PROBLEM STATEMENT: The goal of this project is to analyze the price of gold. The price of gold is volatile, they change rapidly with time. Our main Aim of this project will be to predict the price of gold per unit.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
sns.set_style("darkgrid", {"grid.color": ".6",
                           "grid.linestyle": ":"})

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

# read dataset using pndas function
dataset = pd.read_csv("gold_price_data.csv")

dataset.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

```
dataset.tail()

dataset.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	1.303297
75%	2073.010070	132.840004	37.827501	22.882500	1.369971
max	2872.870117	184.589996	117.480003	47.259998	1.598798

```
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
```

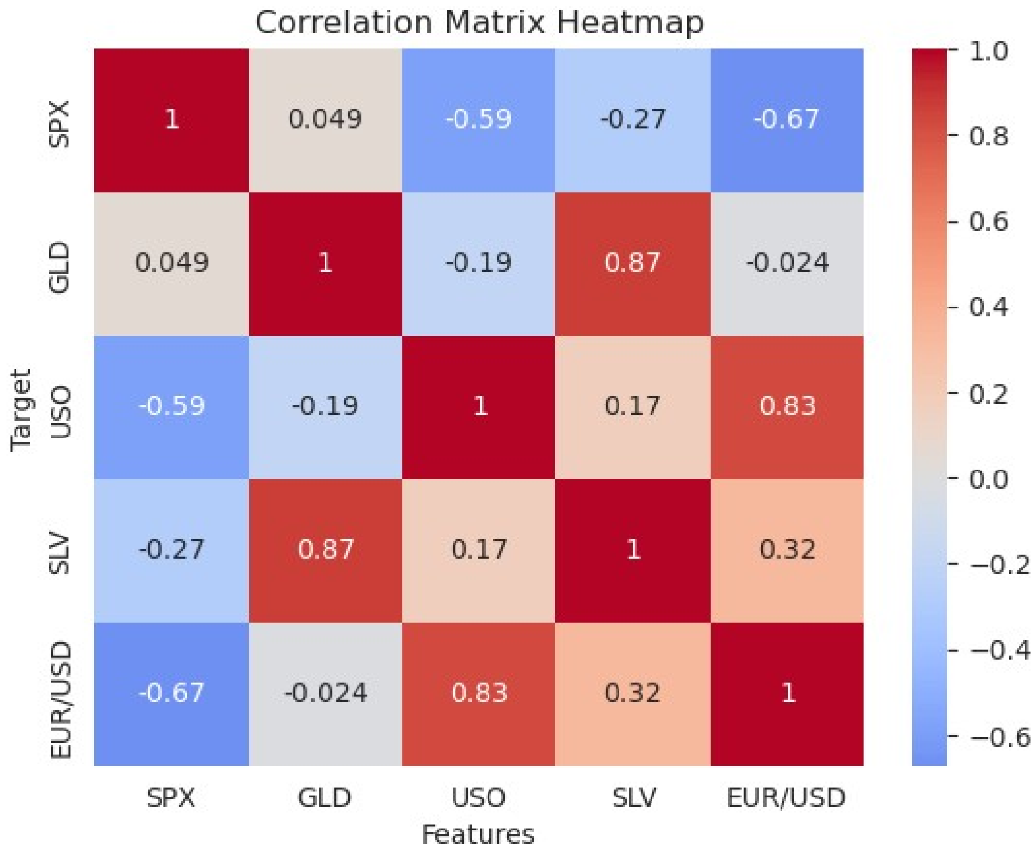
```
dataset.duplicated().sum()
```

```
0
```

```
# Missing Values/Null Values Count
dataset.isna().sum()
```

```
Date       0
SPX        0
GLD        0
USO        0
SLV        0
EUR/USD    0
dtype: int64
```

```
# Calculate correlation matrix
correlation  = dataset.corr()

# Create heatmap
sns.heatmap(correlation, cmap='coolwarm',center=0, annot=True)

# Set title and axis labels
plt.title('Correlation Matrix Heatmap')
plt.xlabel('Features')
plt.ylabel('Target')

# Show plot
plt.show()
```

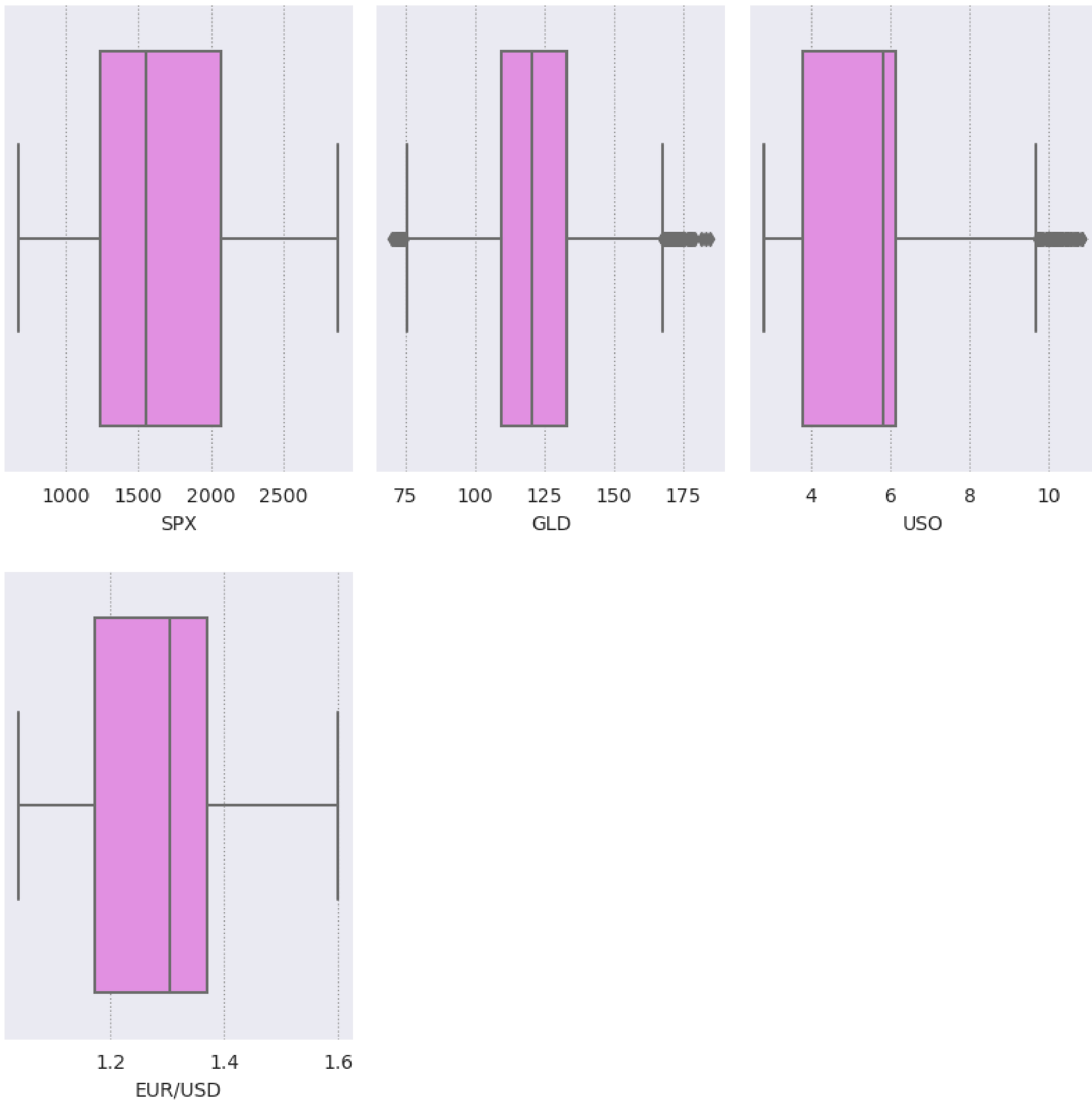



```
# drop SlV column
dataset.drop("SLV", axis=1,inplace=True)

# plot price of gold for each increasing year
dataset["EUR/USD"].plot()
plt.title("Change in price of gold through year")
plt.xlabel("year")
plt.ylabel("price")
plt.show()
```


```python
# skewness along the index axis
print(dataset.skew(axis=0, skipna=True))
```

```
SPX        0.300362
GLD        0.334138
USO        1.699331
EUR/USD   -0.005292
dtype: float64
```

```python
# apply square root transformation on the skewed dataset
dataset["USO"] = dataset["USO"].apply(lambda x: np.sqrt(x))

# handling Outliers
#Plotting Boxplot to Visualize the Outliers
fig = plt.figure(figsize=(8, 8))
temp = dataset.drop("Date", axis=1).columns.tolist()
for i, item in enumerate(temp):
    plt.subplot(2, 3, i+1)
    sns.boxplot(data=dataset, x=item, color='violet')
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=2.0)
plt.show()
```




```
def outlier_removal(column):
    # Capping the outlier rows with Percentiles
    upper_limit = column.quantile(.95)
    # set upper limit to 95percentile
    lower_limit = column.quantile(.05)
    # set lower limit to 5 percentile
    column.loc[(column > upper_limit)] = upper_limit
    column.loc[(column < lower_limit)] = lower_limit
    return column

# Normalize outliers in columns except Date
```

```python
dataset[['SPX', 'GLD', 'USO', 'EUR/USD']] = \
    dataset[['SPX', 'GLD', 'USO', 'EUR/USD']].apply(outlier_removal)

# select the features and target variable
X = dataset.drop(['Date','EUR/USD'], axis=1)

y = dataset['EUR/USD']
# dividing dataset in to train test
x_train, x_test,y_train, y_test = train_test_split(X, y,test_size=0.2)

#scaling the data
# Create an instance of the StandardScaler
scaler = StandardScaler()

# Fit the StandardScaler on the training dataset
scaler.fit(x_train)

# Transform the training dataset using the StandardScaler
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Create a lr object
lr = LinearRegression()

# Fit the object to the training data
lr.fit(x_train_scaled, y_train)

# Predict the target variable using the fitted model
y_pred = lr.predict(x_train_scaled)

# Compute the R-squared of the fitted model on the train  data
r2 = r2_score(y_train, y_pred)

# Print the R-squared
print("R-squared: ", r2)
```

```
R-squared:  0.786488439393935
```

```python
# create instance of the Randomforest regressor
rf = RandomForestRegressor()

# Fit the object to the training data
rf.fit(x_train_scaled, y_train)

# Insiate param grid for which to search
param_grid = {'n_estimators': [50, 80, 100],'max_depth': [3, 5, 7]}

# Define Girdsearch with random forest
# object parameter grid scoring and cv
rf_grid_search = GridSearchCV(rf, param_grid, scoring='r2', cv=2)
```

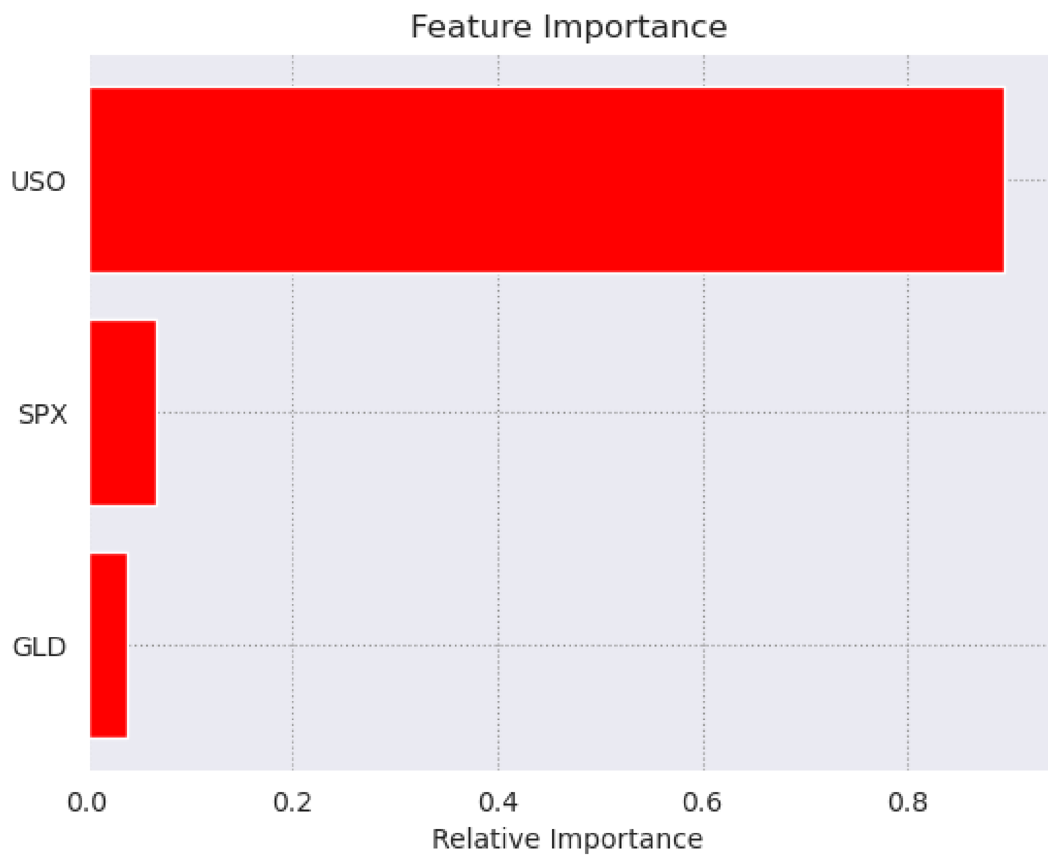
```python
# Fit the GridSearchCV object to the training data

rf_grid_search.fit(x_train_scaled, y_train)
```

```
GridSearchCV(cv=2, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [3, 5, 7], 'n_estimators': [50,
80, 100]},
             scoring='r2')
```

```python
# Compute the R-squared of the fitted model on the test data
r2 = r2_score(y_test, rf.predict(x_test_scaled))
print('Best parameter values: ', rf_grid_search.best_params_)
print("R-squared:", r2)
```

```
Best parameter values:  {'max_depth': 7, 'n_estimators': 100}
R-squared: 0.9600049668776899
```

```python
features = dataset.drop("Date", axis=1).columns

# store the importance of the feature
importances = rf_grid_search.best_estimator_.feature_importances_

indices = np.argsort(importances)

# title of the graph
plt.title('Feature Importance')

plt.barh(range(len(indices)),importances[indices],color='red',align='c
enter')

# plot bar chart
plt.yticks(range(len(indices)),[features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```




```python
# Create an instance of the XGBRegressor model
model_xgb = XGBRegressor()

# Fit the model to the training data
model_xgb.fit(x_train_scaled, y_train)

# Print the R-squared score on the training data
print("Xgboost Accuracy =", r2_score(y_train,
model_xgb.predict(x_train_scaled)))
```

```
Xgboost Accuracy = 0.998311248292406
```

```python
# Print the R-squared score on the test data
print("Xgboost Accuracy on test data =",
      r2_score(y_test,model_xgb.predict(x_test_scaled)))
```

```
Xgboost Accuracy on test data = 0.9614063553048445
```

Conclusion: This feature importance graph shows that USO column plays a major effect (more than 2x) in deciding the gold price in USD.