gold-price-prediction

July 2, 2023

Importing the libraries/dependencies

[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
```

Data Collection and Processing

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

[3]:
```
# print first 5 rows of the data frame
gold_data.head()
```

[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
```

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

[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
```

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

[7]: (2290, 6)

[9]:
```
# basic info of the data frame
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
```

[10]:
```
# checking number of missing values
gold_data.isnull().sum()
```

[10]:
```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

[11]:
```
# getting statistical measures of the data frame
gold_data.describe()
```

[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

Correlation: 1. Positive Correlation 2. Negative Correlation

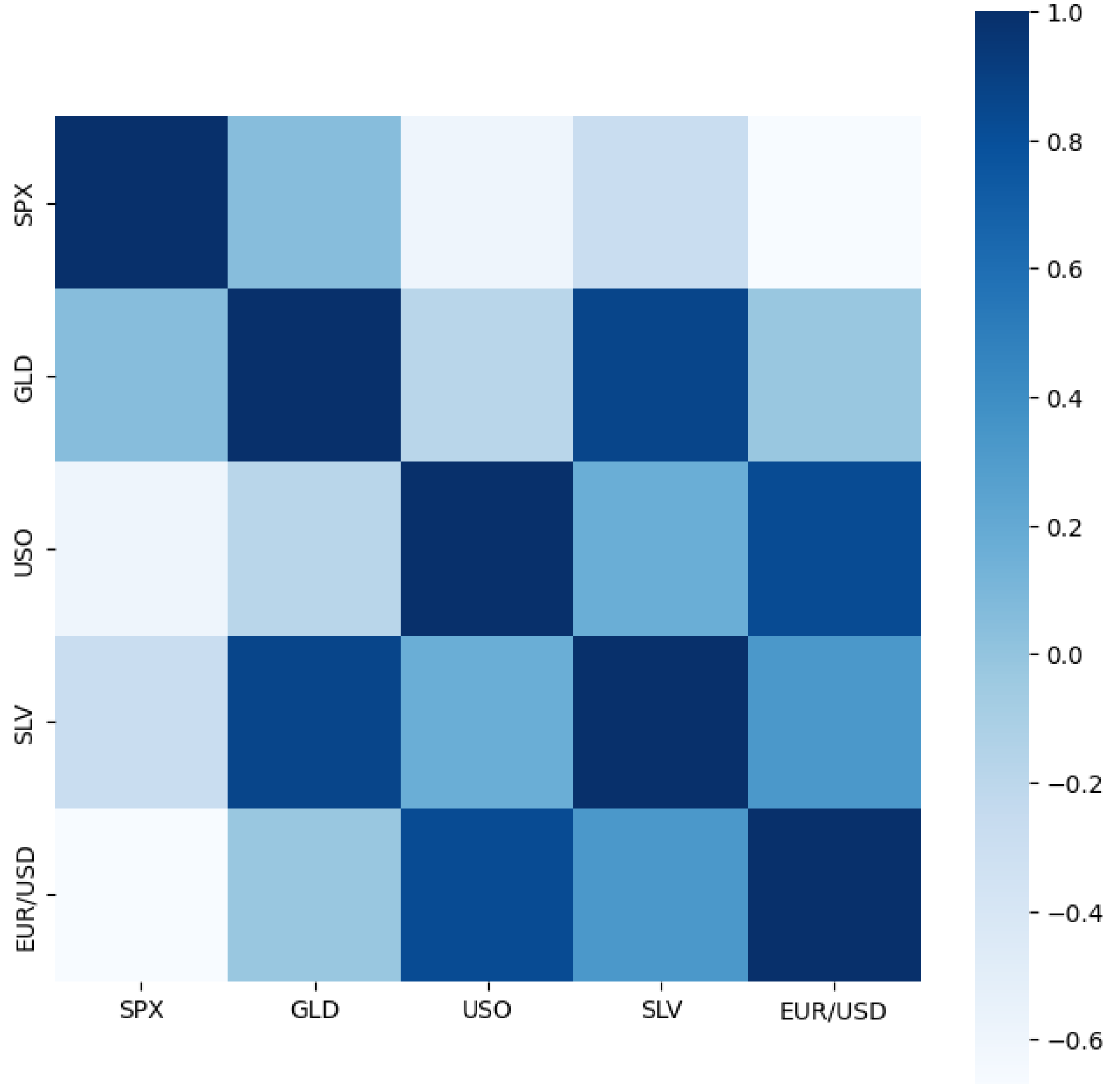
[12]:
```
correlation = gold_data.corr()
```

```
<ipython-input-12-b9d572e5c3ef>:1: FutureWarning: The default value of
```

```
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  correlation = gold_data.corr()
```

[15]:
```
# constructing a heat map to understand to analyse the correlation
plt.figure(figsize = (8,8))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot_kws={'size':
 ↪8}, cmap='Blues')
```

[15]:
```
<Axes: >
```

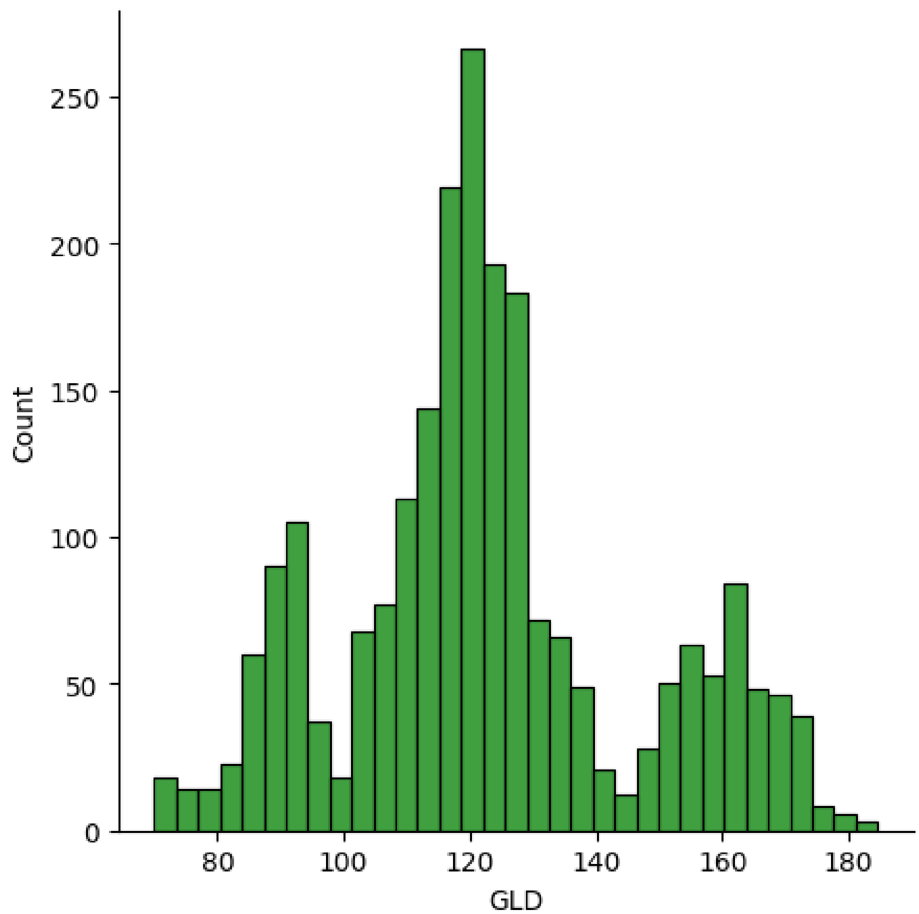


[16]:
```python
# correlation values of GLD
print(correlation['GLD'])
```

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

[17]:
```python
# checking the distribution of the GLD Price
sns.displot(gold_data['GLD'], color='green')
```

[17]: <seaborn.axisgrid.FacetGrid at 0x7fa54bf6b520>



Splitting the Features and our Target

[18]:
```
X = gold_data.drop(['Date', 'GLD'], axis=1)
Y = gold_data['GLD']
```

[19]:
```
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]
```

[20]:
```
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
```

Splitting into Training data and Test data

[21]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=2)
```

Model Training: Random Forest Regressor

[22]:
```
regressor = RandomForestRegressor(n_estimators=100)
```

[23]:
```
# training the model
regressor.fit(X_train, Y_train)
```

[23]:
```
RandomForestRegressor()
```

Model Evaluation

[25]:
```python
# prediction on test data
test_data_prediction = regressor.predict(X_test)
```

[26]:
```python
print(test_data_prediction)
```

```
[168.53079957  81.74409995 116.0889006  127.64820075 120.80620128
 154.79179792 150.38829845 126.25480011 117.32769884 126.14940062
 116.6348011  172.04280119 141.76139908 167.87909877 115.13520019
 117.94860067 138.12770294 170.30420096 159.03540348 160.8239002
 155.1054003  125.09030043 176.86679986 157.13360352 125.14890049
  93.81139997  77.63309983 120.54219991 119.1058994  167.46940062
  88.12860056 125.16159989  91.23050054 117.68700054 121.08299871
 136.24230145 115.43880115 115.04240067 148.29670065 107.19950091
 104.55380252  87.29739794 126.50340085 117.98840012 152.75789858
 119.7339001  108.41569962 107.95649828  93.20460029 127.12289775
  75.31820008 113.63019921 121.27930023 111.17469929 118.87639891
 120.85479913 158.23709937 166.71440109 147.06029643  85.88269883
  94.45670064  86.95729909  90.55640004 118.94650088 126.42270073
 127.26329979 168.36329958 122.29169942 117.17779937  98.3508003
 167.65830055 142.90289863 132.12440245 121.25270243 121.10329944
 119.96670086 114.52080175 118.29830054 107.11230082 127.97990098
 113.73729991 107.6778     116.66620019 119.52259886  89.12280065
  88.21629873 146.34930192 127.19379992 113.44050038 110.08229877
 108.34599908  77.05319918 169.33830152 114.08149911 121.564199
 128.04990217 155.02469819  91.49689927 136.15530067 158.8759034
 126.14970052 125.20670065 130.57390071 114.8817014  119.76989932
  92.06620009 110.19309884 166.51049866 157.66529909 114.22749934
 106.7404013   79.75759997 113.25670041 125.74950051 107.0572996
 119.20200125 156.16970336 159.69339933 120.2381999  134.58280222
 101.73569994 117.69029801 119.31980016 113.02350074 102.77609884
 160.15509783  98.60540056 148.1880991  125.83500107 169.4056992
 125.77269871 127.32549763 127.33440167 113.74179929 112.82250053
 123.81159936 102.17579905  89.16189984 125.00849958 101.54299961
 107.07189915 113.3471004  117.12930072  98.90099961 121.77930058
 163.84819999  87.39919862 106.74050022 117.35820067 127.81080099
 124.04630048  80.86059921 120.35020082 156.51429876  87.95399972
 110.15829959 118.98339911 172.49179876 103.0284993  105.43970013
 122.59000026 157.14869798  87.80149829  93.0898007  112.87050003
 176.98409962 114.19949992 119.37830017  94.71410105 125.7012003
 165.97440046 114.93590096 116.83430121  88.28239867 149.06490074
 120.3196995   89.40079994 112.0929003  117.32330057 118.85920085
  88.01299958  93.95849993 116.97459972 118.51430198 120.24280026
 126.96889805 121.97809969 149.0898001  165.63760157 118.56129965
 120.26460153 151.61890017 118.77899891 172.52719867 106.07069921
 105.01020117 149.28560037 113.94610076 124.87630136 147.94830035
 119.54800111 115.37090047 112.66510049 113.39920204 140.9996009
```

117.70519776	102.91379987	115.79780118	103.83060185	98.59730052
117.26400086	90.74570001	91.76399999	153.63789937	102.73520012
155.12420056	114.32950165	138.87730106	90.09019792	115.47679942
114.38949964	123.02130059	121.75049993	165.35440181	92.80259966
135.38450102	121.40709897	120.71900071	104.63690029	141.6840032
122.09749899	116.63680061	113.500301	127.05599766	122.7288994
125.76509907	121.28850039	86.96519899	132.3714008	144.05350222
92.67679926	156.26910011	158.84050284	126.38099841	165.60899934
108.71059979	109.75930091	103.59629805	94.16010133	127.89010317
107.03070067	159.0493	121.76770055	131.9759999	130.56700085
160.50299923	90.0928984	176.28790262	128.31180038	127.01089819
86.36829924	124.63169981	149.92409723	89.61740026	106.78450008
108.88649993	83.89629906	136.09249902	154.84180266	138.59690381
73.78680003	152.54040045	126.13800021	126.80320035	127.51249877
108.61949938	156.33380038	114.54610085	116.98140171	125.38689927
153.99760127	121.41720027	156.44629867	93.10290069	125.55800095
125.58650033	88.07050082	92.24859913	126.34189912	128.19310336
113.27670051	117.80639752	120.88880009	127.07429812	119.89970046
136.52280158	93.99339956	119.68320032	113.35860109	94.27749955
108.98059959	87.2709994	109.37569904	89.66699973	92.3620001
131.39040253	162.43910091	89.43020018	119.48900072	133.34810194
124.02880024	128.51730201	102.11929882	89.10189863	131.66680041
119.95479994	108.71379989	168.44980089	115.15100034	86.62699913
118.94880062	91.13289936	161.43430061	116.46650047	121.58010028
160.53609814	120.3252991	112.55999926	108.4863989	126.64829978
76.03700064	102.98669963	127.75920284	121.83389892	92.64339994
131.97080047	118.09850087	116.06559983	154.68650292	160.38130114
110.19739957	155.22609831	119.28170091	160.47260059	118.4393
158.18849954	115.04869939	116.52730029	149.16579928	114.71970091
125.47509849	167.12219913	117.69300033	125.27459938	153.18200379
153.51740245	132.11969956	114.77630025	121.33630214	124.77400049
89.83960053	123.52149993	155.14620176	111.86120036	106.72110015
162.09040134	118.84610038	165.74390071	134.13470038	114.64259964
152.99009852	168.66590057	115.27560021	114.13900152	158.36669937
85.53009847	127.16430026	128.05150072	128.95749992	124.0683008
123.81150074	90.50630085	153.15420087	97.08089985	136.75270007
88.79679896	107.65809998	114.91200023	112.77710084	124.13539922
91.39269882	125.29890121	162.35879879	119.95479852	165.08530131
126.960498	112.35340019	127.54149892	94.78069936	90.94969993
103.20039919	120.91630014	82.9845997	126.45549977	159.72230462
117.32170119	118.30939981	120.04089987	122.52439943	120.14670128
121.52939963	117.96570054	107.1089998	148.58150048	126.20089838
115.61740099	73.91010006	127.93160163	153.43140036	123.14769991
125.62290034	88.82149997	103.05089836	124.27040067	120.20510007
73.36890066	151.97960004	121.10820063	104.94899984	86.56949767
114.96899896	172.16019837	119.77330063	160.21389774	113.18149961
121.32550003	118.37820097	96.02389984	118.71129999	125.89460033
118.53789945	95.83530051	154.04610189	122.2287005	147.31099984

```
 159.92830248 113.87410012 122.39599959 148.7041979  127.13340062
 165.66990071 135.19200044 119.90349928 166.73719856 108.42689898
 121.68579845 138.75170062 107.177699  ]
```

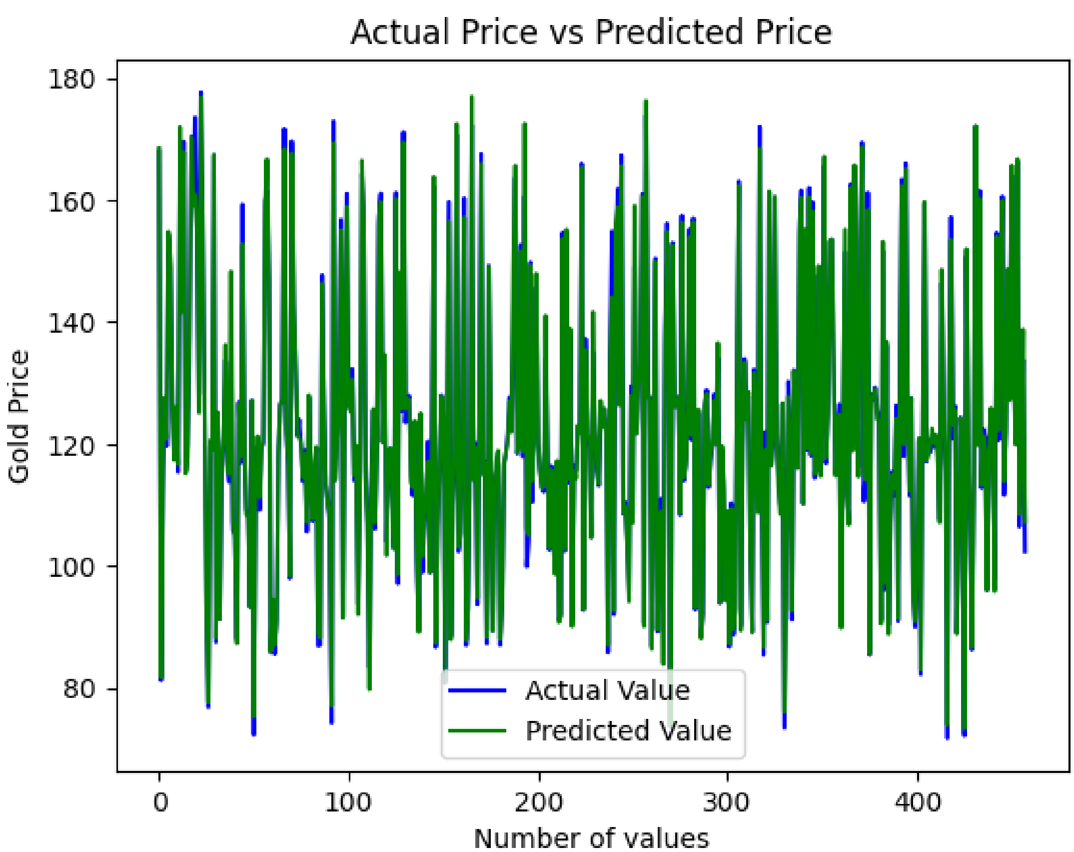
[27]:
```
# R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error : ", error_score)
```

```
R squared error :  0.9898661815204934
```

Comparing Actual values and Preicted values in a Plot

[28]:
```
Y_test = list(Y_test)
```

[29]:
```
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('Gold Price')
plt.legend()
plt.show()
```



```
[ ]:
```