

## ▼ # Project 01: Gold Prices Prediction

### #Day10 of #30DaysOfMachineLearning

Getting Start with our Fir Project **Project 01: Gold Price Prediction** Dataset is Downloaded From Kaggel. The main Purpose to Build This project is to predict The Gold Pricess.By applying Some ML Algorithms.

If You Like Please UpVote.

## ▼ Importing the Libraries/dependencies

1. **Numpy**: It provides a multidimensional array object, as well as variations such as masks and matrices, which can be used for various math operations.
2. **Pandas**: Pandas has been one of the most commonly used tools for Data Science and Machine learning, which is used for data cleaning and analysis. Here, Pandas is the best tool for handling this real-world messy data.
3. **matplotlib**: Matplotlib is a library in Python and it is numerical – mathematical extension for NumPy library. Pyplot is a state-based interface to a Matplotlib module which provides a MATLAB-like interface. There are various plots which can be used in Pyplot are Line Plot, Contour, Histogram, Scatter, 3D Plot, etc.
4. **Seaborn** is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.
5. **Sklearn**: It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Double-click (or enter) to edit

```
import numpy as np
#pandas is used to read the cvs file of our DataSet.
import pandas as pd
#matplotlib for making plots and graphs
import matplotlib.pyplot as plt
#next we will use seaborn and it is also usefull for making plots and graphs
```

```
import seaborn as sns
#from sklearn we will import model_selection so we need to split the original data into Tra
from sklearn.model_selection import train_test_split
#Now we will import our random forest regressor model
from sklearn.ensemble import RandomForestRegressor
#now from sklearn import metrics that is useful for finding the performances of our mode
from sklearn import metrics
```

## ▼ Data Collection and Processing

```
# loading our dataset (the csv data to a Pandas DataFrame) creat a variable and load data
data = pd.read_csv('../input/gold-price-data/gld_price_data.csv')
```

```
#after successfully importing our csv to pd Dataframe. we will print out first 5 rows in th
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

We have the data from 2008 and **SPX**spx is also called Cmp index it is the capitalization index of 500 companies which are publicly traded. **GLD** Are Gold prices **USO** Represents United State Oil Prices **SLV** Silver Price Value **EUR/USD** Currency pair of European and United States

```
# print last 5 rows of the dataframe
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

```
#Printing Total number of rows and columns in Our Dataset/Data
data.shape
```

(2290, 6)

```
# getting some basic informations about the data
#the info function give us information about Number Of Entries and Number of Columns and D
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 in our Data by applying isnull function
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. The describe function will give us some st
data.describe()
```

	SPX	GLD	USO	SLV	EUR/USD
<b>count</b>	2290.000000	2290.000000	2290.000000	2290.000000	2290.000000
<b>mean</b>	1654.315776	122.732875	31.842221	20.084997	1.283653
<b>std</b>	519.111540	23.283346	19.523517	7.092566	0.131547
<b>min</b>	676.530029	70.000000	7.960000	8.850000	1.039047
<b>25%</b>	1239.874969	109.725000	14.380000	15.570000	1.171313
<b>50%</b>	1551.434998	120.580002	33.869999	17.268500	1.303297
<b>75%</b>	2073.010070	132.840004	37.827501	22.882500	1.369971
<b>max</b>	2872.870117	184.589996	117.480003	47.259998	1.598798

Lets do Some Analysis on data so we will find the corellation between the various columns in dataset there are two types of correlation. while we are working on regressin projects we will

always check this correlation. so it tells us the which columns are related to which columns.

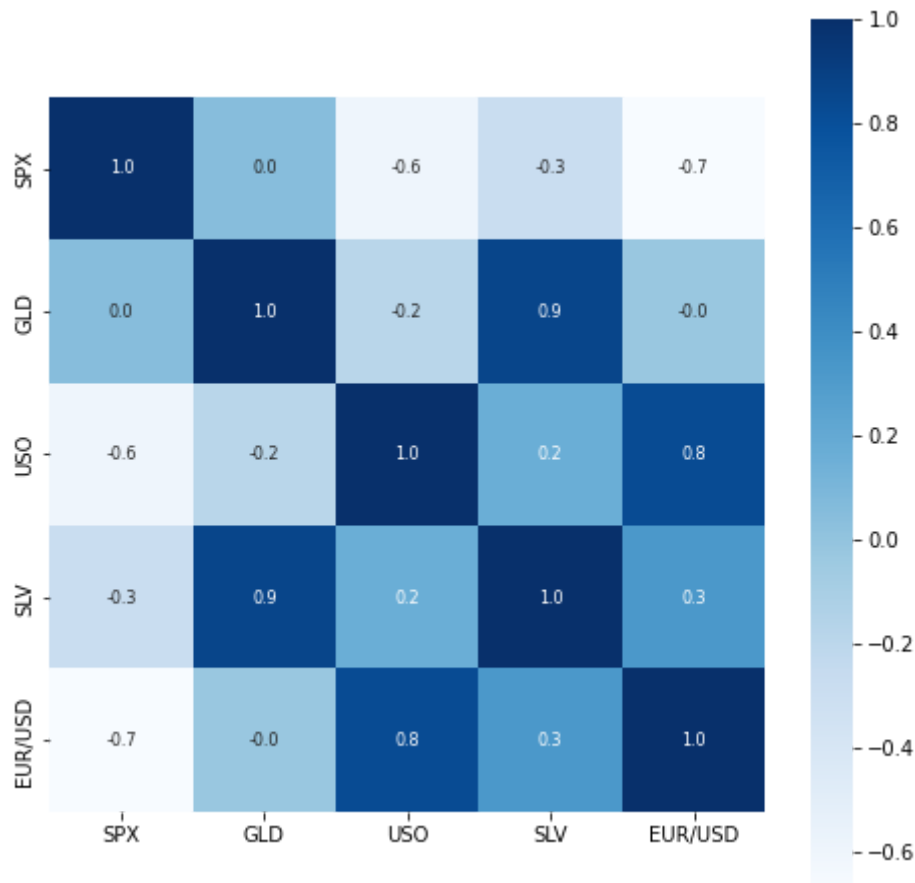
Correlation:

1. **Positive Correlation** in case of positive correlation when we take two variables, one variable will increase if the other variable decrease. so such kind of relations are known as positive correlation. we can say that these variables are directly proportional to each other.
2. **Negative Correlation** in Negative correlation if one value increases the other value decreases. So they are inversely proportional.

```
correlation = 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
```

<AxesSubplot:>



In The plot above The Negative Correlation have the Negative Values and positive Correlation have Positive Values. In this Particular Case the Values Lies between +1 and -0.6, Plus One means They are positively correlated as the value proceeds towards the negative value it means they are negative correlated. The feature we are interested in is Gold And we see it is positive correlated. we can see the silver column. and the silver, Gold column has the value of

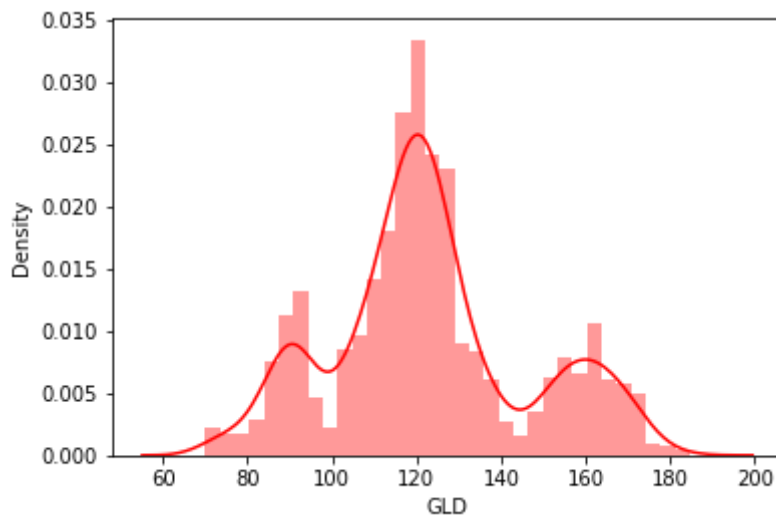
## 0.9 it means they are positively correlated. That Means if Gold prices Increases the Silver

```
# 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(data['GLD'],color='red')
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='GLD', ylabel='Density'>
```



So we Can See Here Most value lies in the range of 120.we have less values around 180,160

## ▼ Splitting the Features and Target

**Split the data to feed this in our machine learning algorithm, so column we are intrested in is Gold So we will be feeding this spx, Uso, Silver,EUR/USD columns with these columns we will try to predict the gold prices so we need to remove Date Column From our Datset.and Separate the Gold Column as well.**

```
X = data.drop(['Date', 'GLD'],axis=1)
Y = data['GLD']
```

```
#printing The X to See That (Date, GLD) Columns are removed or not
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]

```
#printing y For only printing The Gold 'GLD' column
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

we will create four Variable so the "print(x) values Separated into X\_train and X test"  
 The 80% of Values go to X\_Train and remaining 20% of The values will go to X\_test. and The Corresponding Gold Values will go to y\_train and the corresponding gold prices for X\_test will go this y\_test. So we are just splitting the xand y into X\_Train, X\_test, Y\_train, y\_test by using **train\_test\_split function**

```
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: Random Forest Regressor model is an esamble  
 model esamble means it consist of more then one models joined together so it is a non-symbol  
 model of decsion tree

```
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)
```

```
# printing ThePredicted values by our model  
print(test_data_prediction)
```

```
[168.65859942  82.04259985 116.11520038 127.55760053 120.90280122  
154.75619792 150.72199852 126.23050035 117.67429884 126.1030005  
116.70320107 171.86160063 141.37929886 167.98319838 115.15780006  
117.28840038 138.86970262 170.06320148 159.0900034 158.68529927  
155.12800075 125.15750006 175.76789942 157.11870332 125.19450038  
93.79889978 77.88550013 120.86549986 119.18129953 167.52810052  
88.05260048 125.15779981 91.1029005 117.70070009 121.04849881  
136.03000039 115.54310106 115.40480097 148.1765993 107.2694007  
104.22370249 87.07549796 126.40580073 117.83789977 152.91259955  
119.60110006 108.37079978 108.00049819 93.24920086 127.2002979  
75.05810037 113.64409917 121.44269962 111.39799916 118.86459871  
120.5909994 159.50200021 167.92490189 146.98619688 85.9818986  
94.11110043 86.82539919 90.39390046 119.07670071 126.4722005  
127.57050023 169.65870001 122.33599921 117.48789907 98.25629991  
168.1439005 143.02929861 131.51640245 121.14050216 121.70619947  
120.00750054 114.60760172 118.24520044 107.30940086 127.80050026  
114.11599924 107.19719977 116.99560059 119.60489923 88.5911003  
88.22259857 146.46610188 127.22359978 113.54169986 110.27549841  
108.25199911 77.21669887 169.17700147 114.13239927 121.71219918  
127.69690214 154.85689804 91.75779911 135.3808014 159.161603  
125.83390078 125.11820047 130.64270223 114.91930135 119.97200005  
92.20760006 110.1768988 168.12759881 155.99079877 114.27749952  
106.2915014 79.47479976 113.35170055 125.90210066 107.17409945  
119.20260088 156.29650342 160.00459889 120.39890005 134.85970286  
101.47909978 117.52039798 119.25330018 113.06000072 102.79189926  
160.12029859 98.89310038 146.77179936 125.51300075 169.665499  
125.53039954 127.35379754 127.42120154 113.66539974 112.97720074  
123.53869909 102.20829939 88.95919965 124.60329978 102.19089936  
106.98929935 113.7014007 117.3144009 99.12809968 121.90960036  
163.06499838 87.40999881 106.85699963 117.36130036 127.60310099  
124.07220052 80.98529915 120.48030065 156.78739886 87.9197996  
110.35039941 118.86999935 172.27479878 103.092999 105.7897006  
122.56760045 157.27359833 87.69549832 93.39930044 112.95590008  
176.95489964 114.13829994 119.30930011 94.92780116 125.89170035  
165.64790067 114.96850083 116.91950123 88.33379863 148.6257005  
120.24949958 89.50420022 112.02950038 117.11520013 118.81020126  
88.05349947 94.24650007 116.90049988 118.55770187 120.30190055  
126.83839807 121.89199979 150.01280026 165.1947002 118.59669962  
120.20310153 150.74070028 118.46199942 172.99319844 105.39979952  
105.00290092 148.6558006 113.66040029 124.84090111 147.29140019  
119.61160123 115.3884004 112.78310027 113.50700196 142.06400065  
117.77349776 102.91280005 115.86980121 103.58870183 98.8081005  
117.34480054 90.60540003 91.55670036 153.40109873 102.79859988  
154.74040077 114.43170144 138.89980087 90.15719828 115.43829957  
114.29529972 123.17729963 121.79480024 165.36210109 92.93469934
```

135.15630111	121.26759964	120.90240034	104.75200022	141.07770297
121.78109908	116.57380038	113.68660118	127.01229738	122.71889941
125.71929889	121.1644005	86.93129907	132.75780071	145.76990152
92.74059939	158.0257993	158.89480213	126.4729986	164.42489915
108.90179966	110.04510084	103.72969823	94.39470034	127.76840294
107.05720064	161.59109989	121.71550015	131.71790021	130.74670204
160.60150023	90.13169873	174.59960151	127.47010078	126.86179813
86.6188993	124.50109944	150.65879724	89.54820031	106.85269944
108.95229976	84.5604989	135.86959972	154.91870206	139.27230351
73.66460038	152.31080095	126.24299996	126.75090009	127.56169863
108.6183995	156.30360009	114.63240112	116.90230111	125.14319954
153.96830114	121.39019973	156.44819843	93.00190084	125.52860185

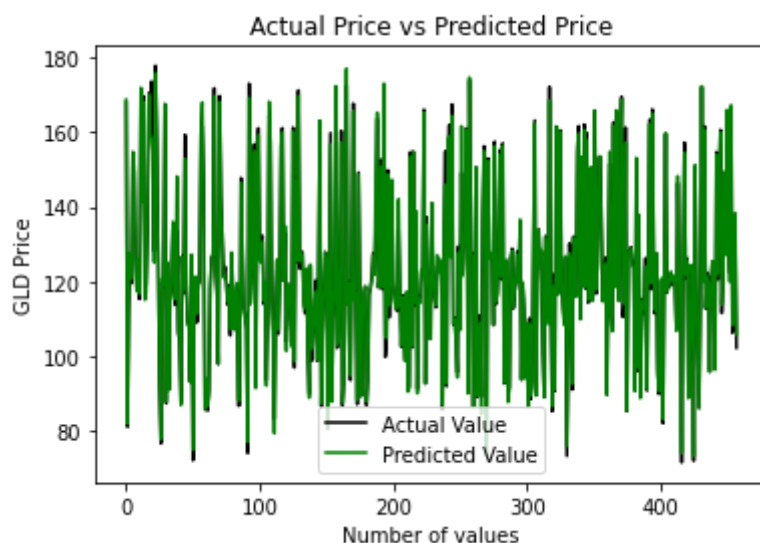
```
# So we need to compare predicted values with actual values by usnig R Square error
# error Score is a range that our model is performing
# R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error Val : ", error_score)
```

R squared error Val : 0.9894290742935247

## Comparing the Actual Values and Predicted Values in a Plot

```
Y_test = list(Y_test)
```

```
#Comparing the actual values with predicted values by labeling it.
# the actual values are labeled with black color and The predicted values are Labeled with G
plt.plot(Y_test, color='black', 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()
```



Thanks :) By Ahmed Ali



