ML PROJECT

GOLD PRICE PREDICTION

Ridhima Parmar

NIT Jalandhar

In [1]:
```
#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 [2]:
```
# loading the csv data to a Pandas DataFrame
gold_data = pd.read_csv('gld_price_data.csv')
```

In [3]:
```
# print first 5 rows in the dataframe
gold_data.head()
```

Out[3]:

	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 [4]:
```
# print last 5 rows of the dataframe
gold_data.tail()
```

Out[4]:

	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 [5]:
```
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 [6]:
```
# checking the number of missing values
```

```
gold_data.isnull().sum()
```

Out[6]:

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

In [7]:

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

Out[7]:

	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

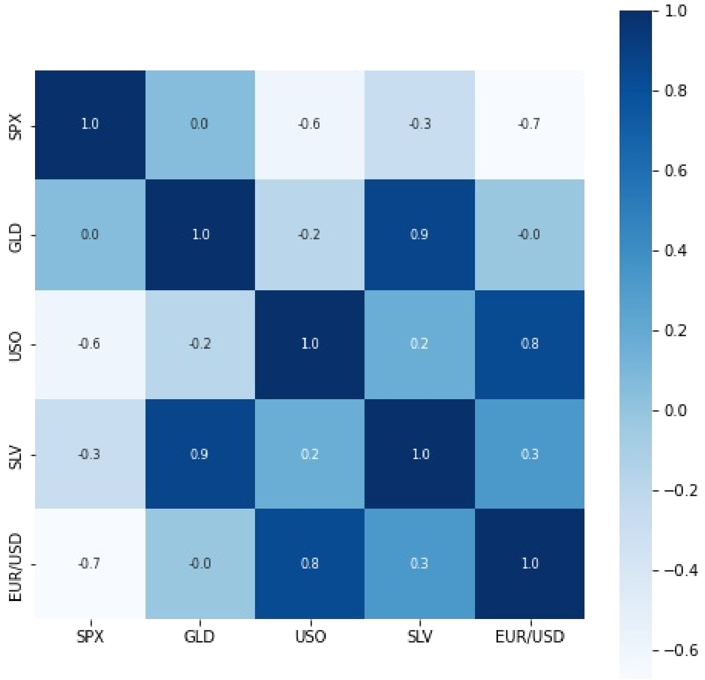
In [8]:

```
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={'size':8}, cmap='Blues')
```

Out[9]: <AxesSubplot:>



In [10]:

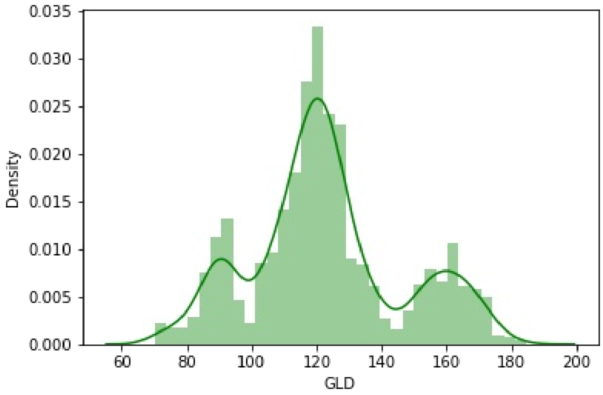
```
# correlation values of GLD
print(correlation['GLD'])
```

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

In [11]:
```
# checking the distribution of the GLD Price
sns.distplot(gold_data['GLD'],color='green')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[11]:
```
<AxesSubplot:xlabel='GLD', ylabel='Density'>
```



In [12]:
```
#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 [14]:
```
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 [15]:
```
#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 [16]:
```
#model training
model= RandomForestRegressor(n_estimators=100)
```

In [17]:
```
# training the model
model.fit(X_train,Y_train)
```

Out[17]:
```
RandomForestRegressor()
```

In [18]:
```python
#model evaluation
# prediction on Test Data
test_data_prediction =model.predict(X_test)
```

In [19]:
```python
print(test_data_prediction)
```

```
[168.6039992   82.05129992 115.80179983 127.67460097 120.74450159
 154.57609766 150.20369912 126.13710074 117.65099869 126.07870011
 116.50500129 172.45990074 141.71269845 167.76709825 115.18650015
 117.50960039 139.02510207 170.21730124 158.40550329 160.10529972
 154.98139993 125.39150005 176.09359982 157.9219034  125.18700025
  93.76589952  77.80650028 120.5291001  119.08429908 167.49939894
  88.23220048 125.12799985  91.24530074 117.63340052 120.95989904
 135.84520147 115.42570141 115.19090063 148.92449952 107.29580121
 104.55590243  87.15499789 126.28160059 117.86779938 152.51359912
 119.58220003 108.28659985 107.88829847  93.27180033 127.04089786
  74.91380046 113.70019936 121.12759996 111.29579921 119.00099869
 120.60559974 158.91039958 169.22570115 146.77999646  86.13799897
  94.38730013  86.89199914  90.48829976 119.00430084 126.37740063
 127.61600021 169.36450023 122.18139933 117.34619913  98.66460057
 168.26130175 143.06599842 131.89520268 121.12430187 121.16369923
 119.94690038 114.47900122 117.78290076 107.25390098 127.99610025
 114.23969965 107.6288     116.91630033 119.56659877  88.86650043
  88.28349866 146.88720243 127.3196001  113.75680009 109.71019846
 108.26459905  77.20389889 169.54710158 114.05649923 121.612799
 128.01180154 154.77289838  91.79789977 135.57950099 158.76650333
 125.62530032 125.72700066 130.56370212 114.76850093 119.76859976
  92.1994999  110.09189887 167.78359934 157.90869954 114.11949966
 106.89800134  79.34309995 113.25980043 125.7563008  107.21859929
 118.85130058 155.92090376 159.91739831 120.43559966 136.02560313
 101.57419994 117.430098   119.27800021 112.97790086 102.80759905
 160.501798    99.53140018 147.5295986  125.82370089 169.43189988
 126.07989845 127.40349713 127.50210144 113.88249938 112.99650052
 123.47979903 102.20539917  89.20299986 124.69849925 101.92799958
 107.28399954 112.90300086 117.53670053  99.87329992 121.95590036
 163.02679858  87.29059867 106.77219981 117.42540031 127.78370084
 124.04070081  80.65199933 120.47780069 158.05069782  87.98159957
 110.17569941 118.76449913 172.6322991  103.02949924 105.41140048
 122.70740028 157.95599749  87.76959856  93.42510043 112.82220029
 177.2963996  114.53699973 119.32320017  94.64880102 125.64639969
 166.13770137 114.76480046 116.93810112  88.35529863 148.61490046
 120.36249907  89.42449951 111.95880014 117.42419976 118.86230106
  88.05959945  94.06940012 116.75979998 118.47050165 120.17980041
 126.83049783 121.91549965 149.28320003 165.74       118.67779972
 120.39900144 150.15780031 118.48399926 172.96999947 105.23939931
 105.02050117 148.84100046 113.99030065 124.84640072 147.21959919
 119.53380112 115.37890037 112.4012999  113.32100233 140.87940161
 117.8720978  102.87940043 115.81880114 103.81420172  99.16280044
 117.41170068  90.53030042  91.79660012 153.39299884 102.69599995
 155.02870082 114.3648019  139.02040118  90.12549816 115.56649949
 114.58110003 122.67780068 121.63420031 165.29720164  92.84469968
 135.68820146 121.34109955 120.67190079 104.64390027 142.7850025
 122.126299   116.67870055 113.58160047 127.13919752 122.62639945
 125.82119925 121.19780073  86.75869898 132.59420155 144.04890259
  92.75579971 157.40919883 158.78230199 126.37079913 165.47269942
 108.91739954 110.29410085 103.78269833  94.22730119 127.83570278
 107.2395007  159.91589972 121.79810014 131.79639969 130.51210148
 160.4447995   90.06469827 176.08130197 127.59290048 126.89479812
  86.46629954 124.70009943 150.11329752  89.66510001 107.23039969
 108.86999982  84.58789927 135.72889978 155.37120224 139.40520297
  74.03280025 152.05330121 126.24720015 126.71230013 127.5373992
 108.71579967 156.1055998  114.42760105 117.08710137 125.21919944
 154.05590148 121.30819999 156.36469913  92.89310077 125.55240131
 125.62060004  87.76630028  92.07499917 126.20349902 128.66930393
 113.0744006  117.56819739 120.8714002  127.16199777 119.73610107
 135.75870082  93.91229913 119.85930046 113.30520105  94.21589919
 108.93909987  88.08579959 109.0925991   89.60249986  92.45600018
 131.67570328 162.21730064  89.26630012 119.66020072 133.31130204
 123.92100018 128.23260177 101.90889852  88.88889845 131.41930043
 119.81340032 108.29059972 169.0137012  115.16190021  86.49349894
 118.97690071  91.00569952 161.35330119 116.42470063 121.55560023
 160.28799815 120.05529935 112.49649942 108.43509871 126.8765001
  75.8049006  102.99809978 127.99300303 121.82649973  92.63320012
 132.50770111 117.99890135 116.29899934 154.64480266 159.3880011
 109.84929977 154.45369773 119.20100066 160.50280078 118.34710068
 158.26349951 115.22129928 116.82170036 149.13929916 114.75210043
 125.61179833 165.26129962 117.74450028 124.99669921 153.47030388
 153.42430279 132.38530095 114.78040003 121.24250169 125.19900045
  89.79340036 122.96349961 154.79970182 111.76810034 106.80019966
 161.38230169 118.56819984 165.69510036 134.04980061 115.00449918
 152.81519917 168.22869921 114.98049988 114.0551013  158.58979864
```

```
 85.28509906 127.11400073 128.02700048 128.9352998  124.33050033
124.06810067  90.74700045 153.17220111  97.0957999  137.65179995
 88.98999924 107.55880007 115.06320037 112.68910082 124.39779929
 91.44649857 125.37090127 162.47069852 119.86289904 165.02740109
127.02019739 112.36839998 127.62519875  95.05219937  90.99159967
103.26579902 120.92760023  83.46719929 126.4671995  159.51860541
117.20180082 118.21619967 119.94460018 122.70609976 120.09010145
121.37980033 118.32870041 107.19310029 148.37550012 126.43439808
115.71030077  74.05349997 127.81290109 153.96230093 122.59889998
125.64790073  88.81710002 103.48509883 124.33350064 120.31920011
 73.27970083 151.58620013 120.91940041 104.63119984  86.48039776
115.02949902 172.24779822 120.00610016 160.13919758 113.10889956
121.23410034 118.48570108  95.93499986 118.49570028 126.14290027
118.44719944  96.01590098 153.77260199 122.09430021 147.37659908
159.53700243 113.70110025 122.57589923 148.60449732 127.19200058
165.72700057 135.81410034 119.82979984 166.70149883 108.20349955
121.79839862 140.38620094 106.2666991 ]
```

In [20]:

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

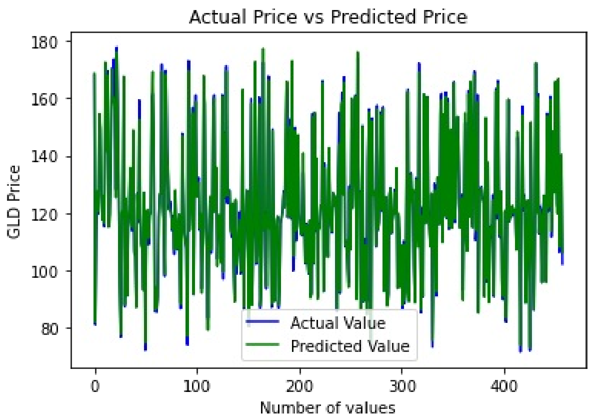
```
R squared error :  0.9893931847227215
```

In [21]:

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

In [22]:

```python
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 [23]:

```python
#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 0.9893931847227215
```

In [24]:

```python
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 0.9984986541563934
```

In [25]:

```python
# 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 : =
[84.81940022]
```

In [ ]:

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