Gold Price Prediction

Importing the 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 the csv data to a Pandas DataFrame
gold_data = pd.read_csv('/content/gld_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 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
```

```
# 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	1.303296
75%	2073.010070	132.840004	37.827501	22.882499	1.369971
max	2872.870117	184.589996	117.480003	47.259998	1.598798

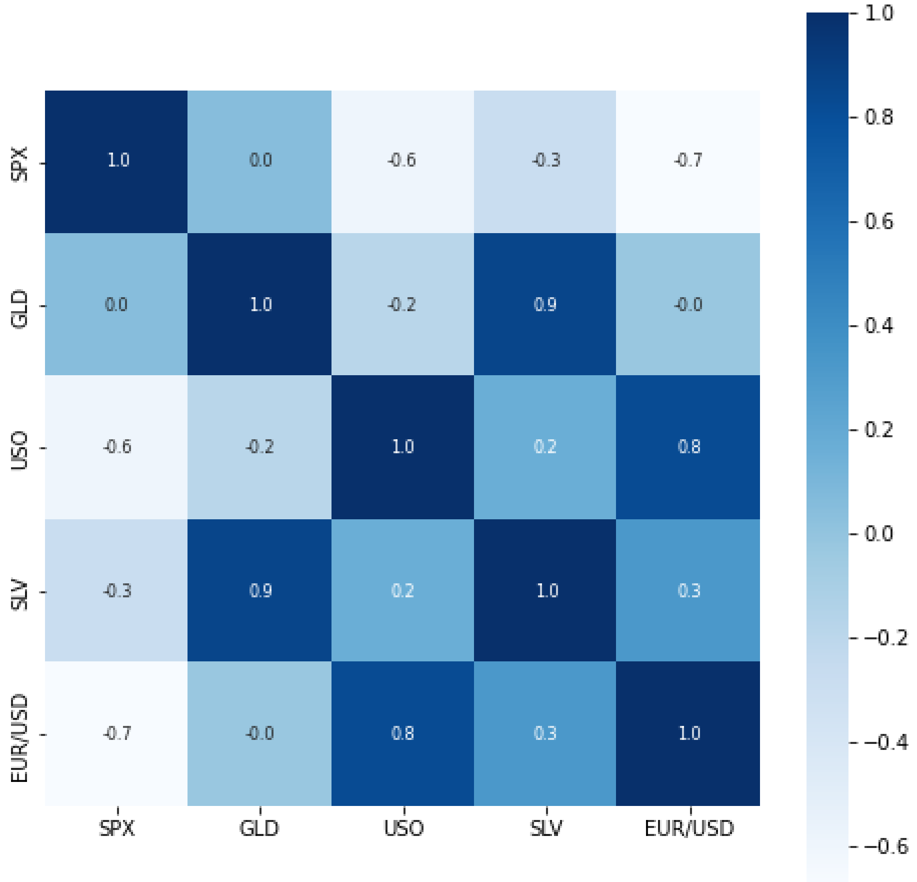
Correlation:

1. Positive Correlation
2. Negative Correlation

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc6c1339390>
```

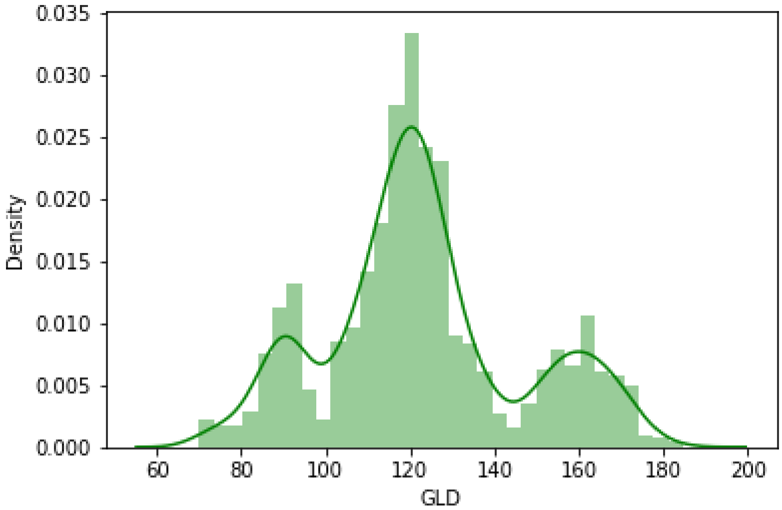



```
# correlation values of GLD
print(correlation['GLD'])
```

```
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/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fc6b8d1db50>
```



Splitting 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(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)
```

```
 177.38329918 114.64039959 119.19960017  94.60850088 125.9105002
 166.16170117 114.78830078 116.72750144  88.23249863 148.79410093
 120.43709949  89.49189963 112.42740005 117.16410062 118.76730107
  88.26819957  94.17959991 117.03980027 118.68860187 120.46700061
 126.69219821 121.93169977 151.0317001  164.8883009  118.57249957
 120.45210147 149.27170066 118.41109917 172.1916982  106.11009901
 104.91510102 149.02780119 113.93390085 124.77760132 147.57520008
 119.66130115 115.33560056 112.7179001  113.59920193 143.72850106
 118.06739759 102.8665003  115.95050147 103.96580166  98.65870041
 117.61180062  90.71590012  91.59580046 153.54069904 102.7856998
 155.15610086 114.2913015  139.59730133  90.08199819 115.45249941
 114.16229967 123.18690026 121.65890068 165.23500136  92.82109957
 134.85450148 121.37869913 120.94280086 104.61560026 141.97710287
 121.82589949 116.7792005  113.36690074 126.96039798 122.68459933
 125.75569952 121.24000017  86.87839896 132.24210114 146.27910094
  92.68879946 158.26489988 158.58370291 126.24269904 164.82119949
 108.8909997  109.69600103 103.69829825  94.45410041 127.55860275
 106.96670056 161.22579953 121.78870032 132.10340021 130.83690201
 160.73099973  90.05979869 174.11350152 128.60310058 126.76329874
  86.36809921 124.42249893 150.20059705  89.70189993 107.08719946
```

```
 108.97100004  85.13819853 135.97499947 154.48120263 139.21290345
  74.71359996 152.49590117 125.86930029 126.69040057 127.46729924
 108.5891995  156.02080028 114.43390151 117.01080169 125.39479912
 154.13720149 121.34759985 156.33229894  92.89730063 125.49800129
 125.6943005   87.82630023  92.0712993  126.23239924 128.8117044
 113.05190065 117.64609735 121.04040024 127.02099811 120.19050056
 136.63010135  93.88119911 119.84680048 113.28520079  94.16339945
 108.97819954  87.4174988  108.96709951  89.4240999   92.42290038
 131.82000302 162.36740104  89.36169991 119.56550052 133.29070178
 124.04260032 128.55100206 101.89989836  89.00159859 132.01170093
 120.07559995 108.28759999 168.94870105 115.2914005   86.63329889
 118.64610046  91.14609965 161.52070042 116.74270042 121.50179962
 160.35549832 120.15499921 112.75329906 108.44869881 126.71399985
  76.2044001  103.01509992 127.38300249 121.8899988   92.58240017
 132.79000111 118.0406011  116.22929972 154.10250311 158.65240054
 109.92139951 157.20539775 119.29380075 159.90010168 118.4356004
 157.60329976 115.03659929 116.73640027 149.9081986  114.90390075
 125.73989882 165.48489944 117.47949998 124.99299923 153.28720384
 153.37650249 132.10700054 114.84490043 121.15340214 125.02000073
  89.79180031 123.22240011 154.72580153 111.56170019 106.68719988
 162.68980167 118.34219971 165.62609875 134.12970126 115.07129979
 153.05659894 168.78400083 115.12760042 114.03420112 158.71769947
  85.21639922 127.0533007  127.91340015 128.82849954 124.37240057
 124.10310042  90.69620029 153.3353999   96.90439992 137.83170015
  89.0958994  107.70480008 115.1630006  112.6551009  124.44729896
  91.43679872 125.50450135 162.28739942 120.0169986  165.07400098
 126.73729813 112.25840014 127.6229994   94.96289867  91.15499989

 103.44289909 120.89539999  83.0883994  126.34360017 160.04540488
 117.36220109 118.4873999  119.83159971 122.79719982 120.11700122
 121.58769967 118.19740048 107.13569998 148.29159994 126.46119828
 115.81090131  74.10299996 127.8334011  154.2249012  122.7043002
 125.64650059  88.77689998 103.1299985  124.82390015 120.27730032
  73.33760061 151.69379971 121.19720067 104.6042      86.26739788
 115.19279914 171.87689806 119.7112005  159.74459713 113.25219969
 121.41780022 118.75260067  96.06619987 119.20199982 125.93160037
 118.52489962  95.80870028 153.76010203 121.98980047 147.55079988
 159.36440268 113.73570001 122.57969935 149.33399788 127.02320036
 165.90130023 135.30230085 120.07469949 166.6752973  108.3125992
 121.75459853 138.33960149 106.92659892]
```

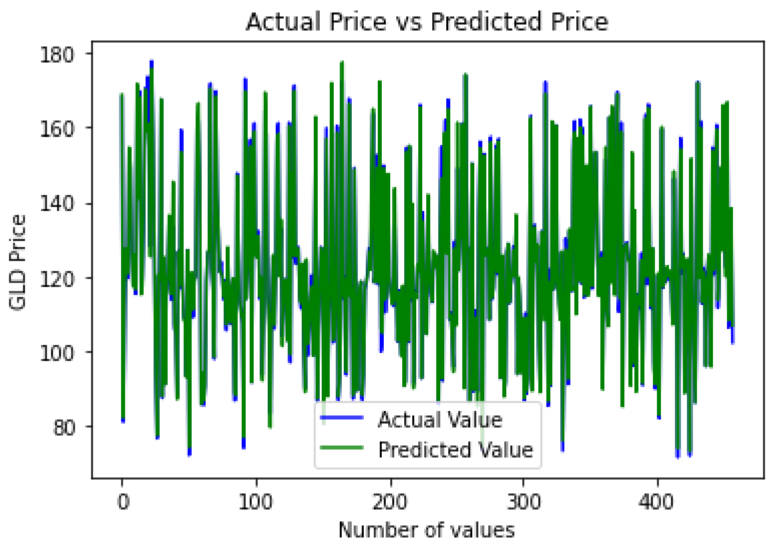
```
# R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error : ", error_score)
```

```
R squared error :  0.9893081607893282
```

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()
```



Score of train and test data

```
regressor.score(X_train,Y_train)
```

```
0.9984517280531134
```

```
regressor.score(X_test,Y_test)
```

```
0.9893081607893282
```

Making a Predictive System

```
#2671.919922  124.589996  14.0600 15.5100 1.186789
input_data = (2671.919922,  14.0600,  15.5100,  1.186789)

# changing the input_data to a numpy array
input_array = np.asarray(input_data)

# reshape the np array as we are predicting for one instance
input_reshaped = input_array.reshape(1,-1)

test_data_prediction = regressor.predict(input_reshaped)
print(test_data_prediction)

#expected output : 124.589996
```

```
[123.86529702]
```

✓ 0s completed at 4:54 PM