**Data Science Project** on

# Gold Price Prediction using Machine Learning

# BACHELOR OF TECHNOLOGY DEGREE

**Session 2022-23**

**in**

## CSE-Data Science

**By:**

**DEVANSH**

**2100321540056**

## Under the guidance of:

**Ms. Dimple Tiwari Assistant Professor**

**DEPARTMENT OF CSE-DS**

**ABES ENGINEERING COLLEGE, GHAZIABAD**

Estd. 2000

NAAC Accredited

Grade A

**AFFILIATED TO**

**DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, U.P., LUCKNOW**

**(Formerly UPTU)**

**ABES Engineering College, Ghaziabad**

**Data Science Project Report CSE-DS**

| S-No | Title | Total-Marks | Marks-Obtained | Sign |
|---|---|---|---|---|
| 1. | Preparing Data | **5** | | |
| 2. | Comparing Minimum 3 Models | **20** | | |
| 3. | Optimizing the Model | **10** | | |
| 4. | Confusion Matrix | **10** | | |
| 5. | Visualization of Result | **10** | | |
| 6. | Prediction on New Data | **10** | | |
| 7. | Certification | **10** | | |
| 8. | Report | **10** | | |
| 9. | Presentation | **15** | | |
| | **Total** | **100** | | |

# IMPORTING THE LIBRARIES ¶

In [2]:

```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import confusion_matrix from
sklearn.metrics import accuracy_score
```

# COLLECTING THE DATA:

Data collection is a process of collecting information from all the relevant sources to find answers to the research problem, test the hypothesis (if you are following deductive approach) and evaluate the outcomes.

In [3]:

```
gold_data = pd.read_csv("gld_price.csv")
```

In [4]:

```
gold_data.head()
```

Out[4]:

| | 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 |

[5]:

```
gold_data.tail()
```

Out[5]:

| | 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
In [6]:

```
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 [7]:
```

```
gold_data.shape
```

Out[7]:

```
(2290, 6)
```

In [8]:

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

Out[8]:

```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64  [9]:
```

```
gold_data.describe()
```

Out[9]:

| | 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 [10]:

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

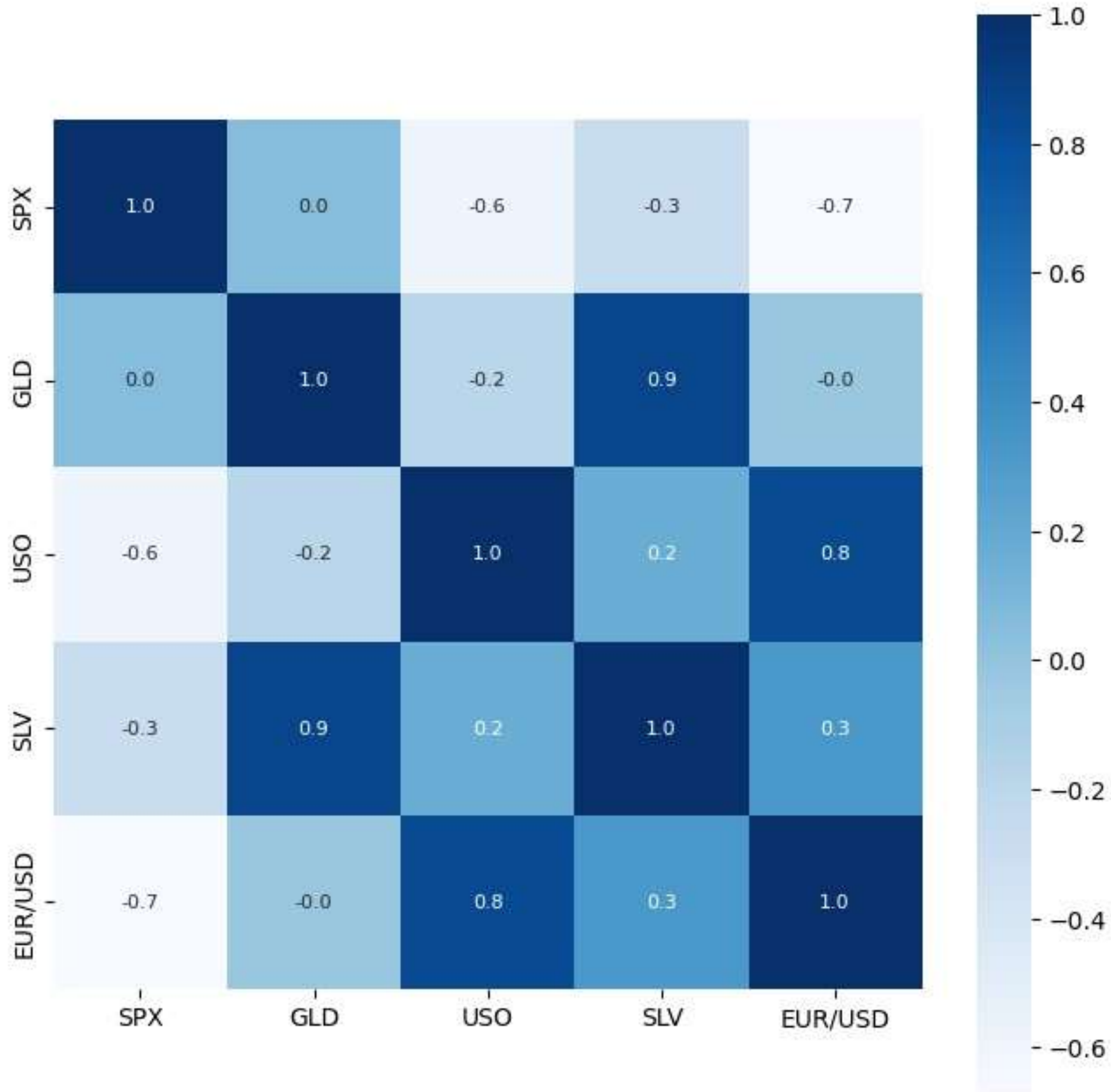
In

[11]:

```
plt.figure(figsize = (8,8))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f',annot=True, annot_kws={'size'
```

Out[11]:

<AxesSubplot:>




In [12]:

```
print(correlation['GLD'])
```

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

[13]:

```
print(correlation['SPX'])
```

```
SPX        1.000000
GLD        0.049345
```

```
In
USO       -0.591573
SLV       -0.274055
EUR/USD   -0.672017 Name:

SPX, dtype: float64 In
```

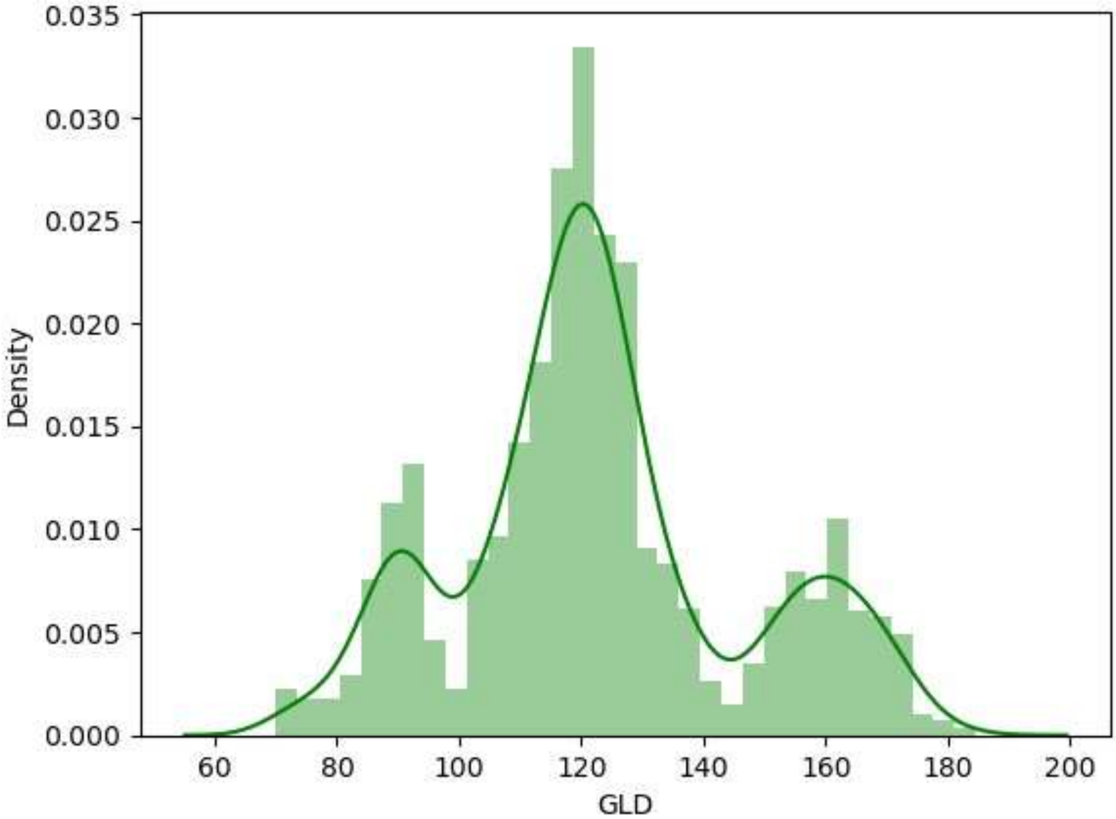
[61]:

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

```
C:\Users\LAKSHYA\anaconda3\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a fig
ure-level function with similar flexibility) or `histplot` (an axes-level
function for histograms).  warnings.warn(msg, FutureWarning)
```

Out[61]:

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




# SEPARATING FEATURES AND LABELS:

One common technique is to split the data into two groups typically referred to as the training and testing sets. The training set is used to develop models and feature sets; they are the substrate for estimating parameters, comparing models, and all of the other activities required to reach a final model. The test set is used only at the conclusion of these activities for estimating a final, unbiased assessment of the model's performance.

[15]:

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

In

```

```

In [16]:

```
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 [17]:
```

```
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
```

# DATA STANDARDIZATION:

Why to standardize before fitting a ML model? Well, the idea is simple. Variables that are measured at different scales do not contribute equally to the model fitting & model learned function and might end up creating a bias. Thus, to deal with this potential problem feature-wise standardized (μ=0, σ=1) is usually used prior to model fitting.

In [18]:

```
scaler=StandardScaler()
scaler.fit(X)
```

Out[18]:

```
StandardScaler()
```

[19]:

```
standardized_data=scaler.transform(X)
```

In [20]:

```
print(standardized_data)
```

```
[[-0.39914541  2.38880956 -0.6917197   1.42975293]
```

In
```
[-0.39914541  2.38368652 -0.67691224  1.45103511] [-
0.46760428  2.32938091 -0.69355301  1.45864621] ...
[ 2.05926403 -0.89307824 -0.61274655 -0.69876145]
[ 2.0728668  -0.89461519 -0.63813078 -0.68838269]
[ 2.06448555 -0.89329341 -0.65305106 -0.77266741]]
```

In [21]:

```
X=standardized_data
X
```

Out[21]:

```
array([[-0.39914541,  2.38880956, -0.6917197 ,  1.42975293],       [-
 0.39914541,  2.38368652, -0.67691224,  1.45103511],
       [-0.46760428,  2.32938091, -0.69355301,  1.45864621],
  ...,
       [ 2.05926403, -0.89307824, -0.61274655, -0.69876145],
       [ 2.0728668 , -0.89461519, -0.63813078, -0.68838269],
       [ 2.06448555, -0.89329341, -0.65305106, -0.77266741]])
```

# TRAIN_TEST_SPLIT:

In [22]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state=2
```

In [23]:

```
print(X.shape)
```

(2290, 4)

In [24]:

```
X_train.shape
```

Out[24]:

(1832, 4)

In [25]:

```
X_test.shape
```

Out[25]:

(458, 4)

[26]:

```
print(y.shape)
```

(2290,)

In

```

```

# TRAINING THE MODEL: RANDOM FOREST REGRESSOR

In [27]:

```
regressor = RandomForestRegressor(n_estimators=100)
```

In [28]:

```
regressor.fit(X_train,y_train)
```

Out[28]:

```
RandomForestRegressor()
```

In [29]:

```
test_data_prediction = regressor.predict(X_test)
```

In [30]:

```
print(test_data_prediction)
```

```
[168.59539932  81.89589979 115.75970059 127.70520077 120.62690141
 154.83319762 150.0523991  126.18610022 117.56999851 125.93860098
 116.77350092 171.9488005  141.92409872 167.96979884 115.15260056
 117.38110042 138.52960342 169.77090034 159.96230298 159.59309974
 155.10560035 124.88160029 175.70799935 156.54640354 125.25910016
 93.8232997   77.15960047 120.67970018 119.13929945 167.44660045
 88.09570052 125.12820021  91.09670097 117.54560056 121.08519839
 136.05360102 115.30410138 114.90740093 146.8016002  107.59110105
 104.29520218  87.3768981  126.45110073 118.00769962 152.9857996
 119.58389978 108.47279969 107.99219801  93.36070075 127.24149758
 75.09870058 113.75149935 121.51770006 111.27319902 118.84879897
 120.62929933 158.42010063 168.01260087 147.03009654  85.74739838
 94.29650048  86.74659908  90.60080022 118.8681009  126.51100087
 127.63690023 170.32120033 122.27579948 117.44409863  98.6618002
 168.05720054 143.21619865 132.21170183 121.09980203 121.16499928
 119.71020043 114.56390152 118.1721008  107.23460101 127.86540037
 113.77509997 107.47679973 116.57760055 119.49489858  88.95570034
 88.14949859 146.60830245 127.39400009 113.34300037 109.63609812
 108.11349881  77.86369905 169.52700162 114.03399925 121.58869924
 127.77870174 155.03119754  91.78859934 135.86500147 158.84520393
 125.36950055 125.59000072 130.58300186 114.88120128 120.02780017
 92.14589986 110.09589896 168.33709983 157.85319932 114.13619941
 106.51020157  79.11479968 113.34020027 125.92300097 107.24559972
 119.24860089 156.28860334 159.39439933 120.16519996 135.0193024
 101.09580002 117.48819775 119.3191003  112.89460049 102.7813992
 160.29529754  98.87510003 148.10599951 125.56100121
 169.34889884
 125.4386993  127.23939825 127.36330163 113.77769921 113.19990093
 123.45959887 102.09249888  89.30599979 124.53109945 101.50729926
 107.10619887 113.6945003  117.30620071  99.53919946 121.8522
 163.5931986   87.37559822 106.81230004 117.24740064 127.70230133
 124.14220061  80.76989901 120.29410072 156.6123981   87.9151995
 110.62519917 119.08119903 172.25109827 103.045999   105.79440069
 122.49710047 156.92859771  87.84399803  93.3635001  112.99580039
 176.75529973 114.05339991 119.21920001  94.57630087 125.53559989
 166.21100082 114.95440071 116.64560134  88.2418986  148.89660108
 120.34609949  89.41379964 112.45669995 117.27830034 118.81870123
 88.00749969  94.1577999  116.75289961 118.69590161 120.13540055
 126.52389874 121.82159997 150.3787999  165.45300074 118.5224996
 120.3677012  150.80100056 118.48089906 173.03779789 105.5849994
 104.91210123 149.13470091 113.74010065 124.73080113 147.26960088
 119.5580012  115.35320043 112.77429992 113.45090194 142.01940129
 117.92819765 102.9274003  115.75880119 103.8209017   98.85710032
 117.39950025  90.71290035  91.54360081 153.48159943 102.73109967
 154.74170096 114.37810137 138.05040096  90.14779774 115.4567996
 114.18609989 122.78240022 121.68940028 165.50400131  92.92709949
 135.4540007  121.39949925 120.96250097 104.69150014 141.61100345
 122.01259902 116.71870035 113.63250084 127.0605975  122.64209956
 125.90759934 121.25650012  86.97649918 132.51890138 147.39510145
 92.59279985 157.2426995  159.06860282 126.43389925 165.0949997
 108.89329972 109.6902009  103.82979819  94.25350069 127.74350272
 107.13130039 161.0859002  121.83719991 131.85499985 130.62110153
 160.56390084  90.06419807 175.35300163 127.40920031 126.52629913
 86.48779942 124.49239938 150.07469752  89.62120006 106.98539992
 109.13619994  84.14559914 136.75760036 155.11860228 137.79640409
 74.26390034 152.81800017 126.0515997  126.80279995 127.57329878
 108.45689949 156.44820002 114.56780088 116.95270144 125.01279982
```

```
 154.21060166 121.2985999  156.39939868  92.96240062 125.45230104
 125.1298001   87.77210036  91.99039899 125.97360046 128.2896035
 113.09540011 117.75389765 120.97040016 127.11109742
 119.60900111
 135.49030066  93.9803992  119.95229991 113.18980097  94.30129921
 108.99379902  87.05839907 108.94069918  89.56149967  92.51740018
 131.74170302 162.10239999  89.36230042 119.50280097
 133.56990192
 123.85609989 128.4188022  101.87239835  88.97319879 131.47580051
 120.38290021 108.64349996 167.88880171 115.18740049  86.63149919
 118.88550085  91.02869954 161.67140019 116.70080023 121.72769989
 160.29579799 120.13049915 112.84169919 108.4013983  126.73169973
  76.25390013 103.00729967 127.66380311 121.81099923  92.64990027
 131.79550098 118.24520097 115.87129977 154.38890281 159.4787004
 109.90749968 154.1875974  119.23720084 160.76750032 118.51850054
 158.14589975 115.13659932 116.48420025 148.43709842 114.80400084
 125.80079856 165.29319932 117.63390002 125.18759952 153.08320356
 153.57160212 132.05219988 114.79130032 121.25020217 125.10350112
  89.69850054 122.80059981 154.7679019  111.47380034 106.8484999
 162.18090138 118.5760998  165.64200041 134.20030081 114.80049959
 152.92019862 168.72640043 115.48990051 114.11050107 159.36079834
  85.45549855 127.22579989 127.92460034 128.75930016 124.32510109
 123.98930092  90.71620081 153.19550046  97.20059962 136.86799977
  89.12889901 107.46229991 115.0106006  112.9899011  124.11279931
  91.41289887 125.50800141 162.37569837 120.03789864 165.05930131
 126.52849872 112.39820041 127.60469957  94.94689937  91.32659972
 103.05899908 120.91879977  83.41239934 126.43609985 161.03810493
 117.1456008  118.4965998  120.04679998 122.78469991 120.11670127
 121.42539989 118.05350067 107.12189948 148.2079998  126.21399796
 115.73490063  74.23109981 127.85220124 153.82810016 122.05850012
 125.60240047  88.89830042 104.08529862 124.45280057 120.17990003
  73.74350065 151.77710007 121.37730034 104.62190015  86.31149759
 115.14089934 172.15099951 119.88810054 160.3243984  113.1944995
 121.06979982 118.55960121  96.08399988 118.94760004
 125.86070037
 118.55569936  95.99640075 153.94680166 122.05640016 147.26970016
 159.66450213 113.9606001  122.58509908 148.76599738 127.12420026
 165.74150024 135.35300019 120.03929947 167.27829885 108.32929939
 121.60339865 139.41400181 106.40879885]
```

# EVALUATION

In [31]:

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

```
R squared error :  0.9898011359567865
```

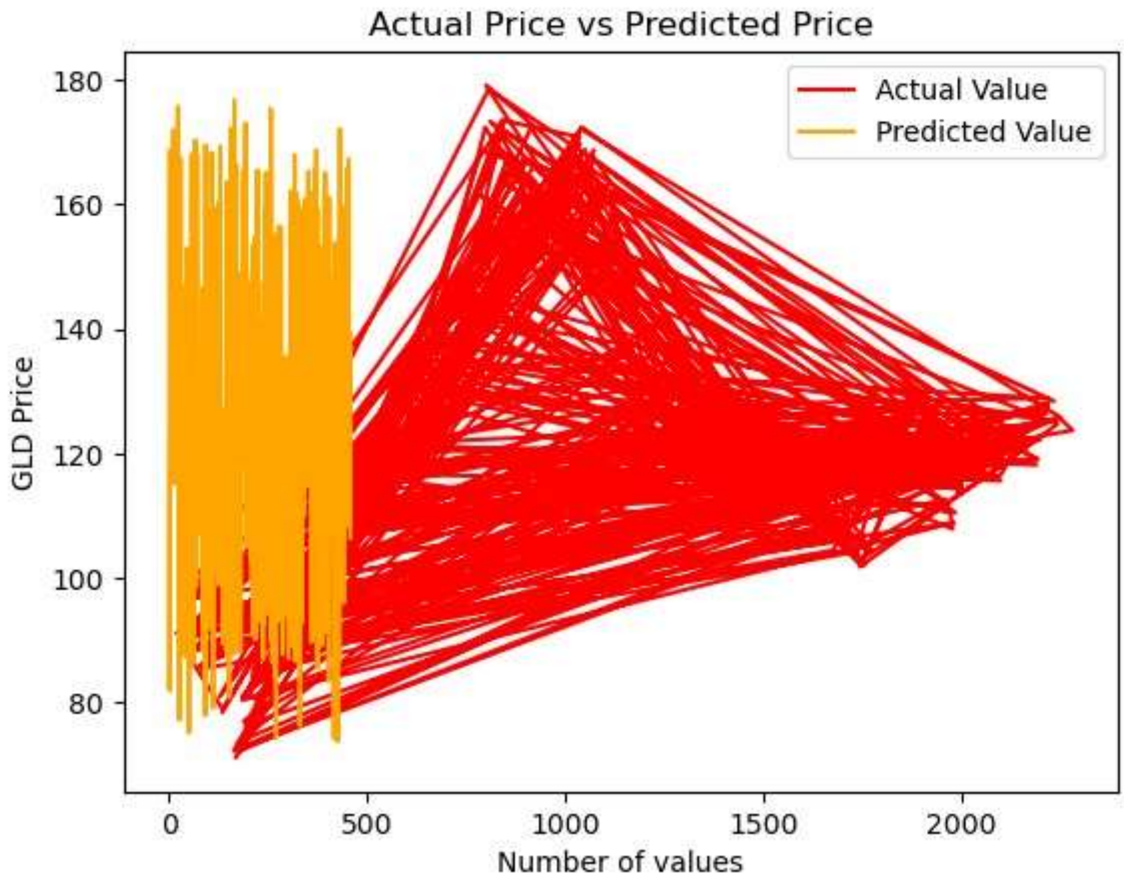
In [32]:

```
y_test = list(y_test)
```

# VISUALIZATION OF RESULTS

In
[58]:

```
plt.plot(y_test,color='red', label = 'Actual Value')
plt.plot(test_data_prediction, color='orange', label='Predicted Value')
plt.title('Actual Price vs Predicted Price')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```




# TRAINING THE MODEL: LINEAR REGRESSION

In [34]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler from
sklearn.linear_model import LinearRegression from
sklearn.metrics import mean_squared_error, r2_score
```

The concept of standardization comes into picture when continuous independent variables are measured at different scales. It means these variables do not give equal contribution to the analysis. Standardization is the process of putting different variables on the same scale. In regression analysis, there are some

In
scenarios where it is crucial to standardize your independent variables or risk obtaining misleading results.
[35]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [36]:

```
regression = LinearRegression()
regression.fit(X_train, y_train)
```

Out[36]:

```
LinearRegression()
```

# EVALUATION

In [37]:

```
y_pred = regression.predict(X_test)
```

In [38]:

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [39]:

```
print('Mean Squared Error:', mse)
print('R-squared:', r2)
```

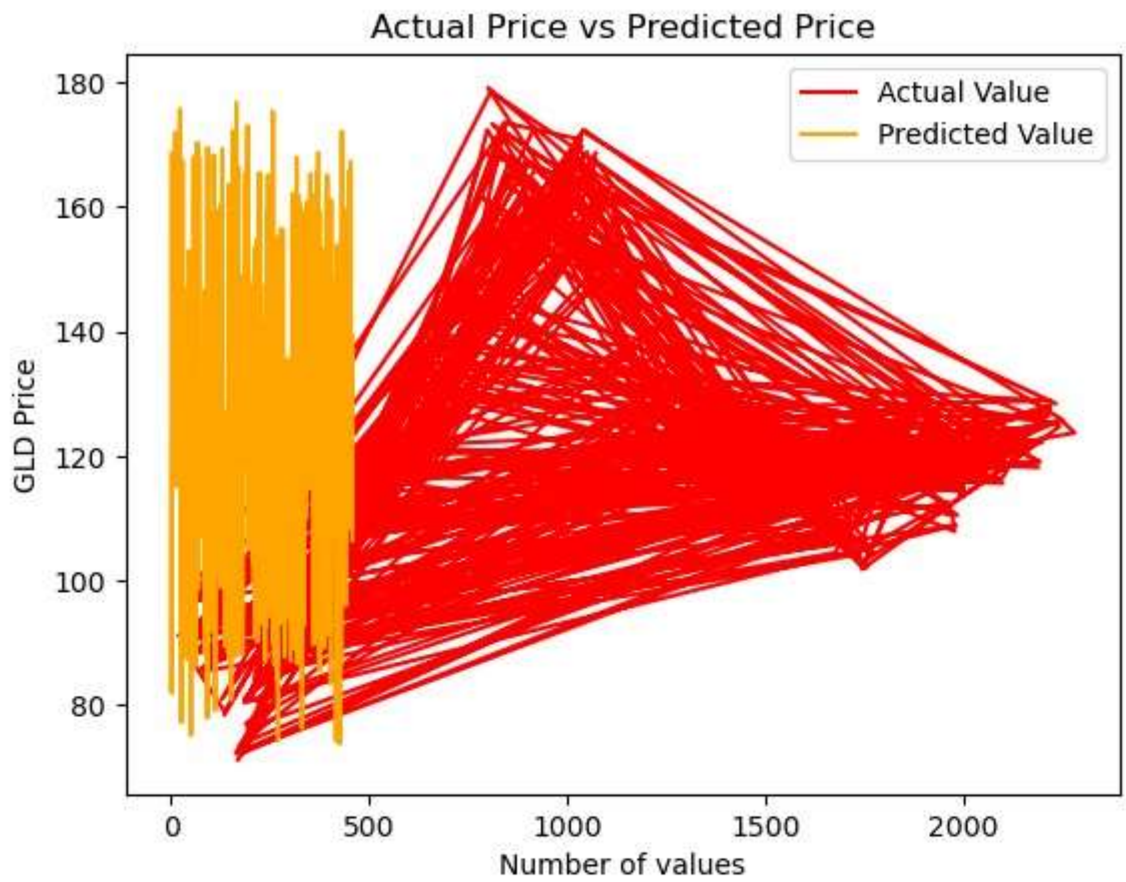
```
Mean Squared Error: 70.78890079721529
R-squared: 0.8657886565869237
```

In

# VISUALIZATION OF RESULT

[59]:

```
plt.plot(y_test,color='red', label = 'Actual Value')
plt.plot(test_data_prediction, color='orange', label='Predicted Value')
plt.title('Actual Price vs Predicted Price')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```




# Training the model: RIDGE

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

In [41]:

```
from sklearn.linear_model import Ridge
```

In
In [42]:

```
np.random.seed(42)
```

[43]:

```
model_3=Ridge()
model_3.fit(X_train,y_train)
```

Out[43]:

```
Ridge()
```

In [44]:

```
model_3.score(X_test,y_test)
```

Out[44]:

```
0.8658285385549093
```

It is necessary to standardize variables before using Ridge Regression. Ridge regression puts constraints on the size of the coefficients associated to each variable. However, this value will depend on the magnitude of each variable. The result of centering the variables means that there is no longer an intercept.

In [45]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [46]:

```
ridge = Ridge()
ridge.fit(X_train, y_train)
```

Out[46]:

```
Ridge()
```

In [47]:

```
y_pred = ridge.predict(X_test)
```

In [48]:

```
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
```

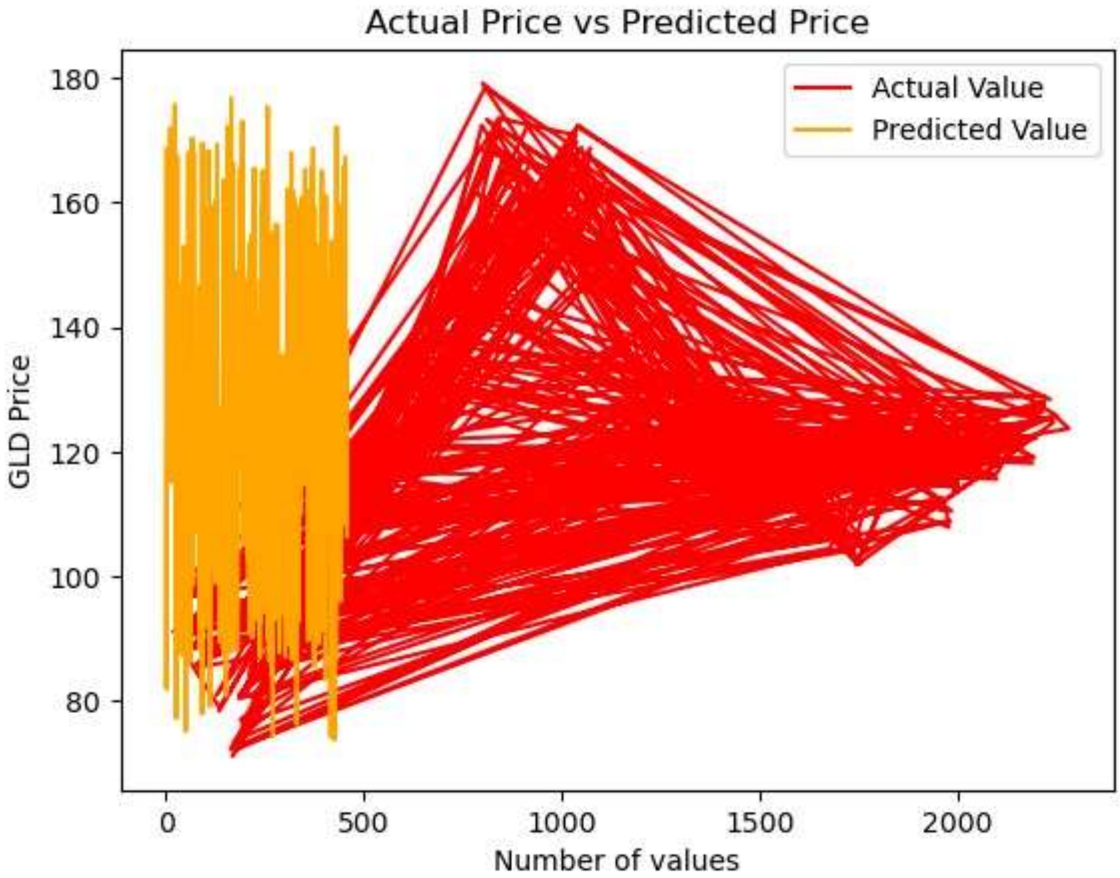
```
Mean Squared Error: 70.76786531240806
```

In

# DATA VISUALIZATION

[60]:

```
plt.plot(y_test,color='red', label = 'Actual Value')
plt.plot(test_data_prediction, color='orange', label='Predicted Value')
plt.title('Actual Price vs Predicted Price')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```




In [ ]:

```

```