Gold Prediction Model

I aim to predict gold prices using machine learning. Utilizing Python libraries like NumPy, Pandas, Matplotlib, Seaborn, and Scikit-learn, we perform data analysis and model training. We use a RandomForestRegressor to develop an accurate prediction model based on various economic indicators and market data. The workflow includes data preprocessing, exploratory data analysis (EDA), feature engineering, model training, and performance evaluation, ensuring a comprehensive approach to developing a reliable gold prediction model.

In [1]:
```
import numpy as np
importpandas as pd
importmatplotlib.pyplot as plt
importseaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
```

Data Preprocessing

In [4]:
```
df = pd.read_csv('C:\\Users\\YOURPATH\gld_price_data.csv')
```

In [5]:
```
print(df.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
```

In [6]:
```
df.tail()
```

Out[6]:

	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 [7]:
```
df.shape #No of Columns and Row
```

Out[7]:
```
(2290, 6)
```

In [9]:
```
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 [10]:
```
#checking if Null Values
df.isnull().sum()
```

Out[10]:
```
Date SPX   0
GLD        0
USO        0
SLV        0
EUR/USD    0
           0
dtype: int64
```

In [11]:
```
df.describe()
```

Out[11]:

	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

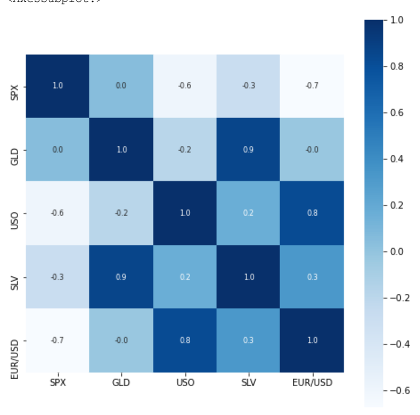
Exploratory Data Analysis

Correlation

In [13]:
```
correlation = df.corr()
```

In [14]:
```
# Constructing a heatmap
plt.figure(figsize = (8,8))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f',annot=True, annot_kws={'size':8}, cmap='Blues')
```

Out[14]: <AxesSubplot:>

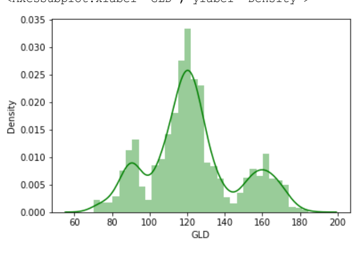


In [15]:
```
print(correlation['GLD'])
```

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

In [18]:
```
sns.distplot(df['GLD'],color='green')
```

Out[18]: <AxesSubplot:xlabel='GLD', ylabel='Density'>



This distribution plot helps us observe the spread and central tendency of gold prices, providing insights into the data's skewness, kurtosis, and potential outliers.

Time-Series Analysis

In [32]:
```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

In [40]:
```
df = df.asfreq('D')
```

In [33]:
```
df['Lag_1'] = df['GLD'].shift(1)
df['Lag_2'] = df['GLD'].shift(2)
```

In [34]:
```
df['Rolling_Mean_7'] = df['GLD'].rolling(window=7).mean()
df['Rolling_Std_7'] = df['GLD'].rolling(window=7).std()
```

In [35]:
```
df.dropna(inplace=True)
```

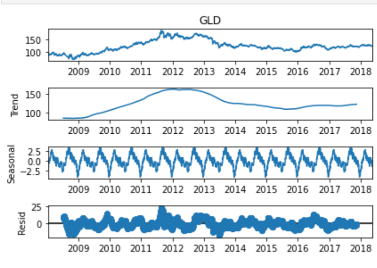
In [36]:
```
X = df.drop('GLD', axis=1)
Y = df['GLD']
```

In [37]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

In [38]:
```
import statsmodels.api as sm
```

In [42]:
```
df['GLD'] = df['GLD'].interpolate()
```

In [43]:
```
decomposition = sm.tsa.seasonal_decompose(df['GLD'], model='additive', period=365)
decomposition.plot()
plt.show()
```



Data Splitting

In [20]:
```
X = df.drop(['Date','GLD'],axis=1)
Y = df['GLD']
```

In [21]:
```
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 [22]:
```
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
```

Model Training

In [23]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=2)
```

In [24]:
```
regressor = RandomForestRegressor(n_estimators=100)
```

In [26]:
```
regressor.fit(X_train,Y_train)
```

Out[26]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

Model Evaluation

In [27]:
```
test_data_prediction = regressor.predict(X_test)
```

In [28]:
```
print(test_data_prediction)
```

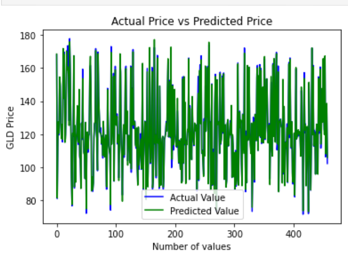
```
[168.43489927 81.65429996 116.16480036 127.58930087 120.67050182
 154.61369777 150.43639826 126.19440041 117.53049873 125.90340092
 116.87530133 171.757301   141.90719881 168.13279935 115.09510011
 117.67160031 138.81830307 170.06770041 158.96580327 160.62059915
 155.18350033 125.30819985 175.79189998 157.24350299 125.22840073
  93.85159966 77.54999976 120.68720001 119.1041991 167.38129975
  88.31020045 125.31510007 91.11040101 117.71130021 121.13599958
 137.22100023 115.52220136 115.11470083 147.09360027 107.16760101
 104.41120227 87.22239808 126.55110037 117.83890054 153.66629902
 119.57730021 108.34879981 107.93549838 93.2384005 127.19729757
  75.06190053 113.51949876 121.45140011 111.17999918 118.85139918
 120.42569963 159.09099999 167.03230131 146.8764964   85.90129849
  94.28580041 86.77269891 90.48749976 118.94020068 126.47540081
 127.65480014 170.46229948 122.31189892 117.38369899 98.51050016
 168.5887012 142.86649834 132.42990196 121.11590218 120.70949962
 119.71090063 114.47510211 118.18120048 107.12590103 127.82460066
 114.0567997 106.36540017 116.77160054 119.50429883 88.60600051
  88.23379874 146.85240263 127.31900085 113.47800002 109.87059852
 107.96889892 77.50609909 169.39150164 114.00799912 121.69019912
 127.96600095 154.88649807 91.89599953 136.664601    158.9525036
 125.71920056 125.41100096 130.73620089 114.85320149 119.71749998
  92.04309999 110.30059871 168.57929844 156.73459976 114.23909964
 106.8019011    79.59719982 113.22250055 125.80200067 107.57089957
 119.3275015 155.56940354 159.64179962 120.18149987 134.11970285
119133862900287112796989087 102.71369906
 160.27589821 99.14170027 147.46709904 125.64170096 169.68839928
 125.77999878 127.31659763 127.48720195 113.69449962 112.68200067
 123.74069917 102.10439907 89.29509988 124.48109969 101.45049931
 106.96969964 113.3670008 117.11170064 99.1709993 121.96060023
 162.82039892 87.2796985 106.78869945 117.08160101 127.61260138
 124.17490081 80.83789911 120.52810084 157.70119922 88.2493993
 110.21849954 118.97049883 172.33359915 102.97099904 105.49830048
 122.5654005 157.90659802 87.84859818 93.38660076 112.88770029
 177.13229984 114.72689979 119.2866003    94.83800095 125.80590001
 166.08480146 114.74670068 116.71750146 88.23759867 149.02410031
 120.24949972 89.29959986 111.89340023 117.23220042 118.78550116
88.10689933 94.34130009 117.07820047 118.44110216 120.29179995
 126.52609871 121.86669984 147.90410089 165.58970019 118.46649971
 120.31250155 151.65440046 118.58159897 172.59789845 105.9682996
 105.00040114 149.31040051 113.60340108 124.85760121 147.1492003
 119.54810141 115.49190049 112.53990014 113.42200188 141.71510123
 117.85269757 102.95770028 115.84550083 103.82400192 98.85320007
 117.31330072 90.67789992 91.48990047 153.43039956 102.74019968
 154.73420097 114.29530147 139.30610051 90.26179816 115.56949921
 113.76099974 122.99120047 121.74840002 165.54130139 92.86269972
 135.57470167 121.375299    120.78320087 104.67880025 142.79120295
 121.42059949 116.66200037 113.6996009 126.81259852 122.14849952
 125.6753993 121.21980021 86.94499914 132.48010067 144.63200177
  92.75959952 158.23089985 158.73120233 126.41419908 164.99199908
 108.74529942 110.03960093 103.81849897 94.54730025 127.60140265
 106.93130064 161.21620047 121.99400002 131.65450012 130.54080096
 160.18710031 90.15569863 175.57460229 128.08870008 126.67299865
  86.2762988 124.55339969 150.03039739 89.65580031 107.08999943
 108.98579999 84.11639943 136.37740014 154.8651024 138.49300381
  74.26730022 152.74910125 125.98490047 126.75980001 127.43289868
 108.54459948 156.45379941 114.58410124 116.91580124 125.15579989
 154.09340126 121.51480011 156.43369919 92.89160039 125.46510137
 125.79280047 88.09860057 92.00859938 126.12189941 128.84280391
 113.09820065 117.43859778 120.9963002 126.74479906 119.65530112
 137.43490085 94.10039954 119.81960078 112.88600114 94.30269944
 108.85249979 87.0972993 109.2310995     89.75439996 92.42800022
 131.42300277 162.43080089 89.24939973 119.70510095 133.39990185
 123.92020021 128.6069024 101.8633984    88.90979869 131.90540001
 119.53580005 108.51960008 168.96960151 115.30720031 86.69539895
 118.84470054 90.94319957 161.5415       116.33920039 121.61270009
 160.40999821 120.13249933 112.60439957 108.34159885 126.67939961
  75.9350004 103.02349978 127.4880021 121.89489907 92.58719972
 131.9362008 118.02890113 116.25009957 154.2915026 159.98860091
 110.25509947 155.34169808 119.25090072 160.75020146 118.72860011
 158.45189858 115.12599874 116.60270027 148.51349961 114.76360094
 125.36419886 166.91239916 117.54869988 124.94839949 153.21920348
 153.44100277 131.96479981 114.75390073 121.25890214 125.43460082
  89.61280045 123.13579968 154.84700136 111.61190041 106.76429977
 161.81730107 118.65129938 165.68549969 134.0720012 115.25689971
 152.99139922 168.65009964 114.81450018 114.10350136 157.66139881
  85.30209888 127.03520058 127.96110035 128.96580073 124.39260048
 124.07520085 90.57280065 153.16079984 97.13049992 136.81329987
  88.91629942 106.27120011 115.05410067 112.72370087 123.98309891
  91.33889865 125.40700132 162.37689876 120.00709879 165.21860121
 126.55479882 112.28920028 127.55719933 94.80619885 91.08379987
 103.25279891 120.7567        83.1977997 126.4173994 160.25190473
 117.20500078 118.3270997 120.1246999 122.78280005 120.00940123
 121.56509972 117.91260048 106.73350012 148.50449992 126.31679864
 115.80820092 73.85229966 127.88610106 153.71820043 122.61100008
 125.59310016 88.87540007 103.70809897 124.75720054 120.24260027
73.30350081 151.38850049 121.28860034 104.64200001 86.38419793
 115.16599915 172.11499832 119.71960041 160.70469783 113.20849971
 121.01790006 118.58310117 95.92949985 118.55760009 126.11640031
 118.71099993 95.88570051 153.72820154 122.1121002 147.21870018
 159.24160266 114.07690043 122.57489941 147.67009755 127.3475002
 165.66320058 135.25650069 119.84289937 167.28909909 108.15599966
 121.83749862 138.65980004 106.0272991 ]
```

In [29]:
```
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error : ", error_score)
```

```
R squared error : 0.9897226125320989
```

In [30]:
```
Y_test = list(Y_test)
```

In [31]:
```
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()
```



Interactive Line Plot for Actual vs Predicted Prices

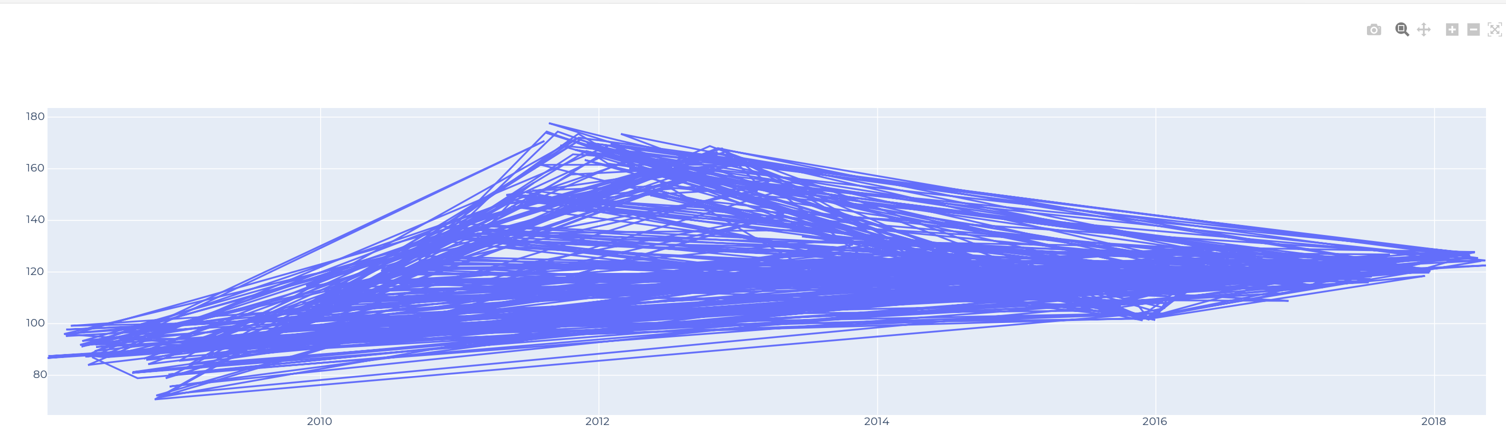
In [49]:
```
import plotly.graph_objects as go
```

In [50]:
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100)
regressor.fit(X_train, Y_train)
```

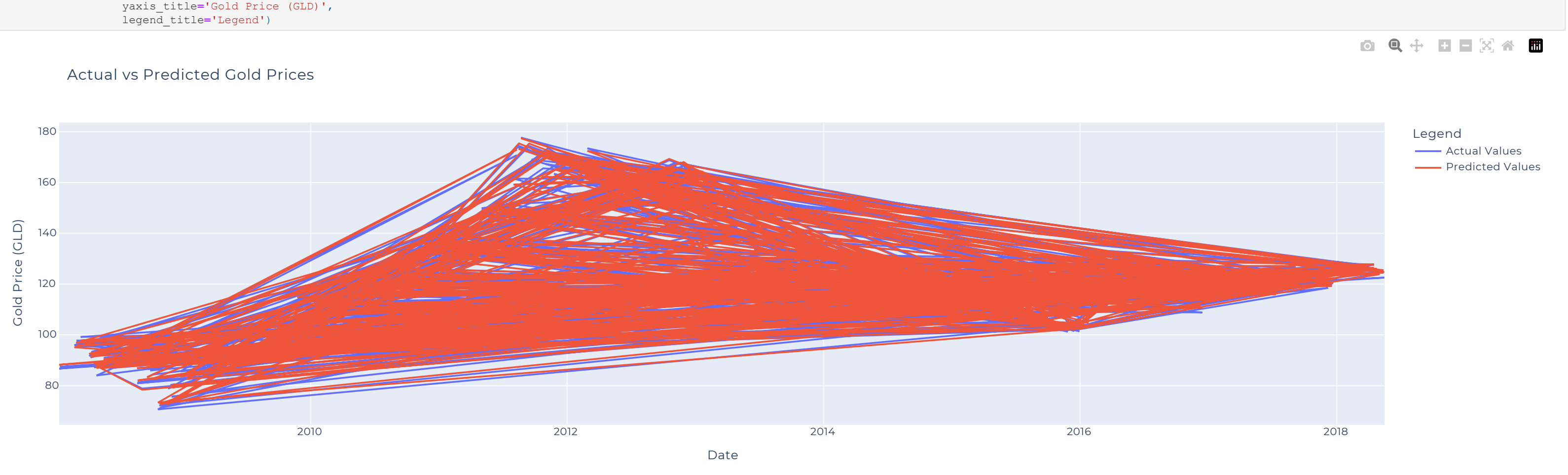
Out[50]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [51]:
```
test_data_prediction = regressor.predict(X_test)
# Create interactive plot

fig = go.Figure()
```

In [52]:
```
fig.add_trace(go.Scatter(x=Y_test.index, y=Y_test, mode='lines', name='Actual Values'))
```



In [53]:
```
fig.add_trace(go.Scatter(x=Y_test.index, y=test_data_prediction, mode='lines', name='Predicted Values'))
# Update layout

fig.update_layout(title='Actual vs Predicted Gold Prices',
                  xaxis_title='Date',
                  yaxis_title='Gold Price (GLD)',
                  legend_title='Legend')
```



Interactive Plot for Time-Series Decomposition

In [55]:
```
import plotly.subplots as sp
```

In [57]:
```
fig = sp.make_subplots(rows=4, cols=1, subplot_titles=('Observed', 'Trend', 'Seasonal', 'Residual'))

fig.add_trace(go.Scatter(x=decomposition.observed.index, y=decomposition.observed, mode='lines', name='Observed'),
             row=1, col=1)
fig.add_trace(go.Scatter(x=decomposition.seasonal.index, y=decomposition.seasonal, mode='lines', name='Seasonal'),
             row=3, col=1)
fig.add_trace(go.Scatter(x=decomposition.resid.index, y=decomposition.resid, mode='lines', name='Residual'),
             row=4, col=1)
fig.update_layout(title='Time Series Decomposition',
                  xaxis_title='Date',
                  yaxis_title='Value',
                  legend_title='Components')
# Show the plot

fig.show()
```