GOLD PRICE PREDICTION MACHINE LEARNING PROJECT USING PYTHON

Name: Darsh Kumar

Data Description

The dataset used in this project contains the price of gold for different dates. The dataset also gives additional information about the price of gold using several other factors such as stock prices, EUR/USD which stands for currency quotation of Euro against US Dollar.

The dataset in total has 6 features which are as under:

*Date - which contains date in the format mm/dd/yyyy
*SPX - it the stock market index of the 500 largest companies listed in the stock market exchange in US.
*GLD - it contains the price of gold.
*USO - which stands for United States Oil Fund
*SLV - which stands for silver price
*EUR/USD - which stands for currency quotation of the Euro against the US.

In [1]:
```
# importing the required library
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
```

Next Step: Data Collection and preprocessing

In [2]:
```
# importing the dataset
df = pd.read_csv('gld_price_data.csv')
```

In [3]:
```
# printing the first five rows of the dataset
df.head()
```

Out[3]:

	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

In [4]:
```
# printing the last five rows of the dataset
df.tail()
```

Out[4]:

	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 [5]:
```
# calculating the number of datapoints in the dataset
df.shape
```

Out[5]:
```
(2290, 6)
```

In [6]:
```
# fecticing general information about the dataset
df.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
```

In [7]:
```
# checking for missing values in the dataset
df.isnull().sum()
```

Out[7]:
```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

In [8]:
```
# getting basic statiscal information about the dataset
df.describe()
```

Out[8]:

	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

In [9]:
```
# finding the features with only one values
for column in df.columns:
    print(column,df[column].nunique())
```

```
Date 2290
SPX 2277
GLD 1930
USO 1514
SLV 1331
EUR/USD 2066
```

In [10]:
```
# exploring the categorical feature
categorical_features = [feature for feature in df.columns if((df[feature].dtypes=
categorical_features
```

Out[10]:
```
['Date']
```

In [11]:
```
for feature in categorical_features:
    print("The name of the feature is {} and the number of categories are {}".for
```

```
The name of the feature is Date and the number of categories are 2290
```

Exploring the Numerical Features

There are two type of numerical features:
1. Continious
2. Discrete

In [12]:
```
numerical_features = [feature for feature in df.columns if((df[feature].dtypes!='
print("The number of numerical features in the dataset is:",len(numerical_feature

# printing the first five numerical features
df[numerical_features].head()
```

```
The number of numerical features in the dataset is: 4
```

Out[12]:

	SPX	USO	SLV	EUR/USD
0	1447.160034	78.470001	15.180	1.471692
1	1447.160034	78.370003	15.285	1.474491
2	1411.630005	77.309998	15.167	1.475492
3	1416.180054	75.500000	15.053	1.468299
4	1390.189941	76.059998	15.590	1.557099

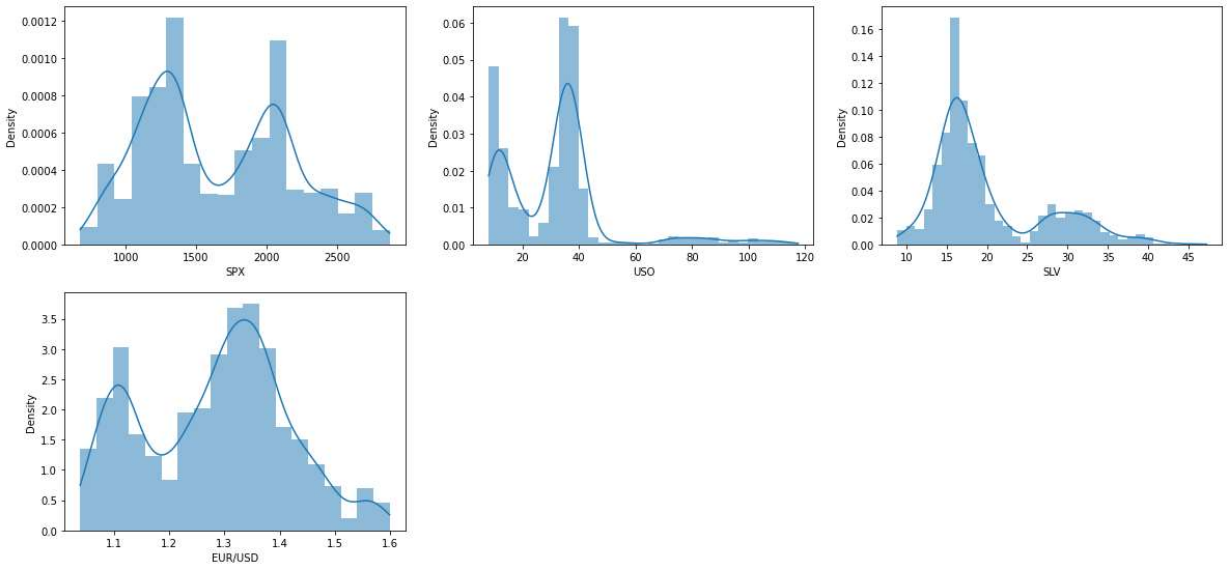
In [13]:
```
# finding the number of discreate numerical feature
discreate_feature = [feature for feature in numerical_features if len(df[feature]
print("The number of Discreate features in the dataset is: {}".format(len(discrea
```

```
The number of Discreate features in the dataset is: 0
```

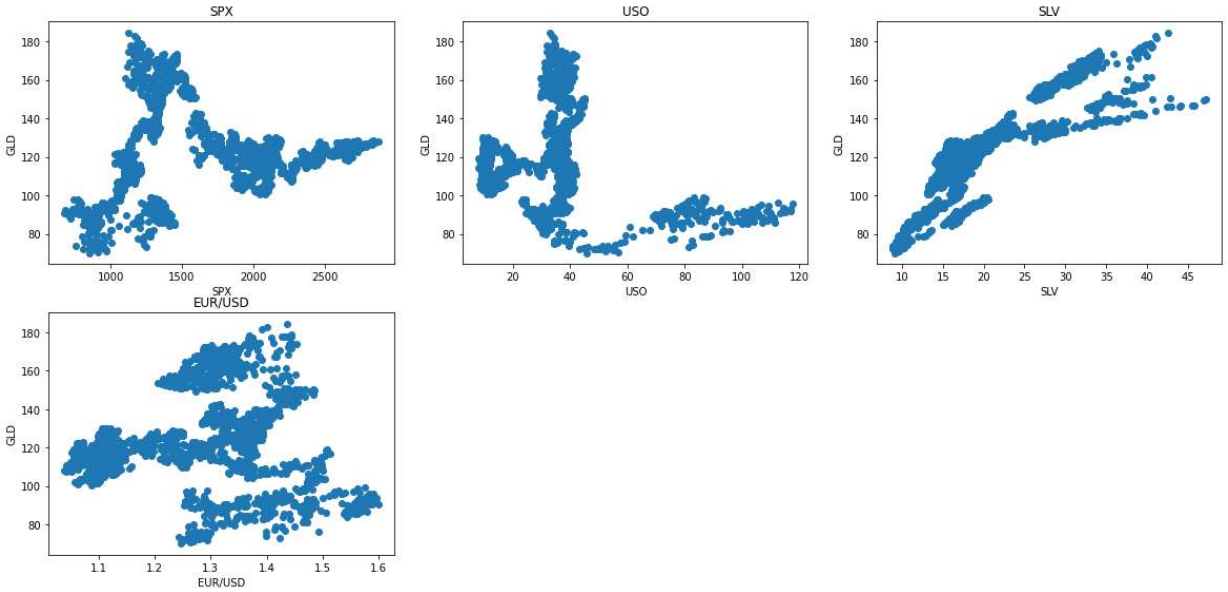
In [14]:
```
# finding the number of continous numerical_feature
conti_features = [feature for feature in numerical_features if feature not in dis
print("The number of continous features in the dataset is: {}".format(len(conti_f
```

```
The number of continous features in the dataset is: 4
```

In [15]:
```
# visualizing the distribution in Continous Numerical Feature
plt.figure(figsize=(20,60),facecolor='white')
plotnumber = 1
for conti_feature in conti_features:
    b = plt.subplot(12,3,plotnumber)
    sns.histplot(df[conti_feature],kde=True,stat='density',linewidth=0)
    plt.xlabel(conti_feature)
    plotnumber+=1
plt.show()
```



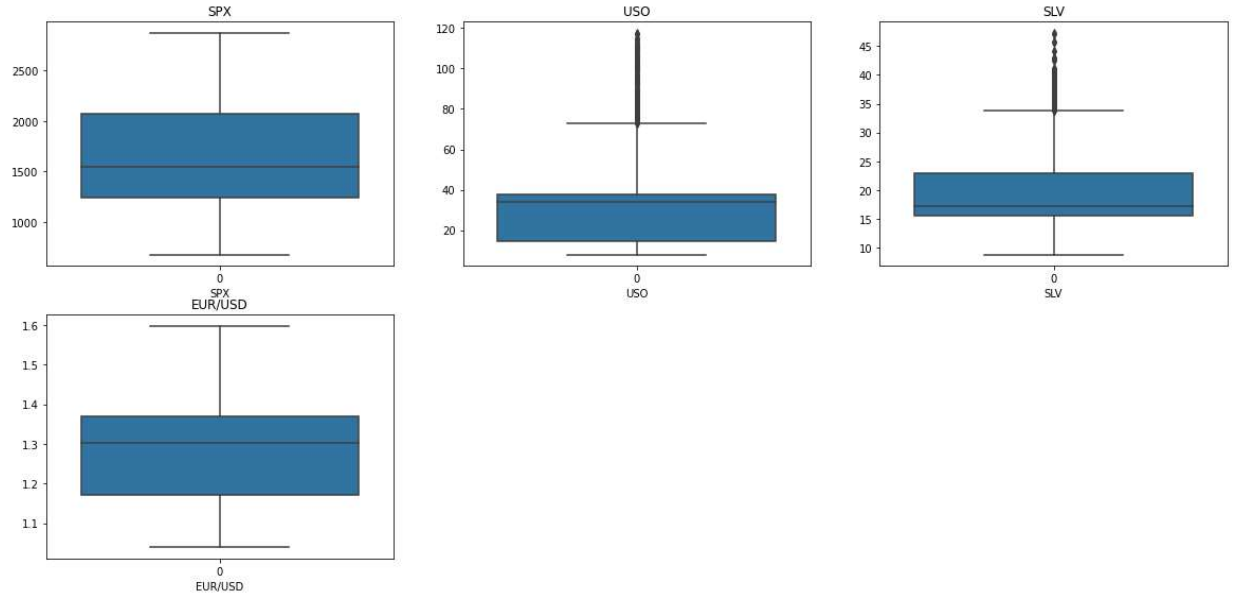
In [16]:
```
# relation between the continous numerical feature and labels
plt.figure(figsize=(20,60),facecolor='white')
plot_number = 1
for feature in conti_features:
    cx = plt.subplot(12,3,plot_number)
    plt.scatter(df[feature],df['GLD'])
    plt.xlabel(feature)
    plt.ylabel('GLD')
    plt.title(feature)
    plot_number+=1
plt.show()
```



Conclusion from above graphs

* The silver feature is linearly progessing with Gold

In [17]:
```
# finding the outiers in the dataset features
plt.figure(figsize=(20,60),facecolor='white')
plot_num = 1
for numerical_feature in numerical_features:
    dx = plt.subplot(12,3,plot_num)
    sns.boxplot(data=df[numerical_feature])
    plt.xlabel(numerical_feature)
    plt.title(numerical_feature)
    plot_num+=1
plt.show()
```



Conclusion of Outlier from above graph

* On visualising the above graphs USD and SPX have Outliers

Next step: Finding correlation between the different features in the dataset

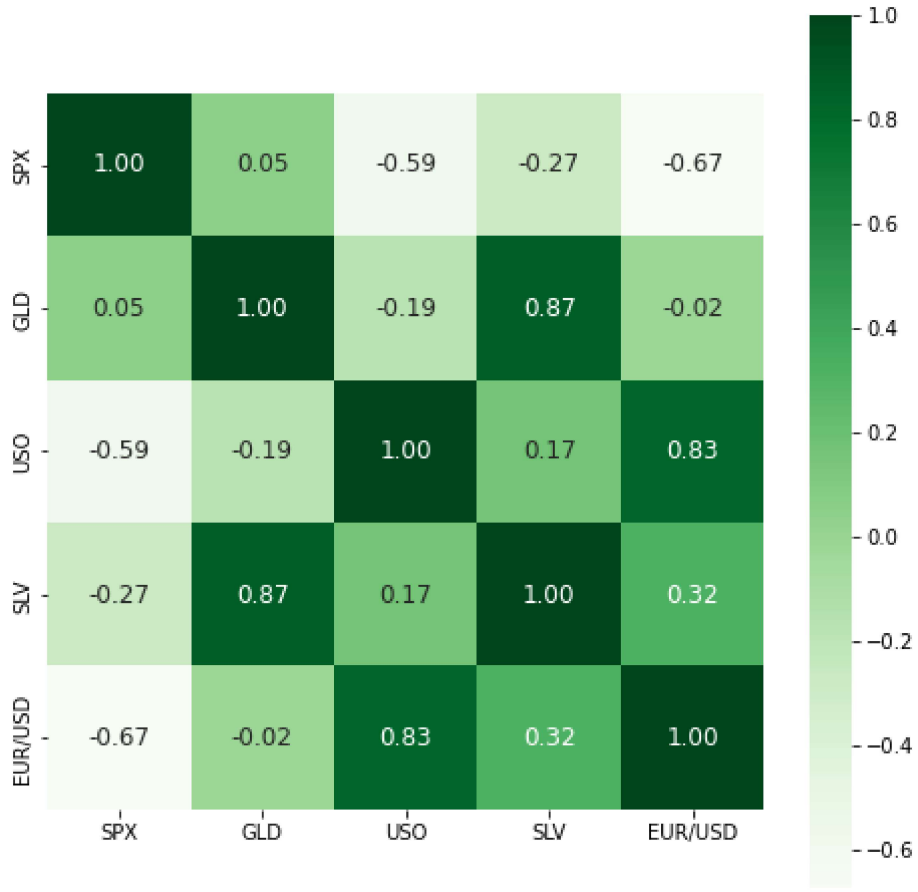
There are two types of correlation:

1. Positive Correlation
2. Negative Correlation

In [18]:
```
# calculating the correlation between the different features in the dataset
correlation = df.corr()
```

In [19]:
```
# constructing a heatmap to understand the correlation
plt.figure(figsize = (8,8))
sns.heatmap(correlation,cbar=True,square=True,fmt='.2f',annot = True,annot_kws={'
```

Out[19]: <AxesSubplot:>

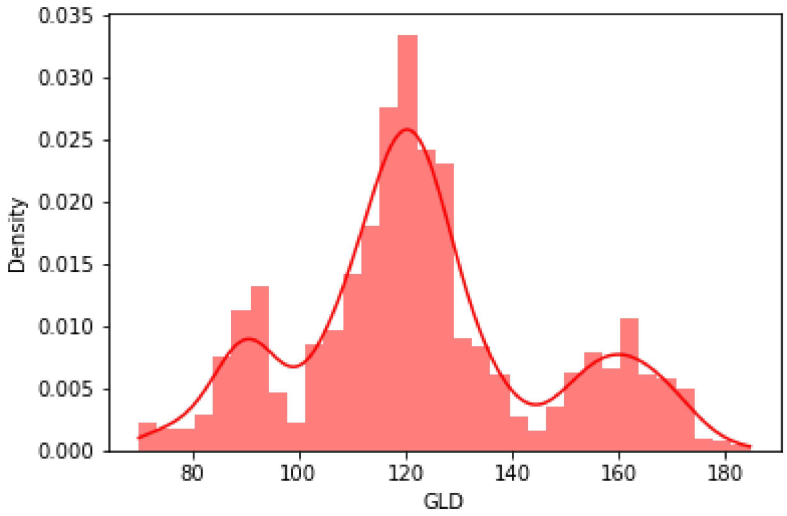


In [20]:
```
# displaying the correlation values for GLD Price
print(correlation['GLD'])
```

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

In [21]:
```
# check the distribution of Gold Price
sns.histplot(df['GLD'],kde=True,stat='density',linewidth=0,color='red')
```

Out[21]: <AxesSubplot:xlabel='GLD', ylabel='Density'>



Next step: Splitting the features and the target

In [22]:
```
x = df.drop(['Date','GLD'],axis = 1)
y = df['GLD']
```

In [23]:
```
# printing the variable x
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]
```

In [24]:
```
# printing the gold prices
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
```

In [25]:
```
# splitting the data into training and test data
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state
```

In [26]:
```
# creating the model and training it
regressor = RandomForestRegressor(n_estimators = 100)
```

In [27]:
```
# training the model
regressor.fit(x_train,y_train)
```

Out[27]:
```
RandomForestRegressor()
```

In [28]:
```
# model evaluation on test data
test_data_x = regressor.predict(x_test)
```

In [29]:
```
# printing the predicted values of x
print(test_data_x)
```

```
[127.10669864 116.77159967 139.39860087 121.88619994  94.18870063
 154.85760092 117.65260111 113.31840128 153.24400058 105.5896006
 103.18539898 120.31310075  92.29209934 159.34420068 121.50279848
 117.5595007   86.85939826  91.82979925  92.8734996  126.05749972
  83.29469939 117.6116995  126.49279893 173.09019724 169.42489762
 139.94430226 114.25039911 159.80360093 133.56560066 115.43030066
 111.3713007  104.79320158 128.01620048 122.10410007  99.2107999
 120.10769973  83.97960024 114.96680039 127.37509874 140.19839844
 107.73710072 131.77100028 108.43349923 135.46150027 133.93769898
 148.77039939 119.63040078 158.12740072 155.94030152 127.67629988
 124.36700035 115.12999837 123.68570065 118.97899999 153.8418002
 121.61139945 112.43120046  86.75519956 155.88630045 118.23470089
  87.93719921 114.76800006  83.08439944 146.56239743 125.20430002
 127.52149945 125.1438995  121.07780008 168.49470144 121.65509932
 125.59430164 173.30039799 121.57970096 169.07460113 119.33850029
 127.43739903 153.77369853 108.14369794 114.48449959 160.1363989
 119.64930047 125.55459823 117.61139938 123.69810022 112.03500056
 160.85070023 114.62409955 113.74980005 125.24749852  96.08240005
 113.76639984 114.11680132  80.81189911 159.63439969 125.69299993
 118.97490097  91.96759981 103.59790171 141.44810211  92.34499859
 154.67790332 122.16560001 123.61219987 115.21579968  87.85260069
 130.89550069 125.57520014 167.49060147 108.91930047 167.11140028
 104.0395995  138.13359838 115.6673014  119.88750058  93.70439993
 166.46360233 113.44970059 109.04639905  82.95650006 109.59629938
 155.55390149  90.69740001  74.84820092 121.95299954 102.87449994
 117.1945989  121.61249997 134.9077001  119.57660007 105.88749972
 106.92329879 112.78859942  93.44419916 122.9259367  119.16199983
 112.27170099  73.55270018 126.72370036  81.9240995  115.3203
 116.78719925 153.24440268 139.02949912 114.53549995 165.77460237
 129.76649978 124.6747002  118.60610122 124.49969924 122.19150044
 161.29189961 106.9842009  117.05139925  87.54329922 123.61020076
 112.2379      85.34299891 113.33949977 117.33520053  86.26189919
 132.02709913 146.0919999  134.51490441 115.89910053 128.09410063
 118.66320072  98.0952009  116.71060108 114.84180101 118.83799876
 155.84710112 122.2943373   90.29400011 120.97590086 105.23549962
 121.53157637 106.29509897 107.10780124 120.83289911 124.95000006
  88.1858986  173.48449946 120.63700148 118.08180119 163.47579873
 119.82770196 110.33869898 128.2406005  131.88850123 120.56119919
 118.54660017 141.8750027  147.9184009  117.07880016 156.71830322
 136.14450022 124.30859992 124.79529873 180.03289726 119.55540219
 162.22370382 104.54610136 132.29740006  92.70829971  97.33139857
  90.32020056 106.01020073 112.64729981 163.15150043 160.74779995
 152.86340319 112.78620145 116.70680143 118.94910172 120.10050051
  91.65540127 102.37750006 154.68629922 108.90319848 102.59219931
 154.74240057  76.0740996  120.3322007  168.70019747 116.24819906
 144.58770148 154.98760026 139.39579961 154.13489868 131.42240356
 129.91149958 124.27000058 116.09030138 135.64259784 163.60320073
 153.96870095 126.98550108 160.57979936 130.23320038 108.76489899
 166.12100036 117.80449833  84.73369935 163.96680274 153.45150041
 116.32980168 129.92880007  92.24279889  91.00740104 112.67579959
  87.7306999  115.44349912 119.08990032 153.48670013 117.00649855
 106.98590122 117.88939948 140.03889911 111.12979997 122.38379944
  83.39229912 125.09220002 125.79210066 167.96140112 127.10349946
```

```
 142.71659831 117.15600162 168.06400016 106.86749861 173.92849984
 109.8778981  115.05950149  88.28809872 125.37670028 117.55070011
 153.23030323 162.2333984   80.20059932 105.05119899 110.67589958
 110.96129893 107.99750058 118.0802009   89.53609981  88.46309922
  84.82470046 137.20529798 127.33060043 119.99370046  96.10620052
 110.06789768  78.45839932  72.70650188 119.33750078 118.67200064
  89.55929987 111.44750128 117.2433     118.36750089 124.47020023
 146.65289873  88.10889968 129.89919808 117.64690083 154.82480194
 108.15259849 144.4913023  124.398599   116.28140041 113.78259873
 125.22509976 135.31730044 141.17999879 117.50659996 128.19219846
  93.7592989  114.32549942  89.19619992  89.36250095 106.26510078
 132.27540189 153.17059931 161.36890193 118.73130038 104.0181996
 100.50930103 150.3725977  152.71609824 106.45100093 137.67610125
 118.79029973 126.86339817  87.73570048 173.66879934 117.67849855
 137.72660187 114.91849942  85.80730001 120.9287998  136.6639012
 117.32680035 138.68559925 118.88830086 165.55890129 121.33440023
 123.06859848 120.98899902 126.77140102  87.79190064 145.775603
 114.13800048 111.45700002 113.35600074 155.86030228  93.24120156
  92.49919971 119.50430083 156.63030236 119.07920035 155.50680301
 126.50470075 161.39459754 123.78939998 131.65330072 123.7694998
  74.62220003  94.42160016  89.9437999  121.08380109 113.44500031
 117.47310021 108.00710042 126.09699989 126.27180194 122.65529998
 120.52479968 103.92449922 109.88719796 110.33320109 149.33730469
 126.01349918 153.40110075 125.78680119 142.72170088 140.92060089
 108.20929888 109.78949804 128.52310265 107.38169907 116.49799932
 120.75150099 145.16300099  93.9329     104.22610028 128.3336016
 124.72329944 163.35839983 121.33820048 148.96400368 110.96199913
 125.15510032  88.99160033  80.60759897 127.3214987  125.94160069
 147.45530282 119.98549902 127.5765979   86.58419938 124.96839965
 118.60559981 115.54359861 102.07299904 121.75060066  93.89619916
 102.85959887 123.78619936 114.86760002 126.51050068  90.07629981
 119.95640081 108.95949893 120.45120008 130.65520138  87.03059888
 123.42119961 101.92199985  99.09560165 102.9214997   89.38539954
 120.65030089 124.01439991 170.50379839 125.75450002  95.48060162
 160.61560479 108.22819975  96.78509934 132.03259919 113.49400044
  87.43279889 168.21859894  89.55769989  90.58999889 167.57589794
 159.69620359 165.84380121 168.78499589 125.71019839 127.3220999
 122.18169966  91.31749937 121.34349975 120.19330072 129.17559847
 160.79469997 116.74669992 108.44589949]
```

In [30]:
```
# compatring the actual values with predicted values using R square error
error_score_x = metrics.r2_score(y_test,test_data_x)
```

In [31]:
```
# printing the r square error
print(error_score_x)
```

```
0.9902289755686619
```

Conclusion of R2 Score is that the R square error is 99.02%

Compare the actual values and predicted values using graphs

In [32]:
```
# converting the y_test to list
y_test = list(y_test)
```

In [33]:
```
# validiating the type of y_test
type(y_test)
```

Out[33]:
```
list
```

In [34]:
```
plt.plot(y_test,color='black',label='Actual Values')
plt.plot(test_data_x,color='orange',label='Predicted Values')
plt.title('Actual Price Vs The Predicted Prices')
plt.xlabel('Number of values')
plt.ylabel("Gold Prices")
plt.legend()
plt.show()
```