Gold Prediction Model

Workflow:

1. Finding data set for gold's prices
2. Preprocessing of the data
3. Analyse the data to understand which factos are important and what is not
4. Spliting our data set into Training Set and Test Set
5. Using Random Forest Regressor Model we will prefict the price
6. Evalutation of our model on the basis of the Test Set

Gold Prices Data Set : Link

Importing all the necessary libraries

In [114…
```python
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
# As we are using two data sets one for testing and another for training we are using the train_test_split function
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
# It is used to find things like the error score, accuracy of model etc
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

Collection of Data and Preprocessing

In [19]:
```python
# Looding the csv fine using Pandas
gold_dataset = pd.read_csv(r"C:\Users\Yash Rajput\Desktop\Gold Price Prediction\Data Set\gld_price_data.csv")
gold_dataset.head()
```

Out[19]:

	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

Understanding the Data

1. Data: It is the date on which the values are noted in the format of MM/DD/YYYY
2. SPX: SPX indicates the S&P 500 index, it is the captilisation for top 500 companies in US (basically it is a stock)
3. GLD: It is the prices of gold on the corresponding dates
4. USO: It inidcates the United States Oil prices
5. SLV: It is the prices of silver on the corresponding dates
6. EUR/USD: It is the currency pair of euro and dollar

In [25]:
```python
gold_dataset.tail()
```

Out[25]:

	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 [26]:
```python
gold_dataset.shape
```

Out[26]:
```
(2290, 6)
```

Our Data Set Contains Dates from 2nd January 2008 to 16 May 2018

Roughly we have a dataset of 10 years

2290 values to be precise

Basic Information About the Data

In [27]:
```python
gold_dataset.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
```

We see that there aren't any null values

In [28]:
```python
# We can also check the number of null values with this method
gold_dataset.isnull().sum()
```

Out[28]:
```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

Gathering Statistical Measures of the Data

In [29]:
```python
gold_dataset.describe()
```

Out[29]:

	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

Analysing the data to find correlation

Identifying whether its a positive correlation or a negative correlation

understanding correlation in a very simple and crude way:

1. Positive Correlation: Directly Proportional
2. Negative Correlation: Inversely Proportional

In [33]:
```python
correlation = gold_dataset.corr()
```

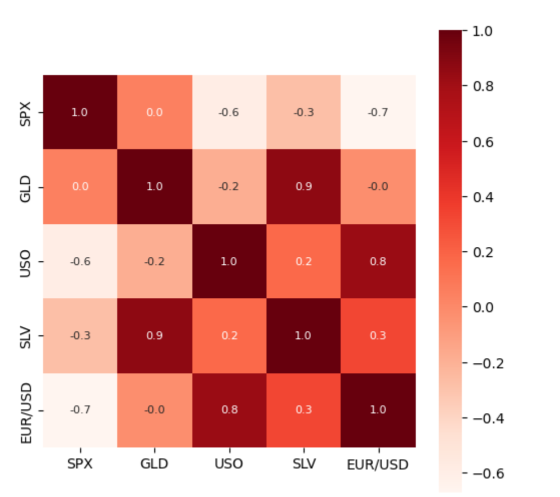
```
C:\Users\Yash Rajput\AppData\Local\Temp\ipykernel_26444\4191627151.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  correlation = gold_dataset.corr()
```

Constructing A Heat Map to understand the correlation

In [66]:
```python
plt.figure(figsize = (6,6)) # Size of the heat map

sb.heatmap(correlation, square=True, cbar = True, fmt= '.1f',annot=True, annot_kws={'size':8}, cmap='Reds')
```

Out[66]:
```
<Axes: >
```



Understanding the parameters used above

1. cbar = True: It shows the color bar on right which acts like a scale
2. square = True: We wanted the map in square format, otherwise it would have been in rectangle format
3. fmt = '.1f': It it for the number of decimals we want
4. annot = True: It is for the values we see on the small squares
5. cmap = 'Reds': It is for the color theme of the map

In [60]:
```python
# Correlation values of Gold
print(correlation['GLD'])
```

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

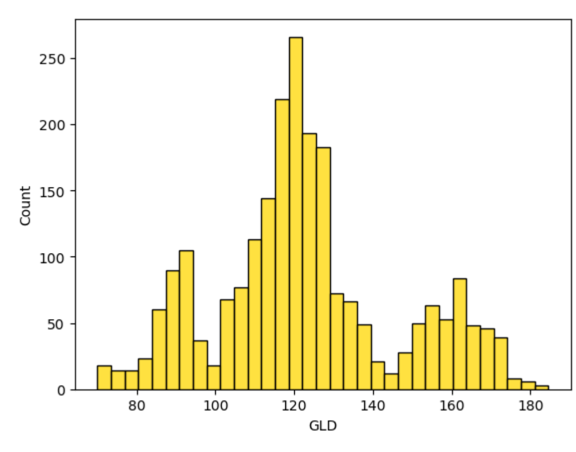
Our Understanding

1. Gold and SP 500 are slightly positively correlated
2. Gold and US Oil is negatively correlated
3. Gold and Silver are positively correlated
4. Gold and currency pair is slightly negatively correlated

Checking the distribution of the Gold's price over these 10 years

In [70]:
```python
sb.histplot(gold_dataset['GLD'],color='Gold')
# Note: .distplot() was used in older versions, now we use .histplot()
```

Out[70]:
```
<Axes: xlabel='GLD', ylabel='Count'>
```



We can conlcude that majority of these values lie between 100 and 140

Splitting Of The Data Set

We use the values of SPX, USO, SLV, EUR/USD to train our model to predict the price of GLD. In this process we will drop the date column

In [85]:
```python
X = gold_dataset.drop(['GLD','Date'],axis=1) # For droping columns axis = 1 and for rows axis = 0
Y = gold_dataset['GLD']
```

In [86]:
```python
X
```

Out[86]:

	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 × 4 columns

In [87]:
```python
Y
```

Out[87]:
```
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
```

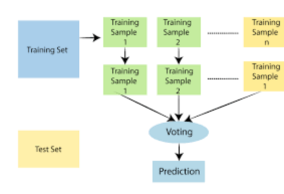
Splitting into Training Data and Test Data

In [88]:
```python
 X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, random_state = 2)
```

Understanding the parameters used above

1. test_size = 0.2 : This means that 20% of the data available will be used for tesing
2. random_state = 2 : It tells us about the order of randomness used while sorting test set and training set

Understanding Random Forest Algorithm



Explaination Video

Our Model: Random Forest Algorithm

In [92]:
```python
regressor = RandomForestRegressor(n_estimators=100)
#is a hyperparameter that determines the number of decision trees to be used in the random forest ensemble.
```

Training The Model

In [93]:
```python
regressor.fit(X_train,Y_train)
```

Out[93]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

Model Evaluation

In [94]:
```python
test_data_prediction = regressor.predict(X_test)
```

In [95]:
```python
test_data_prediction
```

Out[95]:
```
array([168.59779942,  81.94749999, 115.97200043, 127.64660075,
       120.4747015 , 154.77799773, 150.2583978 , 126.01530059,
       117.391699  , 126.03410071, 116.64130072, 172.48760055,
       141.84029836, 167.8545987 , 115.11609997, 117.3707004 ,
       137.85520294, 169.85600043, 158.75230252, 159.45609943,
       155.05780021, 125.36950007, 175.96489984, 156.7293033 ,
       125.23580038,  93.79899992,  77.10950034, 120.6270002 ,
       119.10879939, 167.4877997 ,  88.39940066, 125.30700029,
        91.32460104, 117.54200064, 121.01279913, 136.62060095,
       115.36720137, 115.14500059, 147.94910002, 107.35880093,
       104.39280232,  87.27009791, 126.4887005 , 118.09130018,
       153.33169955, 119.76990013, 108.43290014, 108.08469806,
        93.11150013, 127.00449803,  74.64160048, 113.69479908,
       121.33710016, 111.26739879, 118.80469878, 120.62919929,
       159.92640113, 167.75630113, 146.76969697,  85.71479853,
        94.18380064,  86.68229887,  90.61720038, 119.0535006 ,
       126.5213005 , 127.4849002 , 169.79329881, 122.27609966,
       117.45259905,  98.5311005 , 167.74000143, 142.7704986 ,
       131.43360258, 121.19970194, 121.2431995 , 119.94870053,
       114.4855014 , 118.50760047, 107.23300059, 127.97200082,
       113.85849979, 107.34359982, 117.01050113, 119.70439834,
        88.83120042,  88.23399888, 146.32610217, 127.13319997,
       113.24750037, 110.16599863, 108.16759895,  77.46519914,
       169.88780162, 113.97629909, 121.67539915, 127.99920196,
       155.00499776,  91.70439951, 135.88540068, 159.10090371,
       124.62930092, 125.41870016, 130.38240206, 114.81290109,
       119.77919987,  92.0121998 , 110.1818989 , 167.70189821,
       156.89179901, 114.07029928, 106.7375014 ,  79.40979961,
       113.31900036, 125.87760083, 107.26729937, 119.50720087,
       156.29030366, 159.94459889, 119.81379987, 135.23460264,
       101.283     , 117.3830983 , 119.24190066, 112.91870086,
       102.78019907, 160.29479794,  98.77640035, 147.25949922,
       125.59630108, 170.09739905, 125.77099897, 127.35639742,
       127.50390219, 113.90879903, 112.87300058, 123.5970992 ,
       102.15289902,  89.44110004, 124.44589946, 102.55319945,
       107.02199927, 113.88270045, 117.34870053,  99.01739927,
       121.60060059, 163.2575985 ,  87.35629883, 106.7312997 ,
       117.15390069, 127.77100157, 124.05590081,  80.48079917,
       120.28140068, 158.17949765,  88.22829977, 110.2       ,
       118.81519905, 173.27239812, 102.92029908, 105.59110019,
       122.38490034, 158.36569728,  87.68309828,  93.1258005 ,
       112.79020026, 176.49789982, 114.16029994, 119.1817007 ,
        94.58130093, 125.44919972, 166.1195013 , 114.71770088,
       116.67330141,  88.27409881, 148.75770096, 120.32279974,
        89.64409972, 111.56580013, 117.45190051, 118.72310122,
        88.18299941,  94.0663001 , 117.22040018, 118.5705019 ,
       120.11930048, 126.93449793, 121.8594998 , 148.67860001,
       165.9817007 , 118.46369965, 120.16360146, 150.48710085,
       118.49019895, 173.14809885, 105.33889912, 105.18000083,
       148.98620087, 113.92650063, 124.83600108, 147.17910022,
       119.70370085, 115.39090069, 112.33080022, 113.55690175,
       141.83620154, 117.7263979 , 102.95140056, 115.81800115,
       103.49970167,  98.65670049, 117.42530051,  90.48170015,
        91.49590069, 153.26529932, 102.67199968, 155.2979011 ,
       114.40740138, 138.26870131,  90.25079813, 115.52399954,
       114.93789946, 122.57260064, 121.73980018, 165.40900169,
        92.93279948, 135.75100159, 121.39289908, 120.84420017,
       104.53320008, 142.48690283, 121.70179908, 116.76890055,
       113.4543009 , 126.9689978 , 122.52639974, 125.8922995 ,
       121.23870058,  86.85889876, 132.25760171, 143.601602  ,
        92.75529909, 158.25299921, 158.81030251, 126.19819881,
       165.09559922, 108.95799955, 110.28320093, 103.63509826,
        94.32780093, 128.0423032 , 107.05160065, 162.37179954,
       121.58450054, 132.0514998 , 130.37580157, 160.44479992,
        90.03529822, 175.14410177, 127.11890059, 126.74469836,
        86.48949903, 124.49739941, 150.53979729,  89.6980999 ,
       107.00279976, 109.0172998 ,  83.6954992 , 135.96559943,
       155.07800279, 140.25480304,  74.1399001 , 151.39770127,
       126.12920009, 126.74600019, 127.56649885, 108.54149929,
       156.41829993, 114.49170093, 116.77670118, 125.33369964,
       154.00140127, 121.03890008, 156.42119878,  92.95370064,
       125.49850132, 125.35850058,  87.8959003 ,  92.26359933,
       126.3530992 , 128.32490393, 113.2928005 , 117.54299754,
       120.96049983, 127.28449795, 119.47930106, 136.35370054,
        94.05039953, 119.74520051, 113.2532011 ,  94.39799956,
       108.94979906,  87.23389926, 108.88909949,  89.91959968,
        92.39020011, 131.77830277, 162.30630061,  89.40510004,
       119.6453008 , 133.50470188, 123.9997001 , 128.45870199,
       101.91369863,  88.97839881, 131.70680071, 119.84030063,
       108.90029993, 168.05180073, 115.2575004 ,  86.7158989 ,
       118.6883007 ,  91.08399981, 162.34449986, 116.33420023,
       121.50100008, 160.19569832, 120.05079929, 112.84689916,
       108.55029886, 126.79679985,  76.12010048, 102.99169992,
       127.83120314, 121.69669951,  92.61270028, 131.81100028,
       117.931301  , 115.6868997 , 154.82110242, 159.53660067,
       110.17969925, 153.3835978 , 119.37480083, 160.55470074,
       118.54860049, 157.68909981, 115.11469901, 116.57790027,
       148.5680991 , 114.80010066, 125.62859927, 165.85159973,
       117.78549997, 125.10679951, 153.50900333, 153.57070184,
       132.08999987, 114.63560031, 121.12870236, 124.83360069,
        89.9409005 , 123.15969954, 155.05890241, 111.65020028,
       106.70040015, 161.50400136, 118.36139953, 165.71959913,
       133.95410113, 114.9148996 , 152.97439817, 168.59549941,
       115.48390032, 114.0941014 , 157.7225985 ,  85.39429888,
       127.12160093, 127.9934005 , 128.88999939, 124.06690073,
       123.95770057,  90.67560082, 153.25520096,  96.98929982,
       138.02110002,  89.29929915, 107.22899991, 115.04290049,
       112.75000069, 124.15249923,  91.46789874, 125.35130079,
       162.29669965, 119.88589886, 165.13470179, 126.82899786,
       112.21850018, 127.55569907,  94.98039943,  90.89539971,
       103.97719931, 120.85199999,  83.20229976, 126.25039984,
       159.62110452, 117.20110104, 118.36729978, 119.91680006,
       122.71389961, 120.0998011 , 121.31760036, 118.20190081,
       107.0984    , 148.26949991, 126.43239851, 115.7475009 ,
        74.13349984, 127.8619015 , 154.29390018, 122.13260027,
       125.62060116,  88.95189954, 103.03959852, 124.6219003 ,
       120.23520004,  73.33580082, 151.78179992, 121.24340011,
       104.9882    ,  86.46629777, 115.05889917, 172.1888983 ,
       119.86150021, 159.5119988 , 113.23729954, 121.35839995,
       118.67080126,  95.96769989, 118.88350072, 125.74020035,
       118.48309954,  96.23820093, 153.83170188, 122.01310024,
       147.73829998, 159.35470221, 113.57050015, 122.44099937,
       150.27499849, 127.3366004 , 165.98180015, 136.08310089,
       119.9712993 , 167.31469858, 108.36799996, 121.57919868,
       139.27230107, 106.78889899])
```

R Squared Error

In [96]:
```python
error_score = metrics.r2_score(Y_test,test_data_prediction)
error_score
```

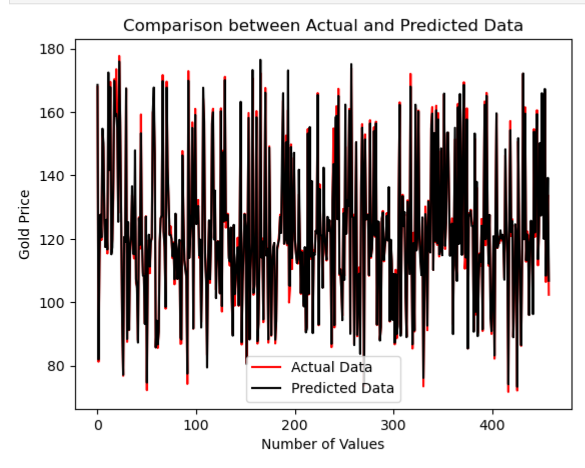
Out[96]:
```
0.9892294696357877
```

Conclusion: As we have the values in range of 100s the error of 0.989 is low which means our model is performing well

Visualising the error using graphs

In [109…
```python
# Y_test is data frame we need to convert it into list
Y_test = list(Y_test)
```

In [110…
```python
plt.plot(Y_test, color='red', label='Actual Data')
plt.plot(test_data_prediction, color='black', label='Predicted Data')
plt.title('Comparison between Actual and Predicted Data')
plt.xlabel('Number of Values')
plt.ylabel('Gold Price')
plt.legend()
plt.show()
```



As we see the the Acutal Data and Predicted Data are almost overlapping

Finding the Accuracy of our model

In [118…
```python
def mean_absolute_percentage_error(y_true, y_pred):
    total_error_percentage = 0.0
    for a, b in zip(y_true, y_pred):
        if a != 0:  # To handle cases where the true value is zero
            total_error_percentage += abs((a - b) / a)
    mean_error_percentage = (total_error_percentage / len(y_true)) * 100
    return mean_error_percentage

# Assuming you have list1 (actual values) and list2 (predicted values)
accuracy_percentage = 100 - mean_absolute_percentage_error(Y_test,test_data_prediction)

print(f"Accuracy Percentage: {accuracy_percentage:.2f}%")
```

```
Accuracy Percentage: 98.92%
```

Our Model is 98.92% Accurate

By Yash Rajput

In [ ]: