Importing 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
```

Data collection and processing

```
#loading csv to pandas data frame
gold_data = pd.read_csv('/content/gld_price_data.csv')

#print first five rows
gold_data.head()
```

	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

```
#print last five data frame
gold_data.tail()
```

	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

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

```
(2290, 6)
```

```python
#getting some basic info about 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
```

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

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

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

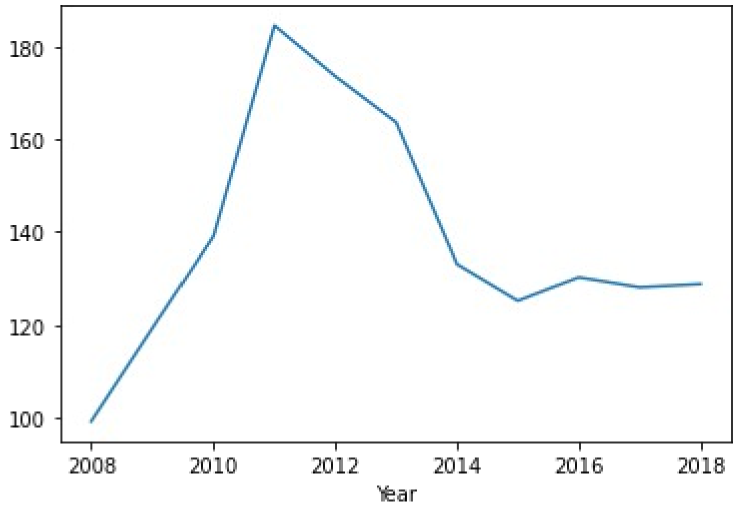
	SPX	GLD	USO	SLV	EUR/USD
count	2290.000000	2290.000000	2290.000000	2290.000000	2290.000000
mean	1654.315776	122.732875	31.842221	20.084997	1.283653
std	519.111540	23.283346	19.523517	7.092566	0.131547
min	676.530029	70.000000	7.960000	8.850000	1.039047
25%	1239.874969	109.725000	14.380000	15.570000	1.171313
50%	1551.434998	120.580002	33.869999	17.268500	1.303296
75%	2073.010070	132.840004	37.827501	22.882499	1.369971
max	2872.870117	184.589996	117.480003	47.259998	1.598798

```
#Converting the date column to the proper format
gold_data['Date'] = pd.to_datetime(gold_data['Date'])
gold_data['Date'] = gold_data['Date'].apply(lambda x:x.date())


#Creating a year column just for some data visualization
gold_data['Year'] =gold_data['Date'].apply(lambda x: x.year)


gold_data['Year'].value_counts()
```

```
2009    224
2014    224
2015    223
2011    222
2010    222
2013    221
2016    221
2012    219
2017    218
2008    209
2018     87
Name: Year, dtype: int64
```

```
#We can see how the maximum gold value changed from 2008 to 2018
gold_data.groupby('Year').max()['GLD'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc962dbbc10>
```



correleation:

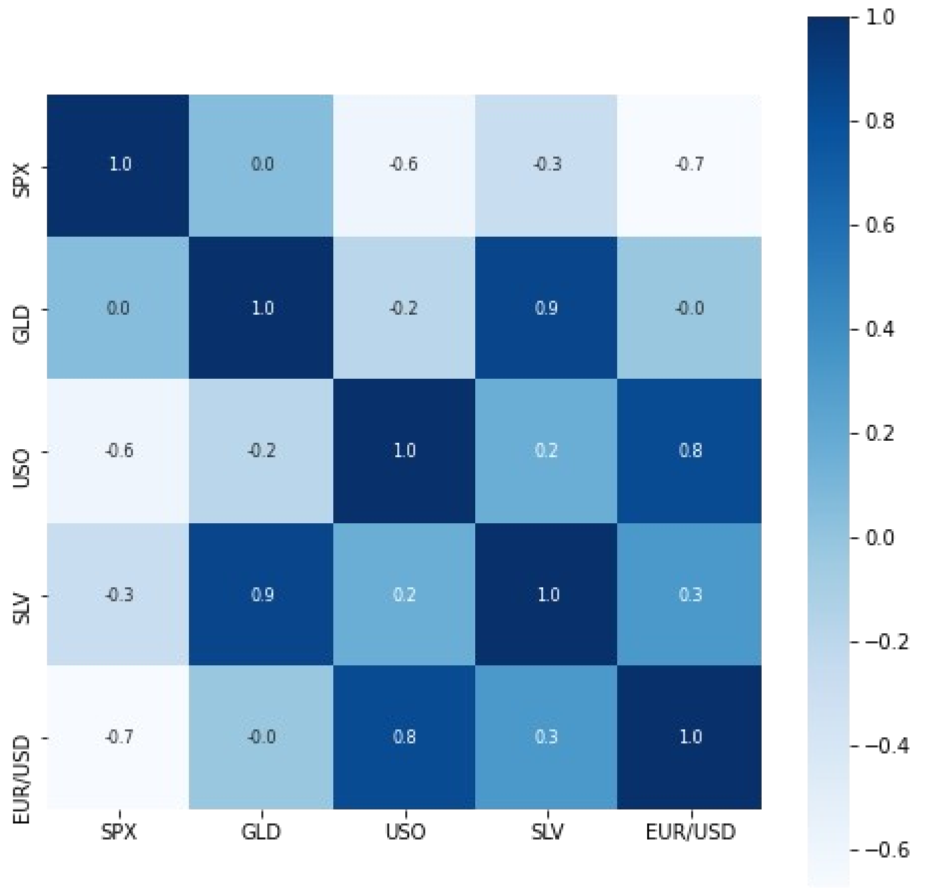
1.positive correlation

2.Negative correlation

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

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

<matplotlib.axes._subplots.AxesSubplot at 0x7fc9717b6ed0>

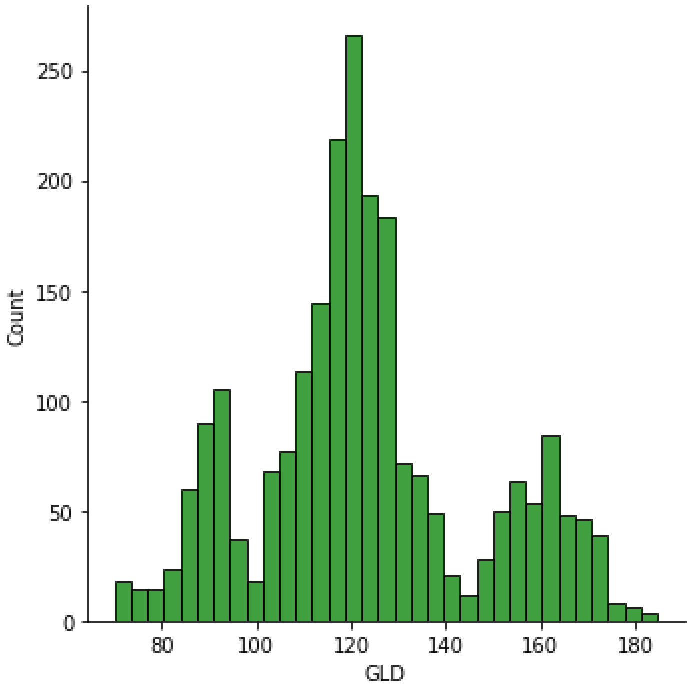



```
#correlation values of GLD
print(correlation['GLD'])
```

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

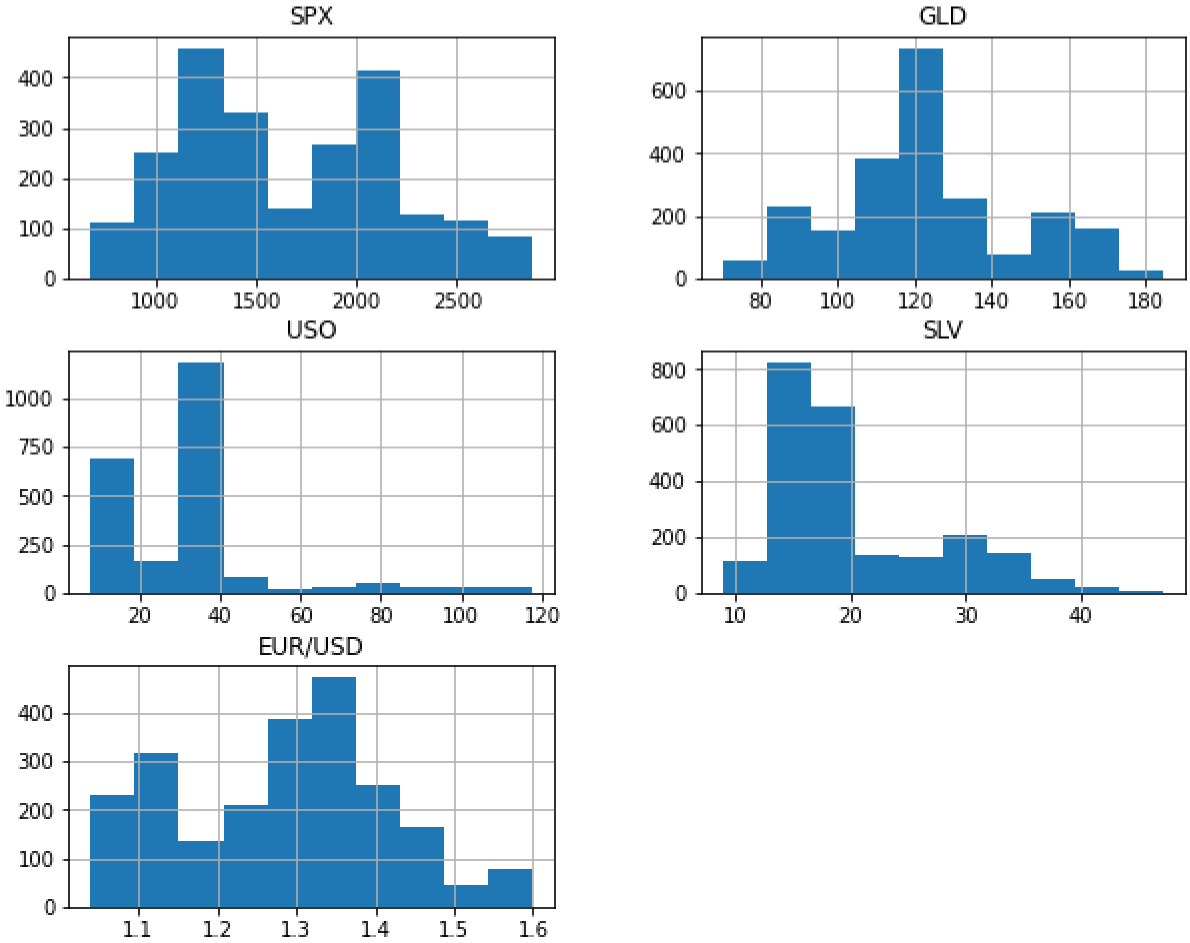
```
#checking the distribution of the GLD price
sns.displot(gold_data['GLD'],color='green')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fc96342c750>
```




```
gold_data.hist(figsize=(10,8))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fc962c6fd10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc962c29350>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc962bde9d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc962c09b90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc962b57710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc962b118d0>]],
      dtype=object)
```


SPX
0 100 200 300 400
1000 1500 2000 2500
GLD
0 200 400 600
80 100 120 140 160 180
USO
0 250 500 750 1000
20 40 60 80 100 120
SLV
0 200 400 600 800
10 20 30 40
EUR/USD
0 100 200 300 400
1.1 1.2 1.3 1.4 1.5 1.6

splitting the features and target (gold and date)

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

```
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]
```

```
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

```
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

```
regressor = RandomForestRegressor(n_estimators=100)


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

RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Model evaluation

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

print(test_data_prediction)

[168.68449967  82.02389996 116.17810003 127.66170085 120.82890131
 154.74819774 150.15929927 126.16140027 117.41029883 125.87800101
 116.76950108 171.5890007  141.95409882 168.19929899 115.20049982
 117.73670044 136.97140352 170.11210059 159.4500027  157.45339942
 154.89170025 125.39660044 176.23649954 156.45690328 125.15760044
  93.9111996   77.66950018 120.59830016 118.9816988  167.45359896
  88.27430047 125.25120009  91.15660076 117.75820031 121.0192995
 136.67070118 115.47530089 114.43630084 147.97420017 107.13990098
 104.36130255  87.19379803 126.56630038 117.81100012 153.80609928
 119.77190019 108.33689983 107.87419829  93.25400027 126.90859838
  74.60200045 113.55529902 121.35750032 111.32749967 118.78259883
 120.56659931 160.07999953 167.69060091 146.90879667  86.03669908
  94.46580049  86.77199867  90.62890026 118.96330092 126.43820105
 127.61669982 169.79350018 122.20339955 117.3561993   98.27080023
 168.61100102 142.87829841 131.9266033  121.18900208 120.70209943
 119.81500057 114.59180187 118.62500059 107.20710086 127.95230093
 113.76389977 107.06070015 116.98160058 119.65559866  89.1104009
  88.34319904 147.17070203 127.16500038 113.74770036 110.25619834
 108.41749885  77.43649904 169.87380201 114.17069943 121.6948989
 127.99960115 154.8721983   91.7199993  136.3751013  159.18670348
 124.47940072 125.16610027 130.15300193 114.99240132 119.79239996
  92.09749982 110.62179901 167.55419919 157.94009905 114.42479969
 106.75170118  79.61849957 113.3078004  125.80910064 107.32459932
 119.68110081 156.13920392 159.49079918 120.19730016 135.02050271
 101.44129995 117.56709819 119.24910034 113.06220088 102.76549919
 160.18629823  99.02350025 147.89979897 125.65220097 169.70719941
 125.76319876 127.31299759 127.43990139 113.85509933 112.60550072
 123.643299   102.09989874  89.42039993 124.5853997  101.61509923
 107.21979932 113.33880089 117.20840056  99.02449949 121.70770046
```

```
163.8007989   87.47029895 106.52219966 116.93770106 127.7486011
124.14610049  80.77819901 120.68200067 158.87759804  87.9658998
110.17469988 118.70489925 172.5307984  103.06559906 105.33810022
122.80670048 158.88769781  87.63849842  93.10080041 112.7066005
177.18359975 114.05389975 119.23850022  94.60920094 125.4568
166.2639006  114.79080043 116.48660139  88.30239884 149.55870083
120.31389971  89.32329984 112.22320006 117.52550016 118.75680106
 88.04179935  94.07690013 116.99959999 118.74670211 120.34000094
126.72729835 121.83909961 147.83840026 164.82509981 118.55569972
120.32310175 151.25430102 118.56999921 172.58109838 105.30009928
105.00220059 150.11530103 113.54730116 124.77520086 147.95020051
119.75630129 115.49720053 112.70910013 113.45340173 141.96240134
117.7650977  102.95310047 115.81740104 103.94700204  98.56550042
117.43340099  90.75380014  91.55400059 153.69229955 102.77949977
154.72820079 114.28770122 138.83100151  90.13569812 115.60169971
114.58449933 123.03300034 121.77270013 165.21610137  92.83459979
135.127201   121.43629898 120.90700049 104.77760014 140.45060319
121.83209901 116.64780041 113.58400063 127.01209765 122.82469942
125.80669935 121.26220039  86.76309869 132.51220131 145.9791022
 92.6725994  159.18219961 159.50600246 126.35929908 164.97949922
108.86529956 109.54000102 103.90329871  94.38880039 127.80910279
107.09960039 161.64299963 121.75510044 132.20719989 130.2038021
160.27409968  90.18799843 176.20870169 127.47270027 126.67839891
 86.36019928 124.4944992  150.60619729  89.55320025 106.85389964
109.02779995  84.41309903 136.06449885 155.10220289 138.41960359
 74.34940021 152.71120143 126.10559995 126.66710051 127.4386992
108.7787     156.05550057 114.64670114 116.89190147 125.13259945
154.00800162 121.35999979 156.38809867  92.8839007  125.44870155
125.80250019  87.78760045  92.11269925 126.14029967 128.2608033
113.15450068 117.50339717 120.95770026 127.0249978  119.46350088
136.20270069  93.92699962 119.8627002  113.11250096  94.20219917
108.83729967  87.43079936 109.24799896  89.52729982  92.46910043
131.55180286 162.42430092  89.36010011 119.67450089 133.23010212
123.81200015 128.55180191 101.9247984   88.81919868 131.9426009
119.48450022 108.49739972 168.00390169 115.21480033  86.57979897
118.90470044  91.12759968 161.59240025 116.72700086 121.46429973
160.21869847 119.99419949 112.97789902 108.4471985  126.73079978
 76.23460007 103.07889972 127.78200237 121.71699933  92.54040025
131.85820007 118.06260094 115.64499979 154.67770305 158.91480072
110.04509954 154.94009856 119.12830075 160.19540061 118.36670062
158.68899956 115.22079948 116.65840023 150.13219882 114.84530084
125.62619871 165.34479934 117.60930004 125.42099959 153.24060366
153.37140269 132.123      114.78350035 121.19260222 125.06910062
 89.71790048 122.99570015 154.97880206 111.70430046 106.61759974
161.75370186 118.37359986 165.54769972 134.27870107 114.62539994
153.08679941 168.77019974 114.40979979 114.0600011  158.95179909
 85.4830985  127.07350089 128.04490081 128.87210024 124.31320059
123.94780071  90.42960063 153.45530039  96.97059981 137.74640001
 89.05689923 106.96510025 115.15290019 112.29600091 124.22189902
 91.4378984  125.38750126 162.53169903 120.08999885 165.07720062
126.80429806 112.14210015 127.63819941  95.00879864  91.14109985
103.06309911 120.92390022  82.8501997  126.27910009 160.38690456
117.37540098 118.45589982 120.04160002 122.72089967 120.18540123
121.45030023 118.2651004  107.15920003 148.5619001  126.32169845
115.69780059  73.8379     127.78490055 155.03130133 121.71380005
125.68200045  88.84430016 103.06019832 124.3260003  120.31860039
 73.41430085 151.82830043 121.2630005  104.79519944  86.48039801
115.26769886 172.18109784 120.01790014 160.48839723 113.30049981
120.93490005 118.51890072  95.93889994 118.80289983 125.63480035
118.57079964  95.88830052 153.91560187 121.9668999  147.99440017
159.4456023  113.90340024 122.53789923 149.09579795 127.30950032
165.86100028 135.48709947 120.12099939 167.7284989  108.27039923
121.72449869 139.0859003  107.7708988 ]
```

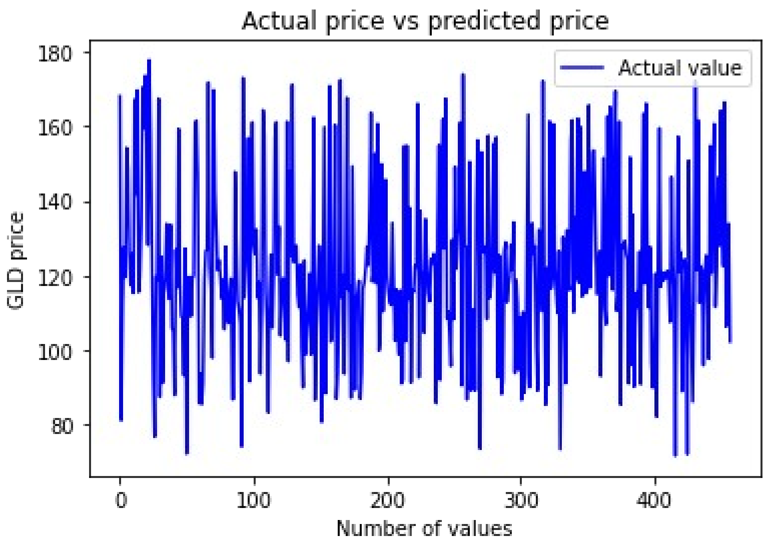
```
#R square error
error_score=metrics.r2_score(Y_test, test_data_prediction)
print("R square error:",error_score)
```

```
R square error: 0.9895186991521377
```

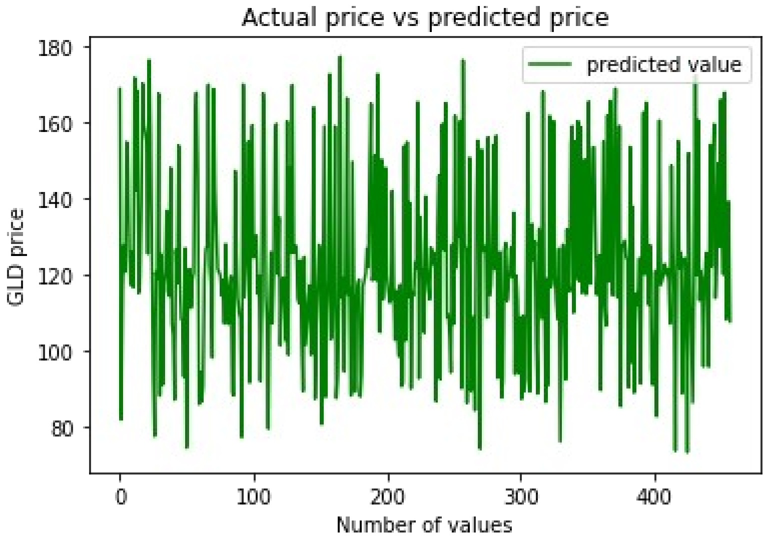
compare the actual value and predicted values in a plot

```
#Y_test to list convertion so we dont get errors
Y_test=list(Y_test)
```

```
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()
```



```
#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()
```




```
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()
```