gold-price-prediction

December 8, 2023

Importing the Libraries

[6]:
```
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

[7]:
```
# loading the csv data to a Pandas DataFrame
gold_data = pd.read_csv('/content/gld_price_data.csv')
```

[8]:
```
# print first 5 rows in the dataframe
gold_data.head()
```

[8]:

	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

[9]:
```
# print last 5 rows of the dataframe
gold_data.tail()
```

[9]:

	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

[10]:
```
# number of rows and columns
gold_data.shape
```

[10]: (2290, 6)

[11]:
```
# 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
```

[12]:
```
# checking the number of missing values
gold_data.isnull().sum()
```

[12]:
```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

[13]:
```
# getting the statistical measures of the data
gold_data.describe()
```

[13]:

	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

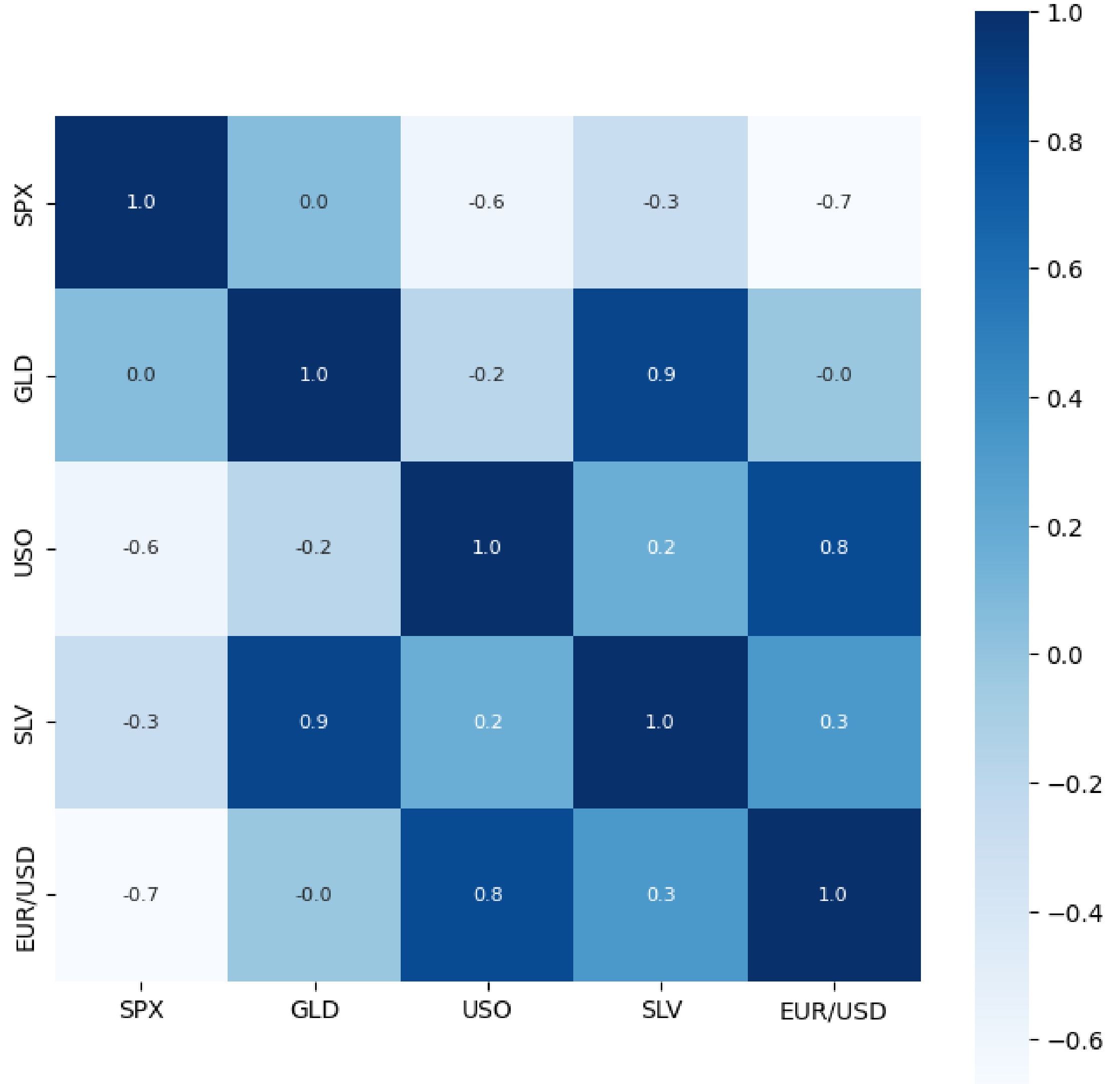
[14]:
```
correlation = gold_data.corr()
```

```
<ipython-input-14-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 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')
```

[15]: <Axes: >



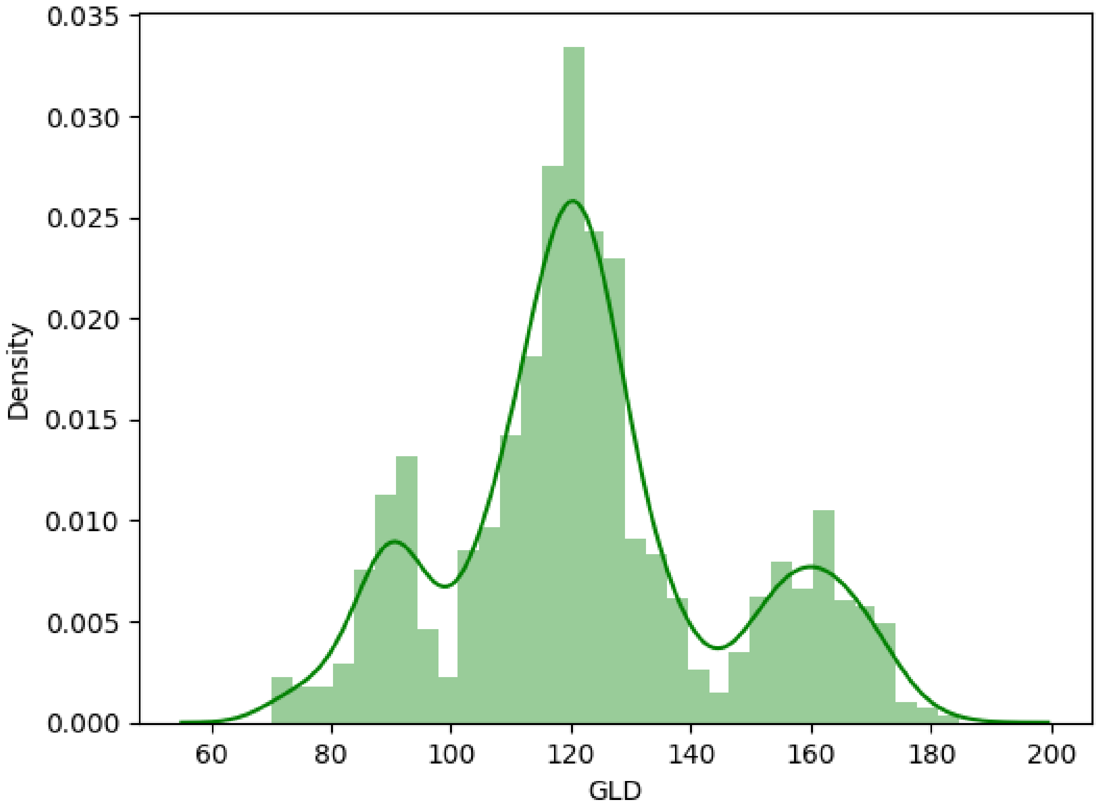
[16]:
```python
# correlation values of GLD
print(correlation['GLD'])
```

```
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.distplot(gold_data['GLD'],color='green')
```

```
<ipython-input-17-b94eac2e88dd>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(gold_data['GLD'],color='green')
```

[17]:
```
<Axes: xlabel='GLD', ylabel='Density'>
```



Splitting the Features and 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

[24]:

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

[25]:

```
print(test_data_prediction)
```

```
[168.60029937  81.67900006 116.10599999 127.72210078 120.6274014
 154.67229824 150.66539895 126.17210035 117.46689868 125.98310104
 116.68880114 172.0841008  141.2393982  167.85569854 115.23129992
 117.56680032 138.95790223 170.20840079 159.54900317 160.674099
 155.06579991 124.79220058 176.15829978 157.44560397 125.22660062
  93.66419981  77.41570019 120.66759978 119.08809939 167.47509964
  88.03820038 125.40200023  90.78000038 117.92210003 121.0640993
 136.97780148 115.38610151 115.23660048 147.97109955 107.2900007
 103.97160253  87.03789775 126.50490064 118.09340024 153.56009873
 119.74389976 108.53310021 108.05159797  93.2552005  126.99949797
  75.08360036 113.70099953 121.27909986 111.35979901 118.77439879
 120.3314998  158.93909995 168.56000151 147.19569748  85.70279872
  94.21680043  86.87029839  90.54589997 118.96560089 126.47970043
```

127.53500012	170.18150075	122.25629913	117.50479908	98.82019983
168.26300068	143.23509818	131.86430299	121.13160251	121.17189912
119.79400055	114.52470154	118.05730066	107.0179007	127.82950061
114.02069956	107.66430005	116.76050097	119.79329849	89.09920083
88.30919868	146.55850292	127.18850006	113.22790006	110.04599846
108.15149914	77.24719904	169.89200207	114.00799921	121.54159924
128.00550205	154.92349852	91.67399918	135.94520132	158.77230296
125.29310067	125.20960035	130.7525017	114.81730097	119.72279933
92.11769987	110.34129906	168.0725997	157.20769866	114.21029967
106.5930011	79.83849958	113.40180016	125.8192006	107.42389918
119.10150067	156.06810345	159.59169946	120.11729972	133.97210305
101.51709967	117.72559795	119.23880013	113.02050099	102.78389942
160.13609758	98.81560036	147.49279939	125.44780095	169.89259879
125.66149922	127.43399724	127.37690168	113.67069939	113.19050081
123.52219884	102.0803993	89.47119996	124.3778996	101.30249941
107.11389918	113.38880085	117.40890066	99.40209932	121.67250046
163.12419985	87.40659861	106.8518999	117.1782007	127.65880179
124.27860076	80.56649937	120.3042005	158.72679826	88.0156997
110.18019977	118.7899991	172.19239866	102.9682989	105.12890053
122.41550039	158.98739806	87.64539839	93.19390044	112.63510029
176.9434003	114.44459993	119.32350017	94.7728009	125.90710017
166.30700074	114.78140044	116.8119011	88.2845985	149.15950212
120.2602993	89.47249999	112.61169997	117.16550071	118.73080119
88.21729933	94.14629994	117.09759958	118.58620206	120.19590058
126.93509793	121.90149998	148.8219009	164.86870069	118.42479945
120.30290148	150.50520083	118.2745996	172.91279899	105.7202994
104.87570124	149.9899016	113.83690063	124.86930118	147.16260044
119.70510122	115.27180057	112.79420016	113.53300186	141.06650105
117.98409758	102.88340039	115.72140094	103.50770177	98.96360028
117.16300069	90.81920007	91.58850058	153.27119923	102.69769974
154.48920098	114.34750124	138.80510151	90.07959792	115.42849974
114.44869977	123.11540046	121.89040034	165.20230112	92.8854997
134.8385013	121.31359931	120.75660076	104.70910035	142.23950307
121.67969904	116.66470048	113.40930102	127.08509787	122.54029924
125.74059952	121.21020043	86.7710991	132.98080131	144.03160217
92.7495994	161.0218	159.15990153	126.54159865	164.94109951
108.98889956	109.61390134	103.57399811	94.30180057	127.84860267
107.47570046	163.19160029	121.46740031	131.9774	130.83650178
159.2865	90.17889839	175.3645025	127.48340062	126.95979805
86.27839925	124.50249943	150.15529733	89.66200032	107.03659975
109.10199979	84.76839901	136.37310002	155.36380236	139.80820286
73.90050026	152.46300167	126.23860064	126.79150016	127.54949909
108.43939901	156.40939999	114.47340125	116.91890159	125.11169985
153.85880139	121.4225	156.44379912	93.03340055	125.52490153
125.64400038	87.84140053	92.34959928	126.26489904	128.30000343
113.28020049	117.61949747	120.91220025	127.15869794	119.641601
137.17630103	93.97319907	119.77970061	113.37320095	94.30529938
108.84189979	87.16819933	109.10279932	89.65069968	92.26080008

```
 131.87070282 162.36270091  89.16810023 119.71300052 133.20580169
 123.93230007 128.29860187 101.78229841  88.99359871 131.5715006
 120.14940006 108.38439972 168.00120132 115.22610053  86.52559879
 119.01930062  91.1225994  161.66920094 116.46360077 121.57350023
 160.43969845 120.06059932 112.4289998  108.52189865 126.71260026
  75.98720029 102.98769991 127.73970241 121.52439968  92.57510025
 131.9447007  118.29440048 115.5798998  154.43630303 159.96340057
 110.17429984 155.30789771 119.14570078 160.67630113 118.56110004
 156.70589958 115.16419931 116.61170015 149.8583989  114.7649004
 125.76129879 165.75849935 117.64540001 125.03139941 153.35420352
 153.43770261 132.3638001  114.58580017 121.24430186 124.56260048
  89.75210035 123.26270016 154.72190122 111.90590036 106.8833998
 162.1504015  118.50489967 165.77189965 134.05940127 114.73519983
 152.93089842 168.85380049 114.61099979 114.00840143 158.43589951
  85.19409869 127.22750032 127.86220101 128.70869952 124.2936008
 123.83420053  90.6086003  153.48490049  96.96199971 137.43279972
  89.03019914 107.64470028 114.95490023 112.93940074 124.14339922
  91.41389865 125.33450143 162.34459896 119.88059926 164.81080073
 126.95499798 112.35240009 127.46529946  94.97489886  90.93109969
 103.19189938 120.83880039  83.3974993  126.37049944 160.27180547
 117.29650068 118.20099984 119.85599989 122.57609974 120.17610099
 121.33850026 118.06640066 107.04059973 148.47370005 126.25119877
 115.71520052  73.86689979 127.87130142 153.86420141 122.02579979
 125.58980054  88.92419988 103.1398987  125.14100082 120.31720025
  73.45540074 151.27190076 121.09670019 104.53719988  86.51709756
 115.14129893 172.25489856 120.09530034 159.45669787 113.13749996
 121.19140008 118.73210055  96.05479989 118.84939984 125.76679995
 118.56579968  96.02590049 153.69620168 122.11400026 147.71819972
 159.5607021  114.19410041 122.51139928 149.74689823 127.18820037
 165.83079991 135.53350084 120.11089944 167.58859901 108.58119947
 121.82259853 138.40890097 106.82289893]
```

[26]:

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

```
R squared error :  0.9889874057219952
```

Compare the Actual Values and Predicted Values in a Plot

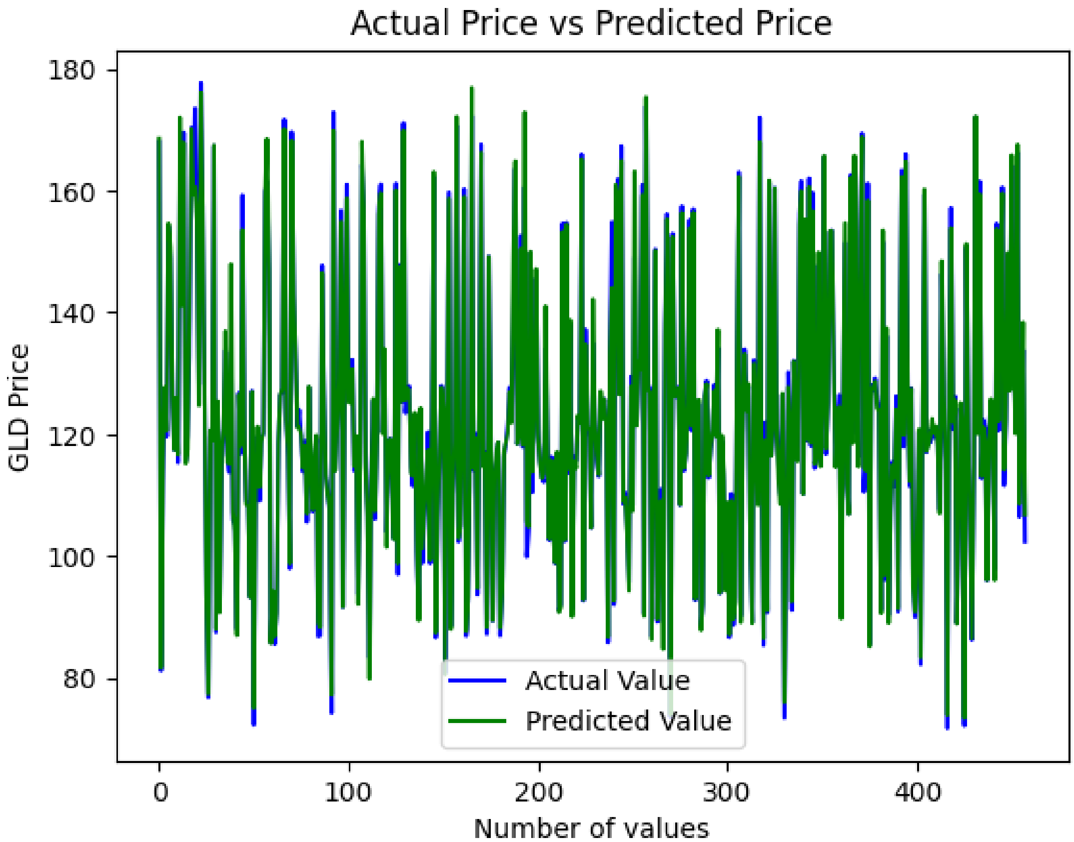
[27]:

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

[28]:

```
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()
```


Actual Price vs Predicted Price
GLD Price
Number of values
Actual Value
Predicted Value
180
160
140
120
100
80
0
100
200
300
400