```
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
```

Importing the Libraries

Data Collection and Processing

```
# loading the csv data to a Pandas DataFrame
gold_data = pd.read_csv('/content/gold price dataset.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

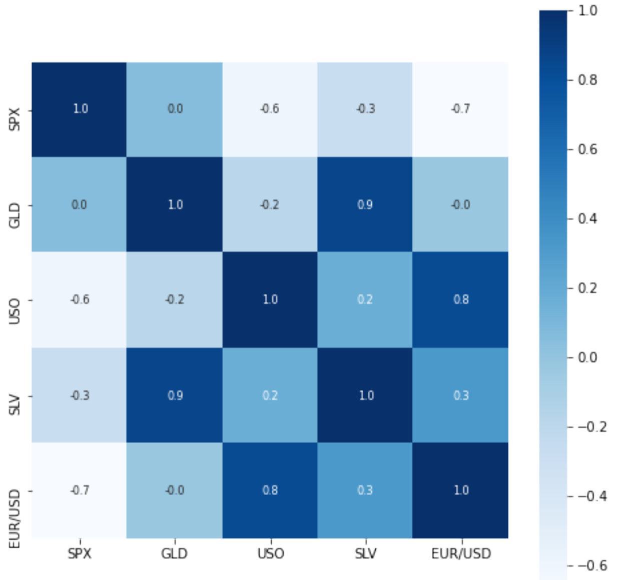
Correlation:

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}, cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff32443b350>
```

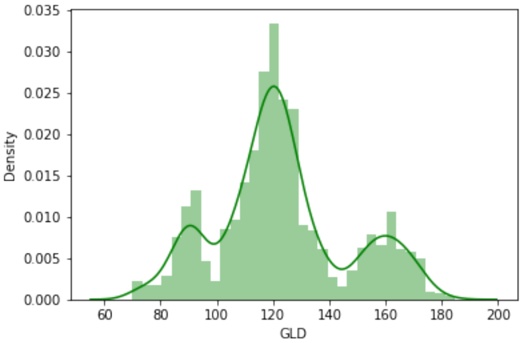



```
# correlation values of GLD
print(correlation['GLD'])
```

```
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:2557: FutureWarning: `distplot` is a deprecated function and will be
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7ff316c5da90>
```



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)
```

```
[168.32699968  81.94819986 115.70480041 127.41010064 120.90700068
 154.71489803 150.13359876 126.05480078 117.49259876 126.10170025
 116.68400094 171.42870056 141.30849872 168.02159836 115.23710006
 117.65880054 139.99590286 170.14650043 159.36550329 157.7788999
 155.08109978 125.5072004  175.80919981 157.22360324 125.23650052
  93.63189949  77.31300031 120.41329992 119.09699944 167.33849992
  87.85020039 125.33850039  91.04200093 117.60520038 121.21729884
 135.91539997 115.60030137 114.8546008  148.90609944 107.45720142
 104.09990231  87.17429793 126.55180059 118.02069986 153.00689898
 119.57450031 108.43489959 108.02569767  93.06320016 127.09119795
  75.14140033 113.55509908 121.47900041 111.35069881 118.91619888
 120.27279924 160.31060037 168.51620089 147.06449674  85.59369876
  94.32620022  86.79829933  90.3237999  119.05030078 126.43060051
 127.49300013 170.54360079 122.24279927 117.62959844  98.55510053
 168.06860158 142.99819816 132.09980232 121.19800249 120.89549925
 119.64790093 114.37230131 118.21010033 107.38680102 127.88100036
 114.0621996  107.15289997 116.76930037 119.70669871  88.9601005
  88.34369882 146.1299022  126.78910015 113.17350037 110.34049843
 108.33409912  77.08189908 168.39660184 114.21329907 121.57399938
 128.03540157 154.89909821  91.79399966 135.54530085 158.92480383
 125.3440006  125.46340044 130.65130101 114.68180071 119.81929957
  92.16999961 110.11749911 167.26549965 158.06019867 114.3765997
 106.54990112  79.40059997 113.24730066 125.75710085 107.12789965
 119.1806013  155.98480288 159.44939933 120.3244001  134.4500024
 101.62789996 117.47919801 119.19520031 112.88180075 102.75059929
 160.15779789  98.91920033 147.58789923 125.57450114 170.28349947
 125.6346993  127.27929753 127.43110165 113.62479943 113.1346007
 123.53849909 102.03549896  89.14589964 124.34589922 101.06539931
 107.10179967 113.92090066 117.34670086  99.18389956 121.84710054
 163.60659964  87.47259883 106.6078996  117.23410057 127.5852014
 124.05910082  80.83269909 120.32440061 157.5099982   87.92499947
 110.07529965 118.95049912 172.14249914 102.97889886 105.84530001
 122.52050047 157.49589778  87.70309815  93.52900017 112.55260046
 176.78399926 114.27400004 119.19190013  94.5947008  125.90419992
 166.05440018 114.78520032 116.87050117  88.41339902 148.8840007
 120.40529937  89.45950005 111.99399996 117.35729981 118.64940112
  88.46529954  94.0036998  116.95630046 118.65200177 120.35880083
 126.7777982  121.93069963 152.01010042 164.7022006  118.62569975
 120.29160153 149.76470051 118.45749925 172.58859947 105.53199931
 104.97940094 149.73020111 113.63000071 124.91270094 147.76489901
 119.57780101 115.28710044 112.54849993 113.44320195 139.84420093
 117.87479771 102.97830042 115.94670105 103.66380164  98.94920037
 117.2947009   90.61229989  91.50530088 153.32129875 102.70039963
 154.57390113 114.22570155 139.47170121  90.1379977  115.61339941
 114.45749983 122.55160026 121.84890017 165.25800212  92.75899948
 135.46920178 121.38429879 120.80620072 104.61770014 142.49950356
 121.50529903 116.91060067 113.49530121 127.0971974  122.83189941
 125.81279949 121.28810017  86.84599938 132.32730111 144.97030211
  92.74709948 158.60999965 158.99690266 126.50689884 164.98709962
 108.75659958 109.67190064 103.6128981   94.41100063 127.78350288
 107.12290043 163.50369972 121.72350055 132.08790076 130.71990146
 160.38430046  90.15699805 175.15240136 127.27550085 126.71979854
  86.46949934 124.64339988 149.78969723  89.67540032 106.65839979
 108.90399983  84.27789913 136.17700021 155.01210249 139.37430394
  73.66920014 151.96820164 126.19919988 126.80400001 127.48689897
 108.61909949 156.18609935 114.56220091 117.04250138 125.31619929
 154.20120192 121.41000021 156.36689873  92.90840064 125.53660142
 125.41140015  87.84860072  92.1129991  126.38899933 128.27820349
```

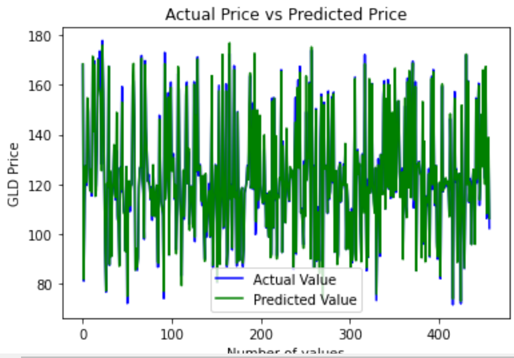
```
# R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error : ", error_score)
```

```
R squared error :  0.9887338861925125
```

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()
```


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