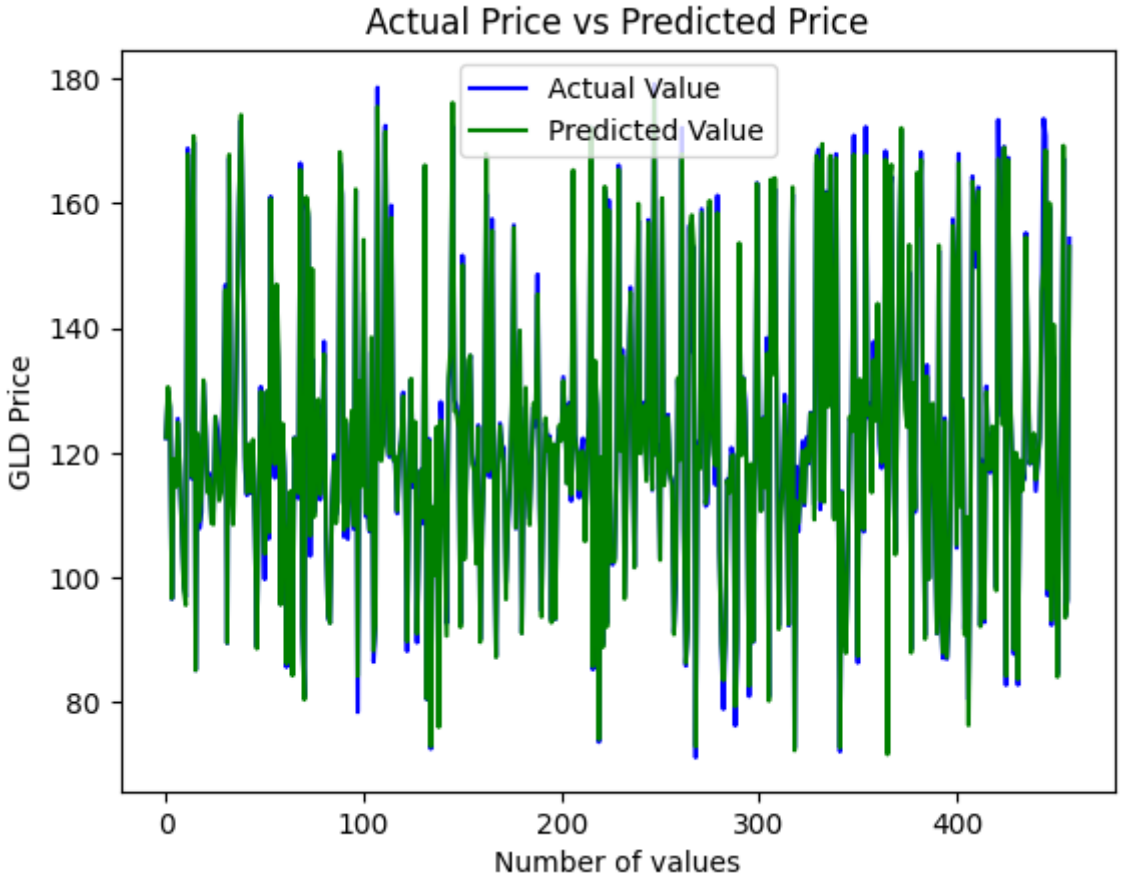
Compare the Actual Values and Predicted Values in a Plot

In [30]:

```
Y_test = list(Y_test)
```

In [31]:

```
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 [ ]: