GOLD PRICE PREDICTION

Data Collection and Processing

```
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
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Data Collection and Processing

```
# Loading the csv data to a Pandas DataFrame
gold_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/gold_price_data.csv')
```

```
# print first 5 rows in the dataFrame
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 5 rows of the dataFrame
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)
```

```
# getting some basic information 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
```

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

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

```
#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	1.303297
75%	2073.010070	132.840004	37.827501	22.882500	1.369971
max	2872.870117	184.589996	117.480003	47.259998	1.598798

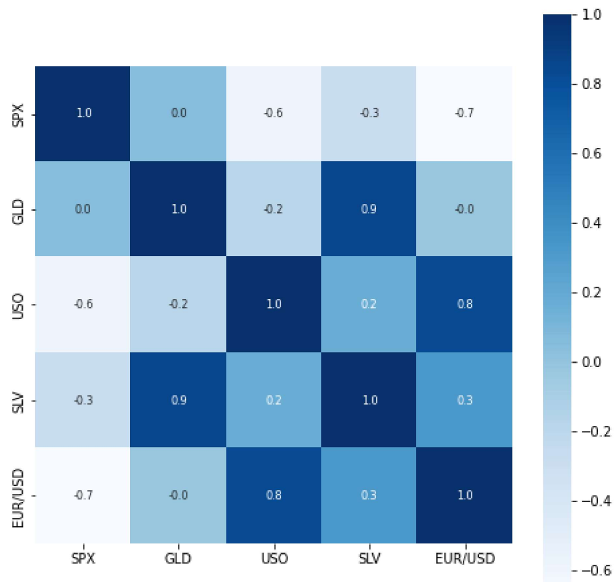
Correlation:

1. Positive Correlation
2. Negative Correlation

```
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 0x7f6366c745e0>
```

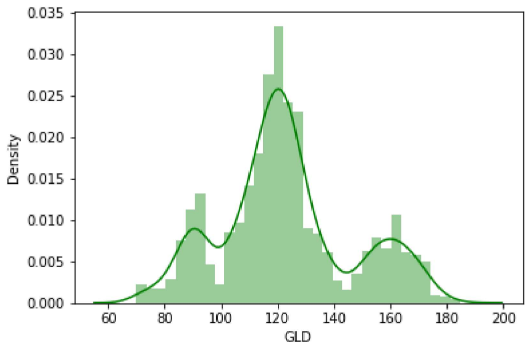



```
# correlation values of GLD
print(correlation['GLD'])
```

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

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a de
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f6364383ca0>
```



Spliting the Features and Target

```
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()
```

Model Evaluation

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

```
print(test_data_prediction)
```

```
[168.53069936  82.04249992 116.15289997 127.33470051 120.58860155
 154.68449766 150.35539894 125.8954004  117.57649872 126.0054009
 116.66900102 171.91140091 141.82589877 167.57799848 115.11180017
 117.20840043 138.87930296 169.9913006  159.20030284 160.04539937
 155.26740027 125.40520042 176.68119941 157.16340364 125.14410024
  93.72499986  78.59069999 120.59269995 119.07879929 167.47549998
  88.22000051 125.35190067  91.01340032 117.71640023 121.14979941
 136.35330083 115.43420125 115.43960097 146.80679998 107.0718011
 104.47580244  87.17629803 126.44590009 117.84989997 154.27109914
 119.69069998 108.33400027 108.01429821  93.05060038 127.04539781
  75.46750014 113.66769922 121.68660004 111.32469957 118.9097989
 120.60139964 159.34980006 167.85940138 146.94539639  86.01269886
  94.38140064  86.69409879  90.32529992 119.00980075 126.41380093
 127.57140019 170.31050042 122.2285994  117.38199924  98.45100002
 168.20320104 143.38279895 132.07800217 121.15530204 121.35959935
 119.66570073 114.26760199 118.26800068 107.07170084 127.91210064
 113.98709948 107.31840006 116.94210064 119.54759838  89.03970087
  88.17489853 146.46550278 127.17239988 113.26110043 110.39679852
 108.20899917  77.16249902 168.52460155 114.01499914 121.648599
 128.03790139 154.87289823  91.85199951 134.01170128 158.63420326
 125.74650043 125.20830076 130.3772012  114.68870145 119.8321
  92.14859977 110.24599899 168.64540014 157.97699907 114.03509937
 106.62430124  79.29469989 113.25230023 125.86220056 107.24049909
 119.47450097 155.65250315 159.44319851 120.26509991 134.72650279
 101.4741997  117.62599777 119.43960026 112.92550095 102.77939922
 159.93739808  99.07230037 147.36999883 125.97040151 169.83919913
 125.71529858 127.3853974  127.5481021  113.69749953 112.85600061
 123.56459896 102.13639893  89.1623     124.47169934 101.76309927
 107.07899915 114.11940044 117.39460049  99.12549961 121.90850018
 163.48419972  87.3794986  106.77860028 117.22500074 127.73690155
 124.25840062  80.65129893 120.09240076 158.28109787  87.94039941
 110.22199944 118.67159908 172.08589928 102.96059902 105.68810036
 122.35160035 159.12309766  87.64939864  93.23360057 112.70560042
 177.36389894 114.55209977 119.50940026  94.59350087 125.50390016
 165.96710037 114.79060097 116.6499014   88.23709865 148.9178001
 120.46429907  89.53749993 111.84720019 117.24350023 118.77070121
  88.05769931  94.1285998  117.10359997 118.74320171 120.21259998
 126.86399818 121.90000008 150.2228003  164.00209964 118.55249945
 120.21220148 151.26160018 118.12699886 172.11539935 105.05319937
 104.96030134 150.47430066 113.85460039 124.91480114 147.96429949
 119.57190097 115.08580034 112.7501001  113.51740194 141.95590217
 117.81559756 102.9981006  115.84430123 103.96530182  98.45650042
 117.34650053  90.73749994  91.5966001  153.88969898 102.74650022
 155.09480098 114.39710147 138.15270112  90.05519817 115.54139951
 114.67479937 122.75800045 121.65160037 165.33530177  92.89579986
 135.09220074 121.26429959 121.03790066 104.74110035 143.62840264
 121.57029949 116.58680047 113.17990107 126.99019775 122.48629927
 125.83709925 121.18560056  86.80019919 132.46730232 145.32630169
  92.87489924 158.07389974 159.08620242 126.24789945 164.82099901
 109.18459957 110.11900062 103.87329858  94.44500036 127.97800318
 107.0728005  161.56280002 121.94210012 131.93240061 130.41850141
 160.20379923  90.17639856 175.6140017  128.00610015 126.85959831
  86.35619923 124.6491993  150.03489713  89.69840011 106.99939994
 109.04949972  83.66329959 135.88090004 154.91080275 140.50710347
  74.47560021 152.51240068 126.18220005 126.6728004  127.4760988
 108.68149941 156.18620018 114.39140135 116.89640155 125.27179916
 154.26420181 121.23030005 156.38169952  92.86820046 125.49560119
 125.78610055  87.89940034  92.14939927 126.21999923 128.46230328
```

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

```
R squared error :  0.9894952432770892
```

Compare the Actual Values and Predicted Values in a plot

```
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()
```