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/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:

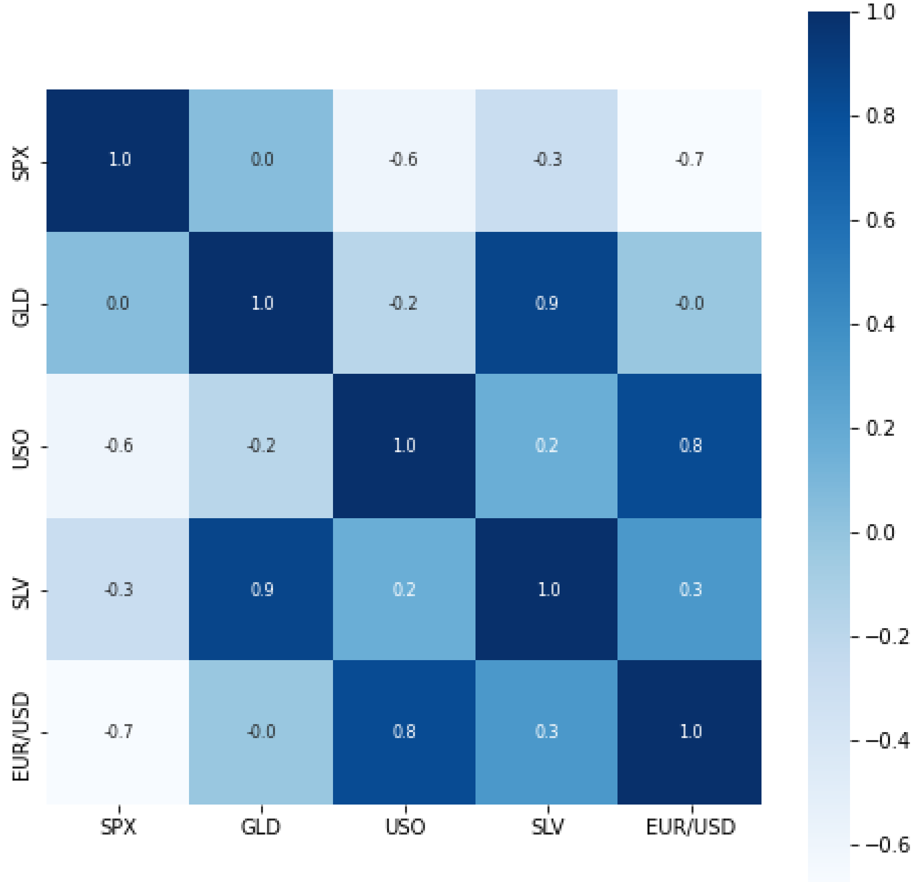
1. Positive Correlation
2. Negative Correlation

```
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 0x7f9b8713dd90>
```

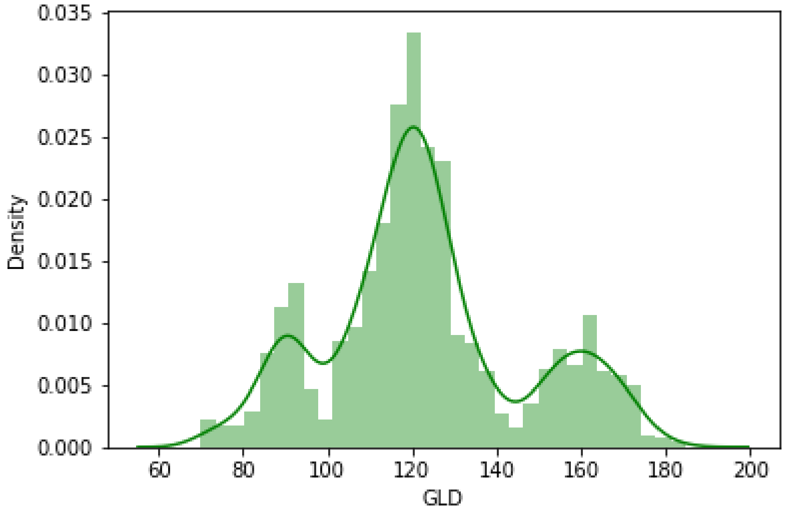



```
# correlation values of GLD
print(correlation['GLD'])
```

```
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 0x7f9b7eb48d10>
```



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)
```

```
[168.3484999   81.89479972 116.09899991 127.69320058 120.6678014
 154.64199797 150.29979794 126.1372009  117.32869872 126.05860017
 116.78590099 171.82160065 141.67249943 168.13139879 115.26520006
 117.84270075 138.18980393 170.53460116 159.40120335 161.35379953
 155.09139994 125.24259982 175.34029886 157.21560335 125.23890061
  93.54179984  77.49010037 120.33620034 119.08419914 167.47819976
  88.36190011 125.21470001  91.15690099 117.56870016 121.09759924
 136.62700064 115.43260114 114.98480054 148.30529953 107.08060101
 103.89910228  87.09729763 126.40650103 118.09849994 153.8537995
 119.62279991 108.44489975 107.95079803  93.27360055 127.20929763
  74.53400072 113.68079949 121.40410017 111.25329934 118.99549913
 120.4207995  158.84929985 166.5354016  146.935397    85.83249822
  94.5964002   86.82279855  90.67270008 119.04590045 126.39880042
 127.46510019 170.19969979 122.37269909 117.54209901  98.64660031
 168.46550132 143.36809821 132.38980249 121.16510245 121.02369943
 119.75890076 114.65130097 118.32750041 107.12510106 127.86670123
 114.07409996 107.33570003 116.88020047 119.66279888  88.66810037
  88.19809859 146.24560254 127.11990008 113.22620053 109.98449842
 108.23459891  77.56429919 169.65600203 113.9485989  121.59029921
 127.93700185 154.93959825  91.79809946 134.49840096 159.09970373
 125.45590051 125.33820081 130.46190155 114.94630071 119.76059963
  92.07289991 110.29899897 168.290599   155.95269868 114.21199941
```

```
 106.64040114  79.46569974 113.32430027 125.81260082 107.09289937
 119.72390131 155.63260301 159.82479958 120.20959999 134.24330334
 101.44839974 117.97099785 119.2060998  112.83120073 102.74529921
 160.38569808  99.11640068 148.84579928 125.396601   169.8935994
 125.97409874 127.28289768 127.41970218 113.93649913 113.27420071
 123.64909903 102.02879878  89.40389967 124.80879963 101.97859919
 107.0984992  113.63120078 117.28350102  99.18269988 121.86410053
 162.97839772  87.31539837 106.83929972 117.29530048 127.68390126
 123.9635008   80.72969922 120.21770071 157.9907979   87.91629953
 110.23619972 118.83359908 172.15179941 102.93659901 105.66260038
 122.58810027 158.27599741  87.51009834  93.09810036 112.64150034
 176.33059928 114.34539968 119.20059976  95.01670122 125.63670051
 166.04360136 114.81310016 116.76880106  88.23309854 148.60900083
 120.3918992   89.48439971 111.91309986 117.10079993 118.76680113
  88.43249925  94.28980008 116.86589996 118.564202   120.4035005
 126.83749799 121.91869994 148.52390014 164.98050105 118.66659959
 120.31970128 151.14460053 118.09589922 172.8204992  105.58109948
 105.02430093 149.46570113 113.63430084 124.82720091 147.2965993
 119.58530118 115.19000014 112.68619988 113.46030221 142.61690134
 117.88729784 102.89380038 115.85720088 103.07940178  98.92810045
 117.59920036  90.66089985  91.7720002  153.43869954 102.66089992
 154.95050106 114.42410184 138.56980162  90.16749802 115.46039954
 114.19110001 122.98360033 121.77160018 165.36960127  92.98899928
 135.41280154 121.43429902 120.9415007  104.5943002  143.39870262
 121.48619929 116.65100024 113.4262009  127.20729731 122.94959927
 125.79559969 121.2850003   86.93939859 132.51810128 143.38510209
  92.64919961 158.04299967 159.34220339 126.13559882 164.29059918
 109.10089929 109.65340103 103.54759811  94.1780008  127.84630296
 107.01740067 160.37620006 121.73450033 132.22420038 130.59770083
 160.23900066  90.24049825 175.01370125 128.15310068 126.87449813
  86.41559882 124.55709973 150.14879741  89.68429997 106.91579956
 109.14169994  83.40619907 135.93460015 155.28530151 139.30180288
  74.21080034 152.47450084 126.13800014 126.70380027 127.49319877
 108.54039968 156.27689965 114.79890088 116.81050139 125.18019916
 154.13900124 121.41889992 156.36819887  93.02010046 125.48780176
 125.7820005   88.08340061  92.3964989  126.1256995  128.32190344
```

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

```
R squared error :  0.9889921470188079
```

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()
```