goldpricemodel

April 9, 2024

1 Gold price prediction using regression techniques

1.1 importing essential libraries

[85]:
```
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.ensemble import RandomForestRegressor, AdaBoostRegressor ,␣
 ↪GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
import pickle
```

1.2 Data Processing

[86]:
```
gold_data = pd.read_csv('golddaily.csv')                    #reading data from␣
 ↪dataset.csv
```

[3]:
```
gold_data.head()
```

[3]:

	Date	Price	Open	High	Low	Chg%
0	01-Jan-24	63320	63225	63379	63181	0.19%
1	29-Dec-23	63203	63246	63385	63051	-0.29%
2	28-Dec-23	63389	63728	63821	63333	-0.45%
3	27-Dec-23	63678	63198	63710	63179	1.04%
4	26-Dec-23	63025	63149	63198	62903	0.11%

[87]:
```
# gold_data['Price'] = gold_data['Price'].int.replace(',', '').astype(float)
# gold_data['Open'] = gold_data['Open'].int.replace(',', '').astype(float)
# gold_data['High'] = gold_data['High'].int.replace(',', '').astype(float)
```

```
# gold_data['Low'] = gold_data['Low'].int.replace(',', '').astype(float)
gold_data['Chg%'] = gold_data['Chg%'].str.rstrip('%').astype(float)
 ↪          #changing datatype of change from % object to float64
```

[88]:
```
# Convert 'Date' column to datetime format with specified format
gold_data['Date'] = pd.to_datetime(gold_data['Date'], format='%d-%b-%y')
gold_data['Date'] = pd.to_datetime(gold_data['Date'], format='%b-%y')

# Extract Year, Month, and Day
gold_data['Year'] = gold_data['Date'].dt.year
gold_data['Month'] = gold_data['Date'].dt.month
gold_data['Day'] = gold_data['Date'].dt.day

# Drop the original 'Date' column
gold_data.drop('Date', axis=1, inplace=True)
```

[89]:
```
## Adding some noise for overcoming overfitting
synthetic_data = pd.DataFrame()

for column in gold_data.columns:
    # Generate synthetic data for each column
    synthetic_column = np.random.choice(gold_data[column], size=100)
    synthetic_data[column] = synthetic_column
```

[90]:
```
merged_data = pd.concat([gold_data, synthetic_data])
```

[91]:
```
merged_data
```

[91]:
```
    Price   Open   High    Low  Chg%  Year  Month  Day
0   63320  63225  63379  63181  0.19  2024      1    1
1   63203  63246  63385  63051 -0.29  2023     12   29
2   63389  63728  63821  63333 -0.45  2023     12   28
3   63678  63198  63710  63179  1.04  2023     12   27
4   63025  63149  63198  62903  0.11  2023     12   26
..    …      …      …      …     …    …      …    …
95  26892  50275  30036  29739  0.06  2022     10   16
96  28541  46851  27274  26350 -1.43  2022      1   18
97  29012  34465  51065  26911 -1.21  2019      7    7
98  27198  46766  32319  31875 -0.59  2021      9   11
99  29487  30930  29105  45479  0.82  2015     11    5

[2692 rows x 8 columns]
```

[9]:
```
gold_data.describe()
```

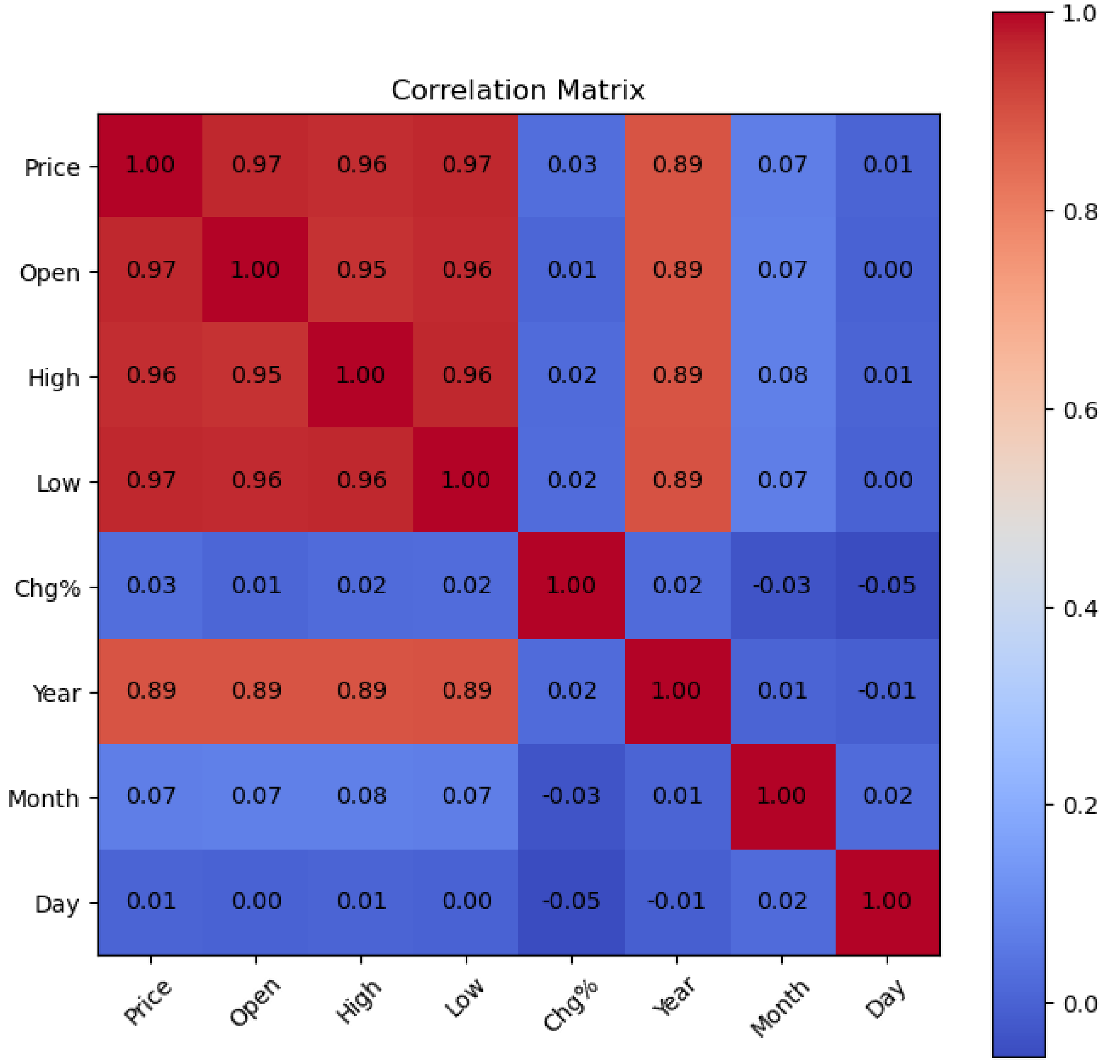
[9]:
```
             Price         Open         High          Low         Chg%  \
count  2592.000000  2592.000000  2592.000000  2592.000000  2592.000000
```

```
mean    38255.199460  38249.878472  38452.240355  38053.575617      0.034904
std     11337.916616  11337.067365  11403.420055  11272.424903      0.847437
min     24597.000000  24535.000000  24701.000000  24451.000000     -8.670000
25%     28788.750000  28796.500000  28910.250000  28667.750000     -0.410000
50%     31638.000000  31633.500000  31827.000000  31513.500000      0.040000
75%     48681.000000  48691.000000  48925.750000  48465.750000      0.480000
max     63678.000000  63728.000000  64460.000000  63333.000000      5.670000

              Year         Month          Day
count  2592.000000  2592.000000  2592.000000
mean   2018.475309     6.513889    15.719136
std       2.884796     3.447545     8.792376
min    2014.000000     1.000000     1.000000
25%    2016.000000     4.000000     8.000000
50%    2018.000000     7.000000    16.000000
75%    2021.000000     9.250000    23.000000
max    2024.000000    12.000000    31.000000
```

1.3 Data Extraction

[92]:

```python
correlation = merged_data.corr()

plt.figure(figsize=(8, 8))
plt.title('Correlation Matrix')
plt.imshow(correlation, cmap='coolwarm', interpolation='nearest')

# Add text annotations
for i in range(len(correlation)):
    for j in range(len(correlation)):
        plt.text(j, i, f'{correlation.iloc[i, j]:.2f}', ha='center',␣
 ↪va='center', color='black')

plt.colorbar()
plt.xticks(ticks=np.arange(len(correlation.columns)), labels=correlation.
 ↪columns, rotation=45)
plt.yticks(ticks=np.arange(len(correlation.columns)), labels=correlation.
 ↪columns)
plt.show()
```



[11]:
```
print(correlation)
```

```
          Price      Open      High       Low      Chg%      Year     Month  \
Price  1.000000  0.965637  0.958791  0.961114  0.026571  0.891795  0.075933
Open   0.965637  1.000000  0.959905  0.952252  0.002116  0.884415  0.072420
High   0.958791  0.959905  1.000000  0.960012  0.021382  0.883587  0.069189
Low    0.961114  0.952252  0.960012  1.000000  0.018365  0.887666  0.070271
Chg%   0.026571  0.002116  0.021382  0.018365  1.000000  0.024354 -0.034528
Year   0.891795  0.884415  0.883587  0.887666  0.024354  1.000000  0.003376
Month  0.075933  0.072420  0.069189  0.070271 -0.034528  0.003376  1.000000
Day    0.005627  0.009561  0.005068  0.001848 -0.054451  0.002105  0.017166

            Day
Price  0.005627
```

```
Open     0.009561
High     0.005068
Low      0.001848
Chg%    -0.054451
Year     0.002105
Month    0.017166
Day      1.000000
```

[12]:
```
gold_data.isnull().sum()
```

[12]:
```
Price     0
Open      0
High      0
Low       0
Chg%      0
Year      0
Month     0
Day       0
dtype: int64
```

[13]:
```
price_column = gold_data.pop('Price')
gold_data['Price'] = price_column
```

[93]:
```
merged_data.isnull().sum()
```

[93]:
```
Price     0
Open      0
High      0
Low       0
Chg%      0
Year      0
Month     0
Day       0
dtype: int64
```

[15]:
```
gold_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2592 entries, 0 to 2591
Data columns (total 8 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Open    2592 non-null   int64
 1   High    2592 non-null   int64
 2   Low     2592 non-null   int64
 3   Chg%    2592 non-null   float64
 4   Year    2592 non-null   int32
 5   Month   2592 non-null   int32
```

```
 6   Day      2592 non-null   int32
 7   Price    2592 non-null   int64
dtypes: float64(1), int32(3), int64(4)
memory usage: 131.8 KB
```

[94]:
```python
X = merged_data.drop('Price', axis=1)
Y = merged_data['Price']
```

[95]:
```python
X.tail()
```

[95]:

	Open	High	Low	Chg%	Year	Month	Day
95	50275	30036	29739	0.06	2022	10	16
96	46851	27274	26350	-1.43	2022	1	18
97	34465	51065	26911	-1.21	2019	7	7
98	46766	32319	31875	-0.59	2021	9	11
99	30930	29105	45479	0.82	2015	11	5

[96]:
```python
from sklearn.model_selection import train_test_split
```

[97]:
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=42)
```

1.4 Standardisation

[20]:
```python
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

1.5 LINEAR REGRESSION

[21]:
```python
linear_regressor = LinearRegression()
linear_regressor.fit(X_train_scaled, Y_train)
```

[21]:
```
LinearRegression()
```

[22]:
```python
test_data_prediction = linear_regressor.predict(X_test_scaled)
```

[23]:
```python
test_data_prediction
```

[23]:
```
array([28977.01380179, 26919.40070297, 51176.7656607 , 47194.70528544,
       28164.12487887, 30829.87416499, 37929.46833345, 27285.98801479,
       29581.99062323, 29271.17830187, 29684.6454153 , 29127.12685534,
       28011.25937088, 58321.69757268, 30944.27604551, 38565.88608774,
       28717.08114202, 28206.26733439, 51380.23354289, 26476.92027073,
       49949.88985957, 46577.46519137, 26813.85161442, 33820.74040148,
       47278.93004689, 49321.10689966, 46738.59537691, 28860.80959624,
```

26330.37028419, 48900.61578841, 28891.43791687, 26765.89209176,
50212.32941128, 54651.83618952, 46476.4070778 , 49927.62270547,
59507.64328959, 45291.25199343, 30131.27539049, 27196.90386334,
31782.23504204, 29418.04051765, 48694.85743548, 51280.6097343 ,
27277.24870066, 38075.695572 , 29507.51278367, 47567.16415645,
51286.23469087, 47968.89818543, 30022.46176357, 58424.44455187,
28890.33956441, 58527.01828689, 29711.89245268, 49986.83431792,
26963.69820883, 27047.82972017, 27549.03117004, 29736.60915555,
38097.81624054, 59968.47612764, 39409.43267667, 27925.27962182,
48231.27131346, 32098.50303675, 25953.96624537, 26001.95567953,
29324.60767097, 32149.61603446, 48353.11915766, 48240.13771143,
45602.63933716, 29569.49656528, 51545.47129181, 33947.92743422,
27430.41419115, 28704.32870488, 51395.54706259, 28604.22057791,
47303.81628361, 40477.98474738, 38132.99012063, 29650.20889916,
28932.27929823, 31523.94515351, 25727.98219254, 30634.32389872,
46043.07253629, 51206.92118266, 42060.47671755, 51593.47210413,
50401.27391105, 58631.78147474, 29909.05600218, 58732.97038587,
60801.8834273 , 58692.08881181, 29692.68917817, 30925.27149712,
46988.58112823, 29835.4853029 , 59055.03833111, 29451.83365309,
26242.69709591, 39220.27814508, 27440.87957666, 38059.43297124,
32128.31152765, 29025.14523826, 47280.89345641, 51787.67884856,
27555.40131237, 27623.42346686, 29461.61772044, 31206.51969055,
51671.04408116, 30657.54357999, 48279.46775832, 52300.78396264,
32363.88162857, 47675.03951997, 28877.78285903, 24903.88354604,
30029.49095434, 32382.66030337, 25665.11050228, 31015.90930041,
25740.62346501, 31878.37414243, 38137.44137197, 29927.03748963,
50610.46402952, 25775.60992571, 29939.24521066, 49873.48516113,
59648.4652893 , 31329.870771 , 55922.70499517, 51980.62556903,
29402.47857454, 32589.83243627, 46577.14226702, 30859.16072183,
30631.79359158, 39753.01852912, 56419.49012589, 28945.9564858 ,
28748.36681932, 29684.50034484, 29720.68512489, 27931.61294004,
38259.16748592, 39295.1017613 , 32511.64764206, 30137.42605213,
26948.82636078, 50065.55352507, 37872.88450499, 42995.90269203,
25686.03577608, 46302.78587327, 30212.81922802, 60315.18441018,
38563.80445647, 30033.65854605, 29004.42265619, 32108.49844579,
29990.16684709, 51005.39234522, 40513.4387057 , 60995.11388111,
31242.08378121, 60444.50505358, 31713.54250437, 48361.75757929,
57051.69590203, 33393.76712508, 29328.60029457, 47562.07567199,
48696.27899294, 51350.36616009, 29920.56619899, 40993.94900797,
40621.69737366, 26073.94238516, 30325.9585736 , 59676.62985432,
49282.21422157, 50690.68591558, 58039.61324427, 27791.50485532,
40094.07675544, 36705.08978388, 55630.70395805, 50075.3099924 ,
50407.86858563, 47834.01703847, 26644.06933713, 38777.86651779,
50951.44287304, 25405.27185422, 32504.60588669, 33804.83318203,
48960.64269247, 26988.18380362, 48216.12925981, 46774.11901998,
59508.82295101, 59414.21449035, 30596.03535028, 31253.79578455,
55096.79620534, 47886.50257596, 31108.4210907 , 32164.81419591,

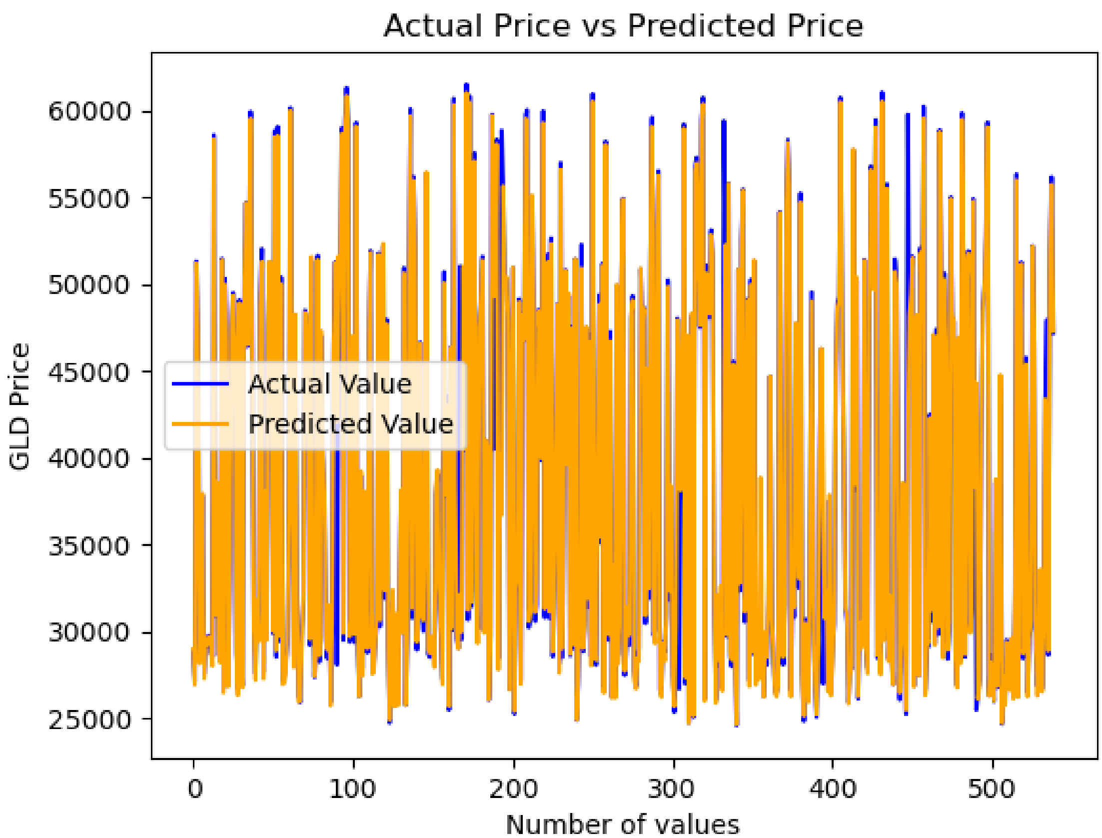
48435.72203138, 47415.83539128, 39957.33771627, 59261.19337631,
31301.44109766, 50051.85627818, 51261.85880135, 31147.66463518,
52297.04714892, 28837.43290822, 33435.2916607 , 40292.78814184,
48808.93885194, 27616.51586295, 56560.08416273, 29023.82530222,
40135.46056656, 50770.19902988, 39630.02882255, 49451.12410597,
28690.63334869, 47401.28962682, 29242.57398294, 51430.49161337,
24902.7943763 , 27301.84562774, 32610.63904669, 50899.23830974,
29032.46925105, 30329.74990979, 47537.19841887, 31905.21906114,
46657.51294367, 28038.04210646, 60499.48049502, 28893.55611252,
28349.55795358, 38042.28340044, 48791.79920606, 35451.41020535,
51035.02310751, 26467.68227697, 58016.81308725, 38025.40709725,
29810.37926923, 46701.36169953, 26174.89094321, 33793.19431288,
26157.21529385, 49945.10958957, 28154.79098525, 30673.64240768,
49789.04421182, 54854.60533408, 27717.52987775, 29375.89564664,
31381.85341116, 29787.98054981, 48319.18537036, 49049.12180426,
27239.19502518, 26692.01726199, 28453.51530779, 28294.95983811,
50899.40620694, 47587.09836174, 48580.08059216, 30904.24156664,
45083.15114613, 32479.11488559, 50858.70940648, 59060.38826018,
30157.32080603, 29320.79032186, 52867.31297897, 56243.03159474,
26637.26143988, 26199.6708044 , 29297.52570084, 28287.87503277,
30873.41759452, 49837.05892005, 32345.27355816, 38372.45301125,
26966.63458583, 25670.17225666, 30459.58636294, 47922.41461604,
45595.35989689, 38104.98617085, 38967.41927826, 58927.5901903 ,
27342.55502553, 47032.85914006, 24725.35229775, 26523.28964148,
48324.16786816, 25141.08383581, 56508.09438194, 56928.1819994 ,
51476.37874453, 47614.6016375 , 59066.78222391, 60368.18738751,
25989.1849433 , 50596.23404683, 49156.69345376, 48304.06499314,
52815.89783324, 51738.33307374, 31521.45558001, 25882.18094749,
32086.80394357, 28384.14141124, 32595.14161385, 26574.29493943,
45593.05025737, 52223.64372225, 29792.45024008, 55758.86928573,
28784.0927831 , 28367.02376691, 45358.7302203 , 28124.07417111,
24673.31036189, 50825.82407573, 32769.60708072, 52058.4142128 ,
55374.87209743, 31142.10660993, 32412.04637586, 49048.58420642,
26837.42242842, 49937.24289864, 28810.66461501, 51381.07729355,
26925.39918164, 30775.33083747, 27257.50033604, 38860.38519673,
28033.13676798, 26227.81260981, 29959.26665399, 28513.26763296,
32329.19968674, 44688.47721058, 30917.60167731, 28696.00802188,
26437.06296242, 26240.74135649, 26826.61553178, 54074.14966172,
33756.69217894, 28512.88930771, 31830.8701845 , 54748.34477539,
58120.78250447, 55737.83954251, 26246.56794081, 33527.34592226,
31387.13140493, 47741.36347703, 38247.51612685, 33061.55191596,
54689.68374097, 27132.79624285, 25144.62295831, 30469.89838039,
28795.89826491, 25930.98875677, 38138.42374674, 49002.89986412,
31487.68223627, 26987.15230131, 25227.45283726, 28655.45561692,
40437.12553725, 46259.47422935, 39278.15429188, 30758.84344227,
32290.78337638, 26551.59548726, 37861.58717188, 26313.98090369,
28292.11533527, 30923.49619233, 44929.96759719, 48598.21601803,

```
49052.96272351, 60464.22683795, 47567.0580802 , 40338.05060198,
31359.02337503, 30951.32922931, 25849.50607227, 27475.25964611,
27962.60898815, 57733.64503105, 38498.98483921, 50371.73632428,
26268.14358543, 30498.50978895, 29044.03856377, 30931.88852796,
51350.40182958, 29182.80173982, 27578.43512771, 47698.59460236,
56538.65841466, 51015.02289048, 49671.68000546, 58991.58746746,
28622.19875524, 30399.72787703, 27514.55103182, 60503.38080638,
29473.09104472, 38307.27007816, 55456.83301988, 28277.72997501,
32034.41948913, 28356.4493156 , 26952.39723377, 50693.35878538,
49846.87836011, 38206.39464775, 26467.82695317, 29850.23680501,
38549.25824995, 31189.17271508, 25464.47534322, 41952.56938826,
48163.63789449, 49404.78606973, 51495.07571279, 26805.75125667,
47033.56479878, 48209.40304408, 27443.52611543, 51765.79303751,
48862.5609984 , 59528.97261765, 26336.11069057, 29492.80127074,
29351.4788884 , 42294.54484063, 31222.50184041, 47060.58419634,
29211.49326034, 46922.21367741, 30242.05441689, 58738.52714391,
50966.60221219, 28477.31772689, 50240.67714068, 46628.29201202,
28941.36592962, 30559.2531468 , 54890.0853066 , 48235.10996977,
47899.21958922, 29051.95547639, 26758.83879976, 46908.04137146,
28162.58658753, 59456.01671344, 50502.89020832, 28685.92777037,
50108.99806981, 51737.23060342, 29917.24856506, 31146.5592936 ,
54749.87036895, 38327.20854398, 44281.28049427, 26105.27081748,
26993.19898491, 40985.57889119, 48403.93058147, 49942.85829358,
52862.21460709, 59002.38779301, 26293.67453567, 28164.80841031,
27453.41471823, 26007.76302032, 38771.67387902, 26829.07202153,
31399.18761281, 44762.14713244, 24721.37137702, 27425.7003095 ,
25780.19968382, 29382.27164949, 27820.28595092, 26615.3226664 ,
29594.99951035, 26114.83058925, 32421.73830879, 55959.46274286,
26208.32039093, 47057.19801291, 51131.55077916, 28728.42620506,
28979.25125154, 45375.10991175, 26246.17494341, 29683.88507502,
30703.35480221, 52163.40605555, 46157.5375512 , 29527.11977638,
26302.75037376, 28968.05103122, 33562.65474021, 26534.22038549,
30248.12618661, 43354.41393013, 41325.82904874, 28888.96049638,
51533.81835687, 55704.98798754, 47309.33315245])
```

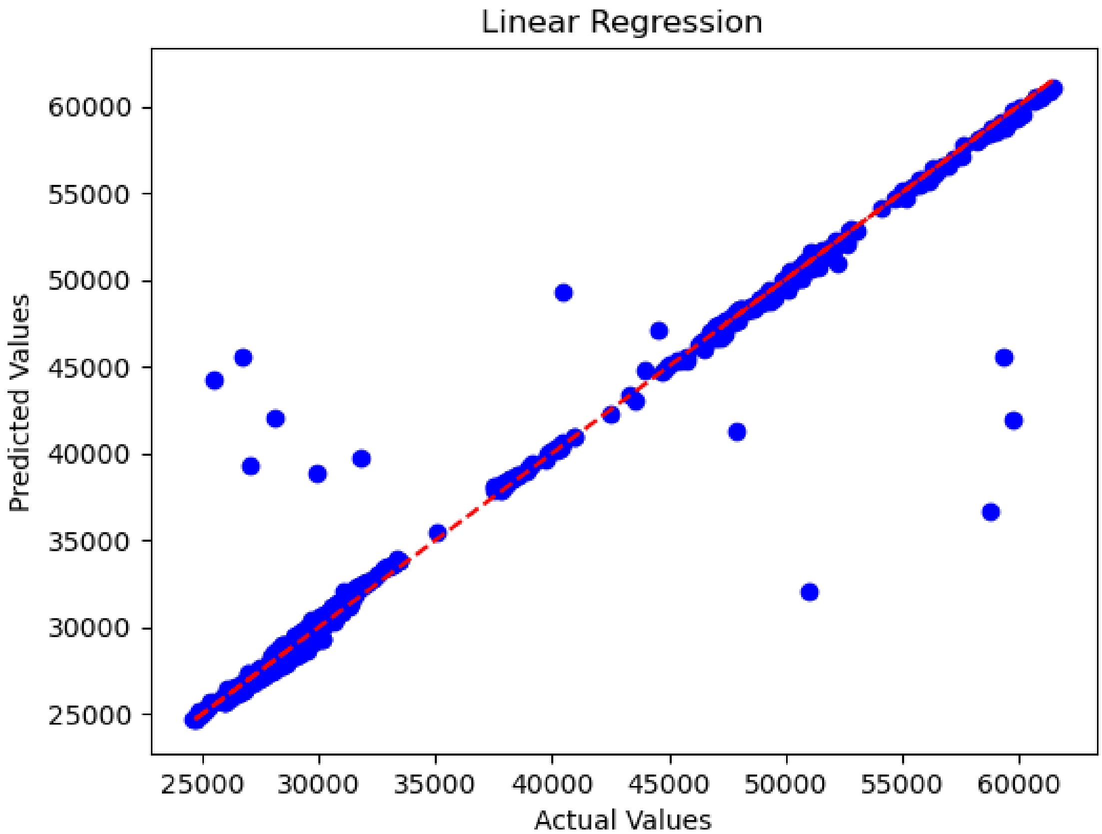
[24]:
```python
error_score = metrics.r2_score(Y_test, test_data_prediction)
mae_lr = mean_absolute_error(Y_test, test_data_prediction)
mse_lr = mean_squared_error(Y_test, test_data_prediction)
print("MAE (Linear Regression): {:.2f}".format(mae_lr))
print("MSE (Linear Regression): {:.2f}".format(mse_lr))
print("R squared error : ", error_score)
```

```
MAE (Linear Regression): 599.70
MSE (Linear Regression): 5098138.32
R squared error :  0.9601713615094156
```

[25]:
```
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(test_data_prediction, color='orange', label='Predicted Value')
plt.title('Actual Price vs Predicted Price')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```

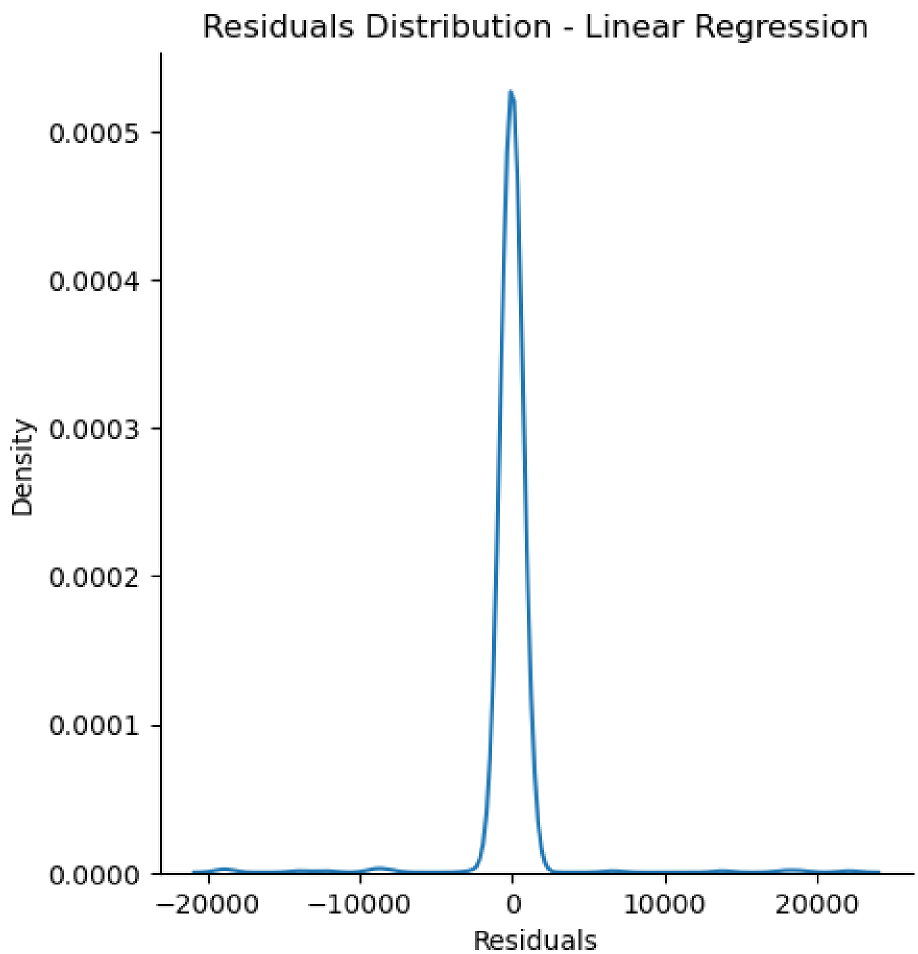


[26]:
```
plt.scatter(Y_test, test_data_prediction, color='blue')
plt.plot(Y_test, Y_test, color='red', linestyle='--')
plt.title('Linear Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



[27]:
```
sns.displot(Y_test - test_data_prediction, kind='kde')
plt.title('Residuals Distribution - Linear Regression')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```

```
C:\Users\sarka\anaconda3\envs\Projects\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[28]:
```
print("Accuracy of linear Regression model: {:.2f}%".format(error_score * 100))
```

```
Accuracy of linear Regression model: 96.02%
```

[29]:
```
pickle.dump(linear_regressor,open('linear_regressor.pkl','wb'))
```

1.6 RANDOM REGRESSION

[30]:
```
random_regressor = RandomForestRegressor(n_estimators=100)
```

[31]:
```
random_regressor.fit(X_train_scaled, Y_train)
```

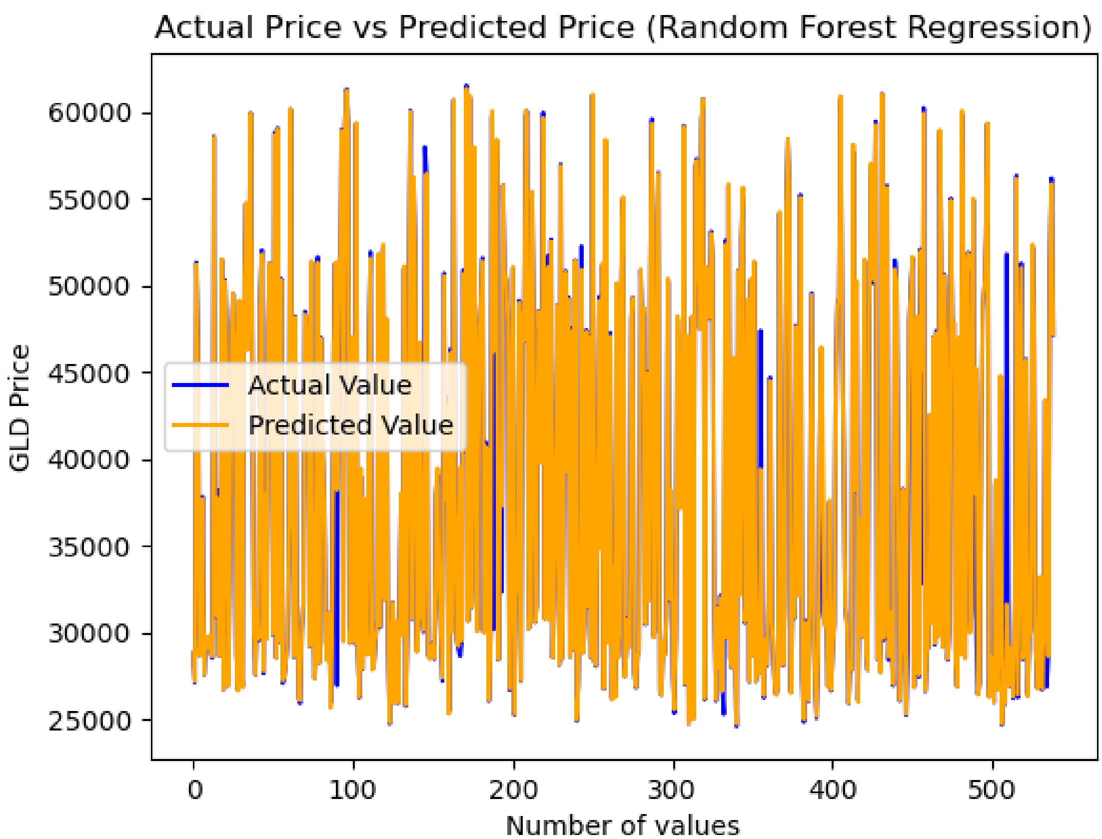
[31]:
```
RandomForestRegressor()
```

[32]:
```
test_data_predictionR = random_regressor.predict(X_test_scaled)
```

[33]:
```
error_scoreR = metrics.r2_score(Y_test, test_data_predictionR) #add␣
 ↪test_data_prediction for random forest
mae_rf = mean_absolute_error(Y_test, test_data_predictionR)
mse_rf = mean_squared_error(Y_test, test_data_predictionR)
print("MAE (Random Forest Regression): {:.2f}".format(mae_rf))
print("MSE (Random Forest Regression): {:.2f}".format(mse_rf))
print("R squared error : ", error_scoreR)
```

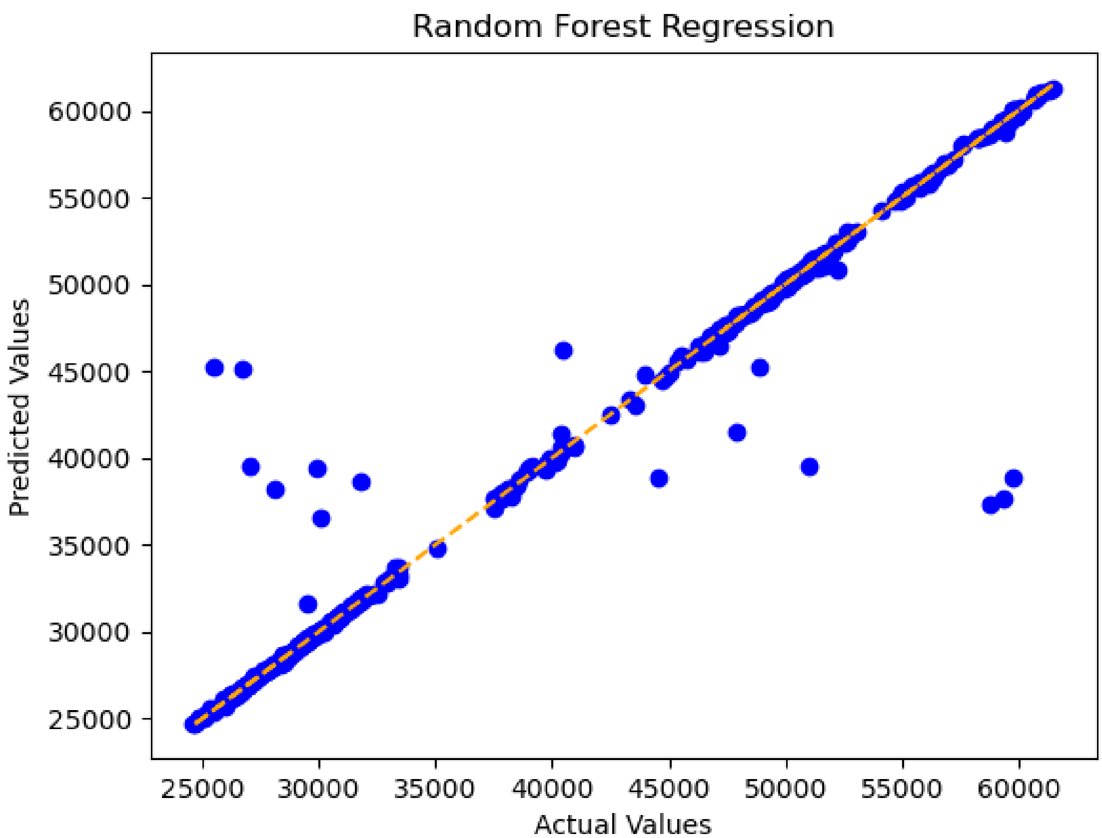
```
MAE (Random Forest Regression): 440.98
MSE (Random Forest Regression): 5208386.08
R squared error :  0.9593100631769695
```

[102]:
```
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(test_data_predictionR, color='orange', label='Predicted Value')
plt.title('Actual Price vs Predicted Price (Random Forest Regression)')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```



[36]:

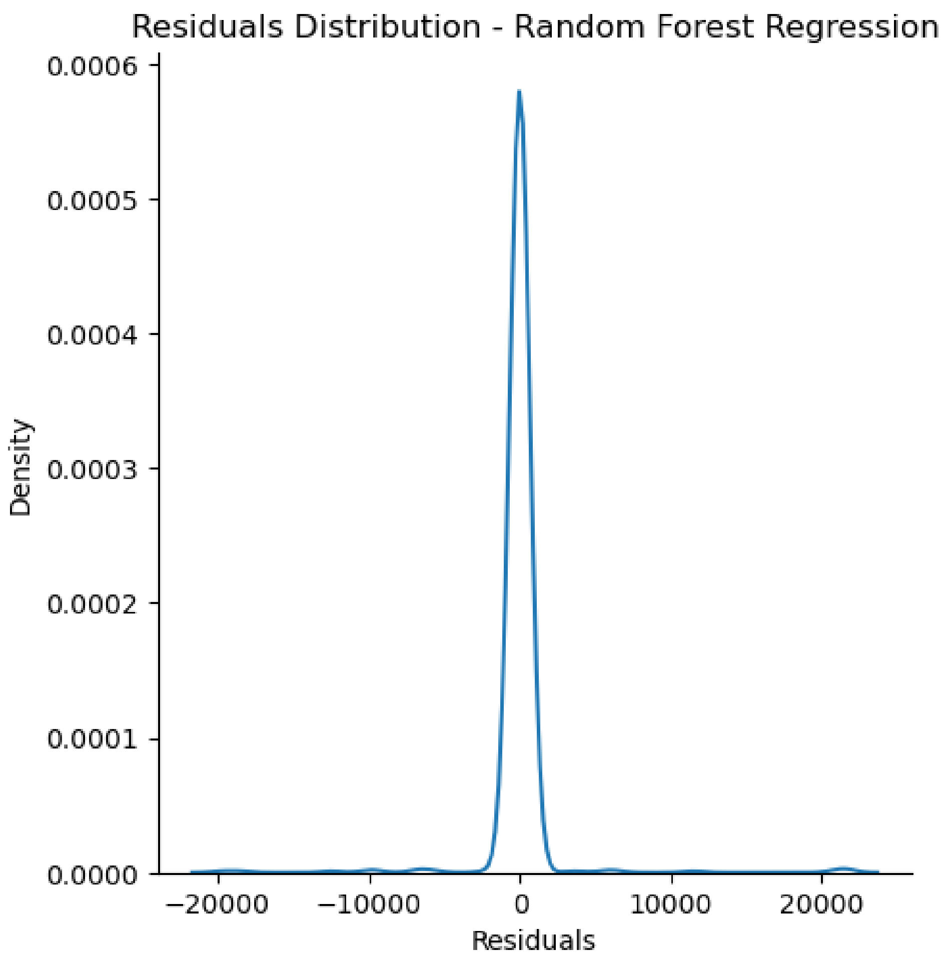
```
plt.scatter(Y_test, test_data_predictionR, color='blue')
plt.plot(Y_test, Y_test, color='orange', linestyle='--')
plt.title('Random Forest Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



[37]:

```
sns.displot(Y_test - test_data_predictionR, kind='kde')
plt.title('Residuals Distribution - Random Forest Regression')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```

```
C:\Users\sarka\anaconda3\envs\Projects\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[38]:
```
print("Accuracy of random forest Regression model: {:.2f}%".format(error_scoreR␣
↪* 100))
```

```
Accuracy of random forest Regression model: 95.93%
```

[39]:
```
pickle.dump(random_regressor,open('random_regressor.pkl','wb'))
```

1.7 XGBOOST

[40]:
```
import xgboost as xgb
from sklearn.metrics import r2_score
```

[41]:
```
xgb_regressor = xgb.XGBRegressor()
xgb_regressor.fit(X_train_scaled, Y_train)
```

[41]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
```

```
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

[42]:
```
Y_pred_xgb = xgb_regressor.predict(X_test_scaled)
```

[43]:
```
Y_pred_xgb
```

[43]:
```
array([28841.941, 27170.264, 51470.887, 45909.82 , 28610.674, 30480.367,
       37219.   , 27680.314, 29764.842, 29117.53 , 29723.436, 28853.213,
       28564.691, 58520.49 , 30947.86 , 38471.51 , 28524.65 , 28817.498,
       51331.39 , 26662.615, 50204.35 , 46489.844, 26903.797, 33044.574,
       47357.566, 49651.062, 47435.918, 28703.87 , 26700.656, 49106.67 ,
       28621.35 , 26894.277, 50327.527, 54649.58 , 46367.355, 50082.74 ,
       60021.316, 45327.56 , 29894.572, 27630.934, 31390.066, 29706.83 ,
       49242.36 , 51530.59 , 27822.86 , 35077.24 , 29511.455, 47408.824,
       51571.133, 47866.133, 29787.97 , 58629.508, 28547.078, 59593.117,
       29599.13 , 50420.305, 27196.814, 27325.336, 28085.049, 29482.254,
       35104.508, 60332.01 , 39083.656, 28571.986, 48130.812, 31538.01 ,
       26232.906, 26108.268, 29999.896, 31796.693, 48151.824, 48132.28 ,
       45541.56 , 29202.67 , 51227.773, 33369.63 , 27321.402, 29538.703,
       51186.06 , 28315.5  , 46937.633, 40790.086, 37680.496, 29392.975,
       28239.25 , 31086.803, 25954.299, 30117.627, 46179.082, 51399.89 ,
       42413.71 , 51429.477, 50870.684, 58741.59 , 29470.258, 59427.996,
       60926.504, 58837.688, 29463.133, 30419.258, 47061.81 , 29471.145,
       59400.742, 28824.117, 26253.684, 38463.69 , 27880.715, 37942.91 ,
       31446.703, 28686.059, 47086.008, 51125.812, 27921.277, 28035.268,
       29179.62 , 30567.014, 52185.656, 30339.332, 48281.867, 52247.67 ,
       31867.309, 48059.957, 29054.31 , 24818.342, 29613.625, 31617.598,
       25850.082, 30512.115, 25960.395, 31158.75 , 37964.863, 29847.264,
       51046.254, 25717.27 , 29971.44 , 50309.29 , 59742.027, 30760.023,
       56034.62 , 51733.676, 28847.23 , 32085.9  , 46686.027, 30945.934,
       30111.475, 40410.242, 56688.027, 28731.984, 28510.723, 29261.49 ,
       29360.707, 28570.197, 38020.316, 38806.215, 32036.889, 29616.938,
       27391.738, 50709.676, 37673.523, 43086.965, 25320.02 , 46268.062,
       29932.996, 60823.953, 38325.65 , 29591.178, 29145.684, 40378.36 ,
       29615.686, 50754.074, 40310.33 , 61324.2  , 30657.361, 60455.844,
       31473.873, 48626.72 , 57447.26 , 32940.66 , 30107.006, 47749.984,
       49262.1  , 51262.8  , 30048.412, 40695.426, 40478.6  , 26070.365,
       29705.049, 60033.688, 52666.94 , 50531.484, 58526.81 , 28461.727,
       39812.76 , 34024.348, 55774.715, 50631.074, 50528.03 , 47716.043,
```

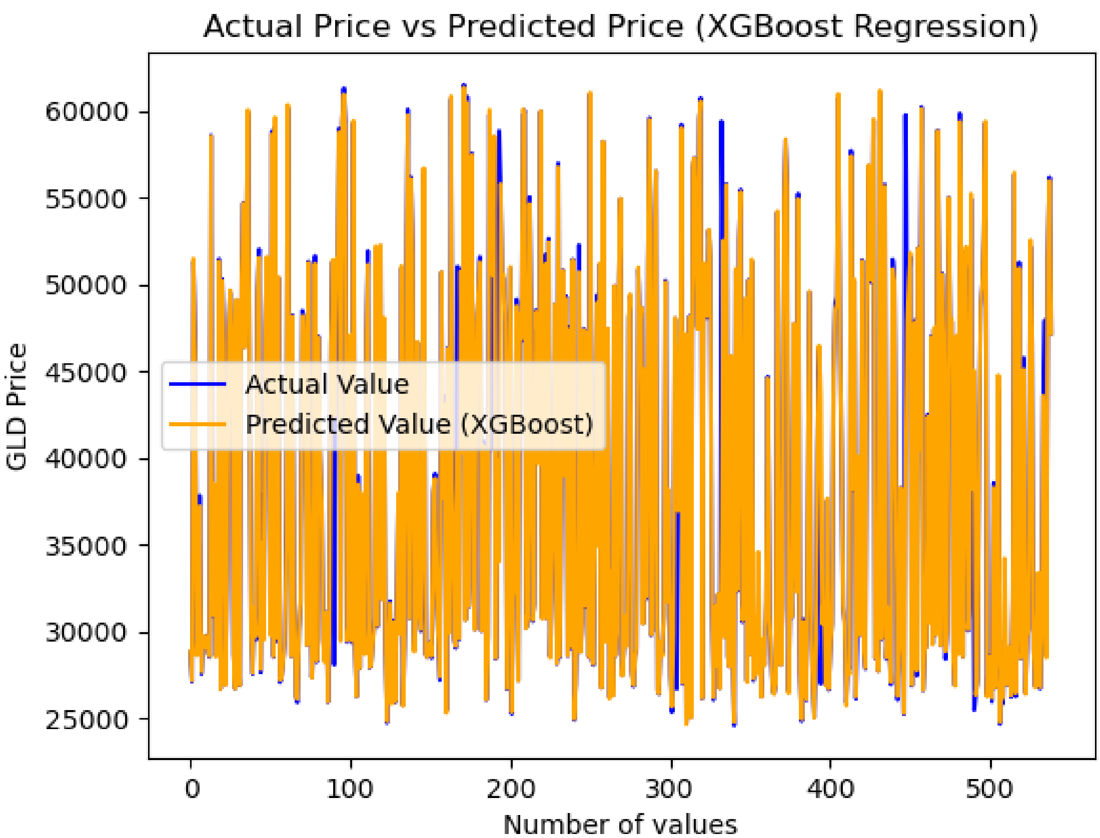
26715.895, 38394.91 , 50990.277, 25354.887, 31987.525, 32793.805,
48729.92 , 27163.496, 48257.06 , 46878.836, 60080.324, 59890.004,
30178.07 , 31032.744, 54613.09 , 47870.46 , 30665.402, 31525.244,
48415.258, 47217.305, 39671.047, 59964.766, 30721.896, 49999.73 ,
51163.848, 30749.896, 52400.703, 28484.04 , 33470.777, 40172.15 ,
48857.754, 28149.725, 56755.457, 28559.959, 39941.06 , 50786.14 ,
39000.395, 49153.793, 28600.598, 47365.098, 29128.225, 51394.152,
24950.875, 26958.588, 32043.521, 50700.383, 28823.332, 30478.307,
47373.305, 31479.209, 47215.406, 28546.592, 61031.535, 28423.729,
28136.3 , 37633.758, 48928.434, 34921.453, 51188.73 , 26747.719,
58209.094, 37737.707, 29402.34 , 47448.902, 26125.744, 33486.69 ,
26297.453, 49887.35 , 28630.28 , 30183.027, 50111.824, 54927.27 ,
27446.19 , 29078.68 , 30919.928, 29143.36 , 48156.01 , 49414.38 ,
27772.506, 26903.62 , 28424.81 , 29078.738, 50970.055, 47651.066,
48656.094, 30451.262, 45069.812, 31991.646, 51121.195, 59470.242,
29805.064, 30175.848, 52870.504, 56508.73 , 26570.822, 26414.412,
29401.4 , 28587.402, 30918.98 , 50137.176, 31755.094, 38178.484,
27491.854, 25660.71 , 29886.094, 48047.73 , 45320.64 , 36930.676,
38720.902, 58996.36 , 27057.201, 47144.312, 24666.844, 26385.428,
48113.004, 25019.4 , 56915.156, 57238.598, 51607.445, 47413.344,
59541.277, 60543.04 , 26149.684, 51196.082, 49478.414, 48141.496,
53134.105, 51796.46 , 31274.553, 26189.258, 31459.861, 28030.607,
32164.424, 26713.84 , 35630.562, 52511.184, 29673.416, 55708.125,
28593.848, 27966.   , 45895.6 , 28337.186, 24786.424, 50882.145,
32401.717, 52368.81 , 55311.902, 30645.195, 31777.799, 49152.652,
27095.451, 50208.734, 28696.307, 51434.02 , 27269.04 , 30856.65 ,
27587.469, 34587.26 , 28209.623, 26218.06 , 29571.64 , 28082.146,
31829.174, 44571.832, 31029.77 , 28702.652, 26487.29 , 26396.643,
26930.992, 54185.676, 33459.6 , 28065.322, 31505.502, 54998.17 ,
58337.29 , 55656.785, 26447.34 , 32966.35 , 30844.396, 47710.332,
37762.223, 32278.885, 54904.16 , 27739.342, 24942.354, 30561.45 ,
28157.748, 26099.357, 37857.047, 49587.707, 31123.521, 26871.83 ,
25018.215, 28482.146, 40232.35 , 46444.02 , 43652.16 , 30434.506,
31722.963, 26878.023, 37676.965, 26696.28 , 28912.193, 30524.125,
44827.387, 48518.69 , 48857.984, 60957.883, 47509.99 , 40624.914,
31472.43 , 30460.455, 25723.54 , 27908.506, 27537.176, 57384.355,
38194.62 , 50279.65 , 26247.275, 40114.207, 29777.438, 30568.994,
51333.77 , 28930.357, 27893.18 , 47361.574, 56866.555, 50943.414,
50228.816, 59486.113, 28488.69 , 29709.135, 27715.96 , 61154.65 ,
29598.566, 38022.6 , 55639.12 , 28485.24 , 31365.55 , 29089.738,
27084.445, 50871.445, 50028.832, 38025.594, 26289.338, 29541.025,
38281.29 , 30852.248, 25302.854, 39312.57 , 47941.14 , 50245.082,
51789.49 , 26915.059, 47186.664, 47817.863, 27760.621, 52083.504,
40478.754, 60091.48 , 26624.137, 29041.844, 29098.398, 42389.75 ,
30491.7 , 46936.055, 29252.812, 47479.234, 29807.666, 58817.477,
50905.8 , 29179.62 , 50538.895, 47058.797, 28859.322, 30065.896,
54980.965, 48163.477, 47918.51 , 28812.582, 26855.074, 47041.582,

```
 28691.578, 59361.26 , 50222.34 , 28535.121, 50559.066, 52159.062,
 30116.537, 31244.68 , 55206.805, 38188.31 , 45003.74 , 26341.08 ,
 27202.8  , 40813.746, 48771.547, 50049.938, 53021.72 , 59368.508,
 26246.496, 28549.145, 27949.838, 26163.564, 38285.555, 26762.826,
 30833.727, 44759.78 , 24786.424, 27882.744, 26155.145, 34207.953,
 28489.152, 26925.129, 29121.496, 26322.527, 31824.053, 56423.85 ,
 26449.121, 33068.113, 50917.406, 28452.633, 28784.447, 45090.617,
 26411.168, 29617.94 , 30148.89 , 52546.004, 46279.14 , 28952.191,
 26754.918, 28951.873, 33391.07 , 26787.83 , 29954.197, 43588.953,
 40934.812, 28487.537, 51437.766, 55943.344, 47146.18 ],
dtype=float32)
```

[44]:
```python
r2_xgb = r2_score(Y_test, Y_pred_xgb)
mae_xgb = mean_absolute_error(Y_test, Y_pred_xgb)
mse_xgb = mean_squared_error(Y_test, Y_pred_xgb)
print("MAE (XGBoost Regression): {:.2f}".format(mae_xgb))
print("MSE (XGBoost Regression): {:.2f}".format(mse_xgb))
print("XGBoost Regression R squared error: ", r2_xgb)
```

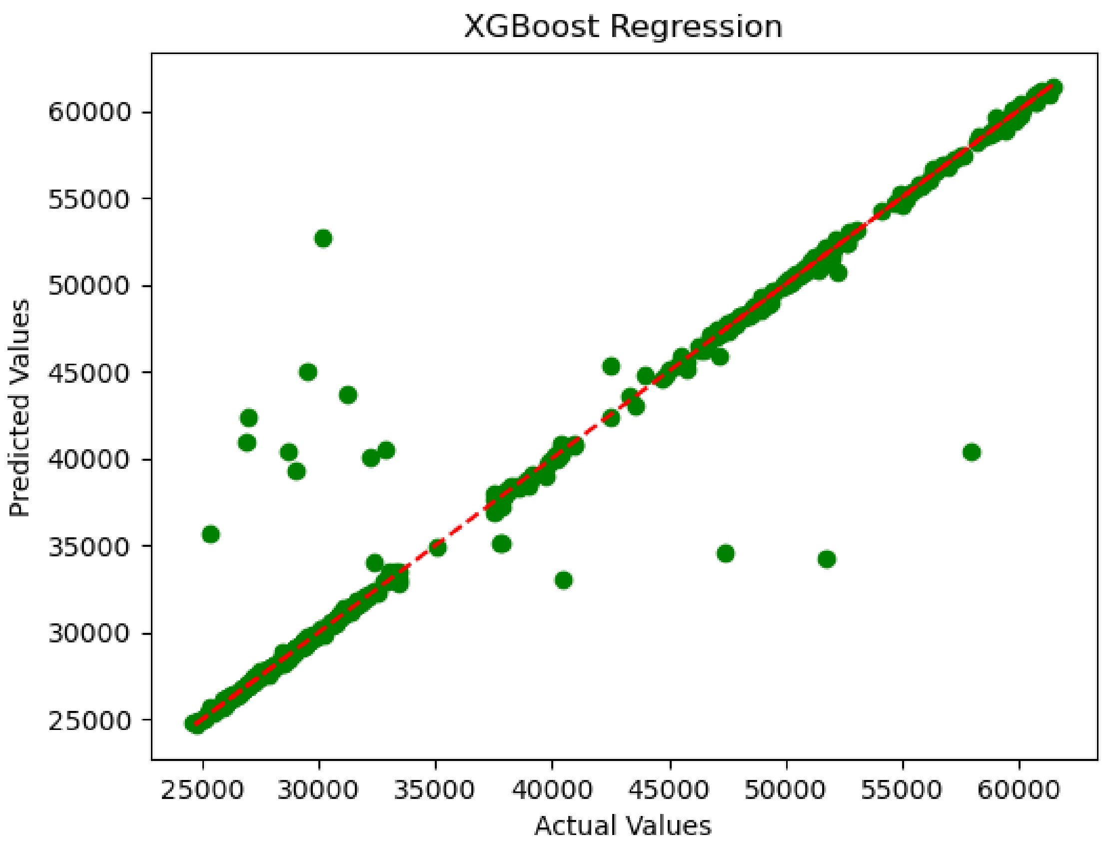
```
MAE (XGBoost Regression): 529.31
MSE (XGBoost Regression): 6631199.81
XGBoost Regression R squared error:  0.9481944891347766
```

[45]:
```python
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_xgb, color='orange', label='Predicted Value (XGBoost)')
plt.title('Actual Price vs Predicted Price (XGBoost Regression)')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```



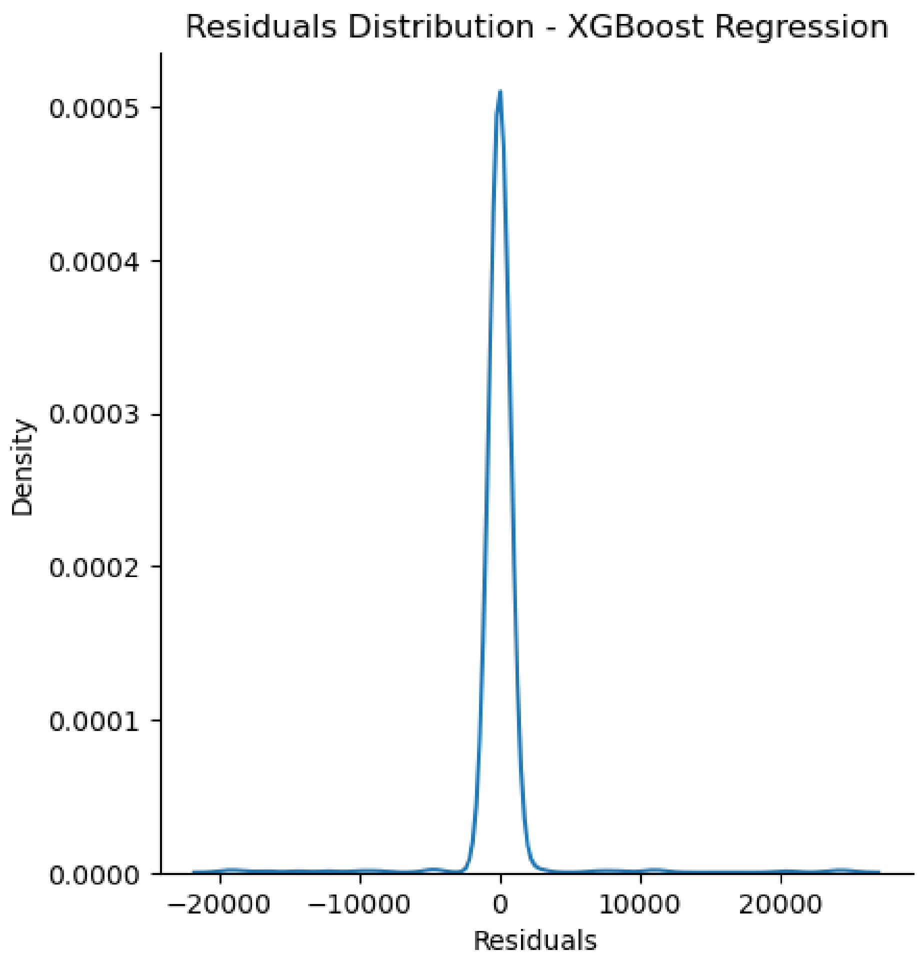
[101]:

```
plt.scatter(Y_test, Y_pred_xgb, color='green')
plt.plot(Y_test, Y_test, color='red', linestyle='--')
plt.title('XGBoost Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



[47]:
```
sns.displot(Y_test - Y_pred_xgb, kind='kde')
plt.title('Residuals Distribution - XGBoost Regression')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```

```
C:\Users\sarka\anaconda3\envs\Projects\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[48]:
```
print("Accuracy of XGBoost Regression model: {:.2f}%".format(r2_xgb * 100))
```

```
Accuracy of XGBoost Regression model: 94.82%
```

[49]:
```
pickle.dump(xgb_regressor,open('xgb_regressor.pkl','wb'))
```

1.8 ADA BOOST Regression

[50]:
```
adaboost_regressor = AdaBoostRegressor()
```

[51]:
```
adaboost_regressor.fit(X_train_scaled, Y_train)
```

[51]:
```
AdaBoostRegressor()
```

[52]:
```
Y_pred_adaboost = adaboost_regressor.predict(X_test_scaled)
```

[53]:
```
Y_pred_adaboost
```

[53]:
```
array([30713.88687783, 30380.91340782, 48718.67172676, 43991.74691865,
       28658.74385511, 30713.88687783, 40563.46759639, 28658.74385511,
       30412.94402421, 30713.88687783, 30713.88687783, 30713.88687783,
       30380.91340782, 57359.91428571, 32814.89508197, 39729.40931373,
       28762.53266332, 30713.88687783, 48718.67172676, 28443.88975155,
       50683.62162162, 43955.58040665, 28524.7877095 , 35304.01846154,
       43955.58040665, 44093.07635468, 43991.74691865, 30545.12987013,
       28268.52873563, 44093.07635468, 29182.94979079, 28443.88975155,
       43955.58040665, 50683.62162162, 42378.88400703, 43955.58040665,
       59415.46315789, 42005.50229885, 30713.88687783, 29182.94979079,
       34589.74659401, 30713.88687783, 43955.58040665, 48718.67172676,
       28658.74385511, 39729.40931373, 30713.88687783, 43955.58040665,
       48718.67172676, 43955.58040665, 30713.88687783, 57359.91428571,
       28658.74385511, 57359.91428571, 30713.88687783, 48718.67172676,
       29342.52219873, 28658.74385511, 28762.53266332, 30399.40298507,
       40563.46759639, 59415.46315789, 40563.46759639, 30380.91340782,
       44092.33023796, 32814.89508197, 28437.38770053, 28268.52873563,
       30713.88687783, 33864.73134328, 44092.33023796, 46210.80573951,
       42005.50229885, 30399.40298507, 48718.67172676, 39729.40931373,
       30412.94402421, 30713.88687783, 50477.55339806, 28658.74385511,
       45112.97333333, 40917.08280255, 40563.46759639, 30713.88687783,
       29182.94979079, 34589.74659401, 28151.89889026, 30399.40298507,
       42378.88400703, 48718.67172676, 42342.04470588, 48718.67172676,
       48718.67172676, 58671.52884615, 30412.94402421, 58671.52884615,
       59415.46315789, 57359.91428571, 30380.91340782, 33864.73134328,
       43955.58040665, 30412.94402421, 59415.46315789, 30545.12987013,
       28437.38770053, 40563.46759639, 28658.74385511, 40563.46759639,
       32814.89508197, 30380.91340782, 43955.58040665, 48718.67172676,
       30412.94402421, 30412.94402421, 30713.88687783, 32814.89508197,
       48718.67172676, 30399.40298507, 43991.74691865, 50683.62162162,
       32814.89508197, 44092.33023796, 30412.94402421, 28268.52873563,
       30412.94402421, 32814.89508197, 28268.52873563, 32814.89508197,
       28268.52873563, 32814.89508197, 40622.22585227, 30713.88687783,
       50683.62162162, 28151.89889026, 30380.91340782, 48718.67172676,
       59415.46315789, 32814.89508197, 57059.08888889, 48718.67172676,
       30545.12987013, 33864.73134328, 43955.58040665, 32814.89508197,
       30399.40298507, 42806.64996908, 57059.08888889, 30380.91340782,
       30412.94402421, 30713.88687783, 30380.91340782, 30412.94402421,
       40563.46759639, 40563.46759639, 32814.89508197, 30399.40298507,
       29342.52219873, 43991.74691865, 40563.46759639, 42378.88400703,
       28151.89889026, 42378.88400703, 30713.88687783, 59415.46315789,
       40563.46759639, 30713.88687783, 30713.88687783, 37217.54166667,
       30713.88687783, 50477.55339806, 40563.46759639, 59415.46315789,
       34589.74659401, 59415.46315789, 32814.89508197, 44092.33023796,
       57059.08888889, 35304.01846154, 30412.94402421, 43955.58040665,
```

```
43955.58040665, 48718.67172676, 30713.88687783, 42342.04470588,
40917.08280255, 28268.52873563, 30399.40298507, 59415.46315789,
43200.42703863, 48718.67172676, 58353.04863222, 28762.53266332,
40563.46759639, 38597.29441118, 50477.55339806, 44093.07635468,
43955.58040665, 43955.58040665, 28443.88975155, 40563.46759639,
48718.67172676, 28151.89889026, 33864.73134328, 34589.74659401,
43955.58040665, 29182.94979079, 43991.74691865, 43991.74691865,
59415.46315789, 59415.46315789, 30412.94402421, 34589.74659401,
50477.55339806, 43955.58040665, 32814.89508197, 33864.73134328,
46210.80573951, 43955.58040665, 40563.46759639, 59415.46315789,
32814.89508197, 43955.58040665, 48718.67172676, 34589.74659401,
48718.67172676, 29342.52219873, 34238.88043478, 40917.08280255,
43955.58040665, 30412.94402421, 57059.08888889, 29342.52219873,
42342.04470588, 48718.67172676, 39186.01918159, 43955.58040665,
28762.53266332, 43955.58040665, 30412.94402421, 48718.67172676,
28268.52873563, 28524.7877095 , 32814.89508197, 42806.64996908,
30713.88687783, 30412.94402421, 43955.58040665, 34589.74659401,
43955.58040665, 29342.52219873, 59415.46315789, 30380.91340782,
30380.91340782, 40622.22585227, 43955.58040665, 40563.46759639,
48718.67172676, 28443.88975155, 57059.08888889, 40563.46759639,
30399.40298507, 43955.58040665, 28151.89889026, 39729.40931373,
28443.88975155, 42828.89940828, 28658.74385511, 30399.40298507,
43991.74691865, 50683.62162162, 30380.91340782, 30713.88687783,
32814.89508197, 30412.94402421, 46210.80573951, 43955.58040665,
30412.94402421, 28443.88975155, 29182.94979079, 30713.88687783,
48718.67172676, 43955.58040665, 45904.28235294, 30713.88687783,
42378.88400703, 32814.89508197, 48718.67172676, 58671.52884615,
30399.40298507, 30713.88687783, 50683.62162162, 57059.08888889,
28437.38770053, 28437.38770053, 30713.88687783, 29182.94979079,
32814.89508197, 44093.07635468, 34238.88043478, 40563.46759639,
30412.94402421, 28151.89889026, 30399.40298507, 44092.33023796,
42342.04470588, 40563.46759639, 40563.46759639, 58671.52884615,
28658.74385511, 43991.74691865, 28268.52873563, 28268.52873563,
46210.80573951, 28151.89889026, 57059.08888889, 57059.08888889,
48718.67172676, 43955.58040665, 58671.52884615, 59415.46315789,
28443.88975155, 48718.67172676, 44093.07635468, 46210.80573951,
48718.67172676, 48718.67172676, 32814.89508197, 28443.88975155,
32814.89508197, 29342.52219873, 35304.01846154, 28443.88975155,
37474.95778612, 48718.67172676, 30380.91340782, 52909.0625    ,
30412.94402421, 29342.52219873, 42005.50229885, 29342.52219873,
28151.89889026, 48718.67172676, 35304.01846154, 48718.67172676,
50683.62162162, 32814.89508197, 34589.74659401, 43955.58040665,
30380.91340782, 48718.67172676, 29342.52219873, 48718.67172676,
28658.74385511, 32814.89508197, 29182.94979079, 38240.98188751,
29182.94979079, 28437.38770053, 30412.94402421, 29342.52219873,
34238.88043478, 42378.88400703, 32814.89508197, 30412.94402421,
28443.88975155, 28437.38770053, 28443.88975155, 50683.62162162,
```

```
39729.40931373, 29342.52219873, 32814.89508197, 50477.55339806,
57359.91428571, 52336.755     , 28437.38770053, 35304.01846154,
32814.89508197, 43955.58040665, 40622.22585227, 35304.01846154,
48718.67172676, 28658.74385511, 28151.89889026, 30713.88687783,
29342.52219873, 28268.52873563, 39729.40931373, 43955.58040665,
32814.89508197, 28437.38770053, 28151.89889026, 30380.91340782,
40917.08280255, 43955.58040665, 35304.01846154, 30713.88687783,
35304.01846154, 28443.88975155, 39729.40931373, 28443.88975155,
30713.88687783, 34589.74659401, 42378.88400703, 44093.07635468,
44093.07635468, 59415.46315789, 43955.58040665, 40917.08280255,
32814.89508197, 30713.88687783, 28151.89889026, 28658.74385511,
30380.91340782, 57059.08888889, 40622.22585227, 48718.67172676,
28443.88975155, 30991.3255814 , 30412.94402421, 32814.89508197,
48718.67172676, 30713.88687783, 28762.53266332, 45112.97333333,
57059.08888889, 48718.67172676, 43955.58040665, 58671.52884615,
29342.52219873, 30399.40298507, 30412.94402421, 59415.46315789,
30713.88687783, 40563.46759639, 50477.55339806, 30380.91340782,
30713.88687783, 30713.88687783, 28443.88975155, 48718.67172676,
44093.07635468, 39729.40931373, 28151.89889026, 30380.91340782,
40563.46759639, 34589.74659401, 28151.89889026, 40917.08280255,
43991.74691865, 43955.58040665, 48718.67172676, 28443.88975155,
43991.74691865, 44093.07635468, 29182.94979079, 48718.67172676,
43991.74691865, 59415.46315789, 28443.88975155, 30545.12987013,
30713.88687783, 40917.08280255, 30713.88687783, 43955.58040665,
30713.88687783, 43955.58040665, 30380.91340782, 58671.52884615,
48718.67172676, 30713.88687783, 50477.55339806, 43955.58040665,
30412.94402421, 30713.88687783, 50477.55339806, 43991.74691865,
43955.58040665, 30713.88687783, 28443.88975155, 43955.58040665,
30545.12987013, 59415.46315789, 48718.67172676, 30412.94402421,
48718.67172676, 48718.67172676, 30412.94402421, 32814.89508197,
50477.55339806, 40563.46759639, 48718.67172676, 28268.52873563,
28658.74385511, 42342.04470588, 43955.58040665, 43955.58040665,
50683.62162162, 58671.52884615, 28268.52873563, 29182.94979079,
28658.74385511, 28268.52873563, 40563.46759639, 28658.74385511,
30991.3255814 , 42378.88400703, 28151.89889026, 28658.74385511,
28268.52873563, 37474.95778612, 30380.91340782, 28437.38770053,
30412.94402421, 28443.88975155, 34238.88043478, 57059.08888889,
28437.38770053, 39729.40931373, 48718.67172676, 30380.91340782,
30713.88687783, 42005.50229885, 28151.89889026, 30380.91340782,
30399.40298507, 48718.67172676, 42378.88400703, 30545.12987013,
28443.88975155, 30713.88687783, 33864.73134328, 28443.88975155,
30399.40298507, 42342.04470588, 42828.89940828, 29342.52219873,
48718.67172676, 50477.55339806, 43955.58040665])
```

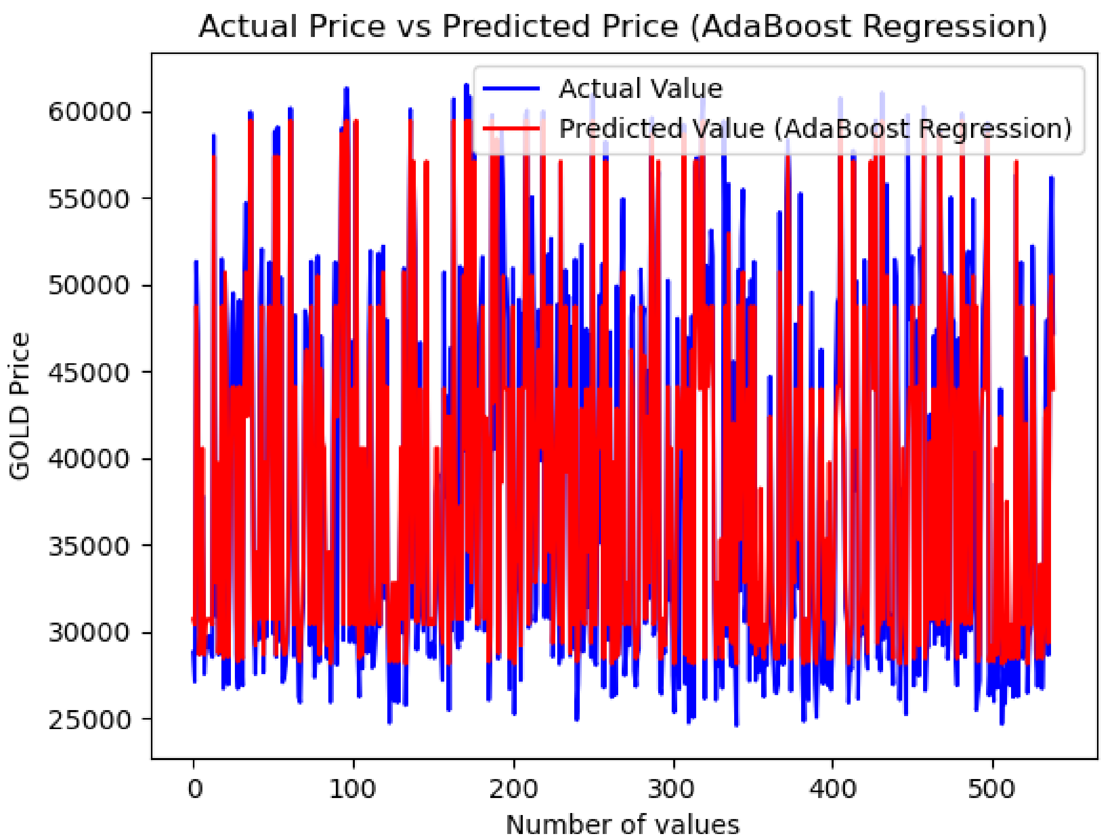
[54]:

```
r2_adaboost = r2_score(Y_test, Y_pred_adaboost)
mae_adaboost = mean_absolute_error(Y_test, Y_pred_adaboost)
mse_adaboost = mean_squared_error(Y_test, Y_pred_adaboost)
```

```
print("MAE (AdaBoost Regression): {:.2f}".format(mae_adaboost))
print("MSE (AdaBoost Regression): {:.2f}".format(mse_adaboost))
print("R squared error (AdaBoost Regression):", r2_adaboost)
```

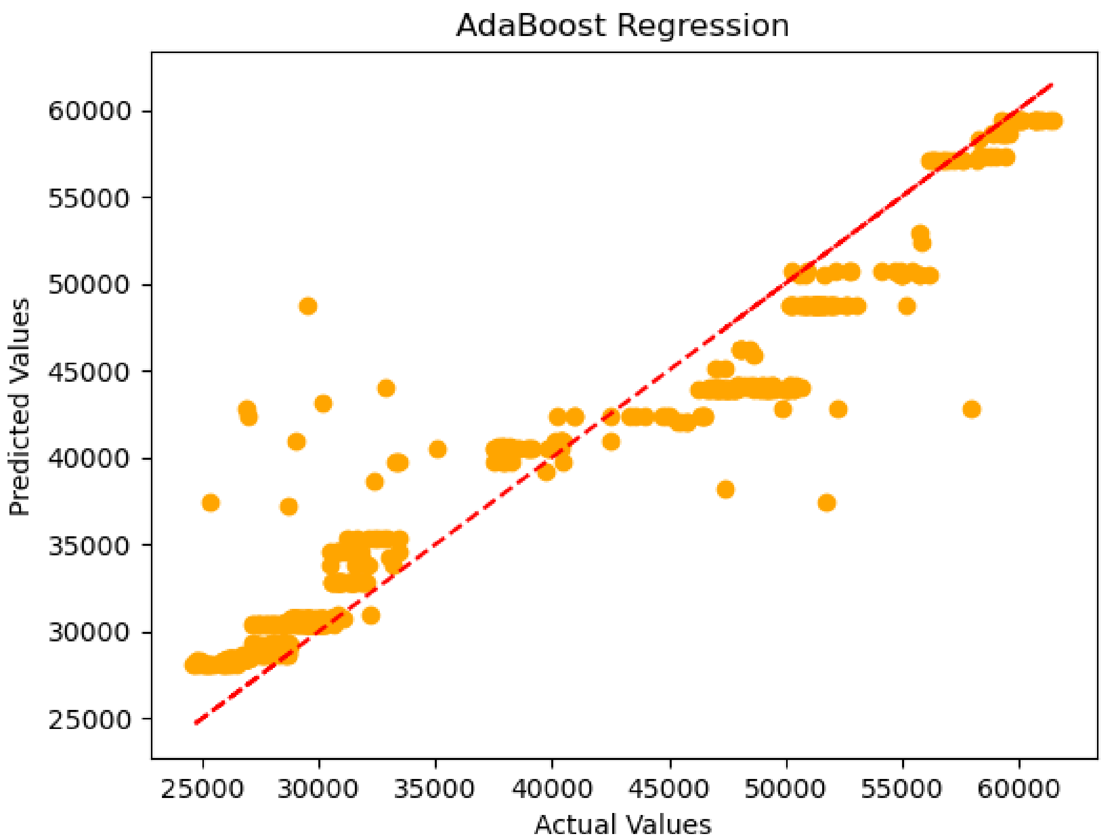
```
MAE (AdaBoost Regression): 2457.56
MSE (AdaBoost Regression): 11990808.47
R squared error (AdaBoost Regression): 0.9063231427262854
```

[55]:
```
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_adaboost, color='red', label='Predicted Value (AdaBoost␣
 ↪Regression)')
plt.title('Actual Price vs Predicted Price (AdaBoost Regression)')
plt.xlabel('Number of values')
plt.ylabel('GOLD Price')
plt.legend()
plt.show()
```



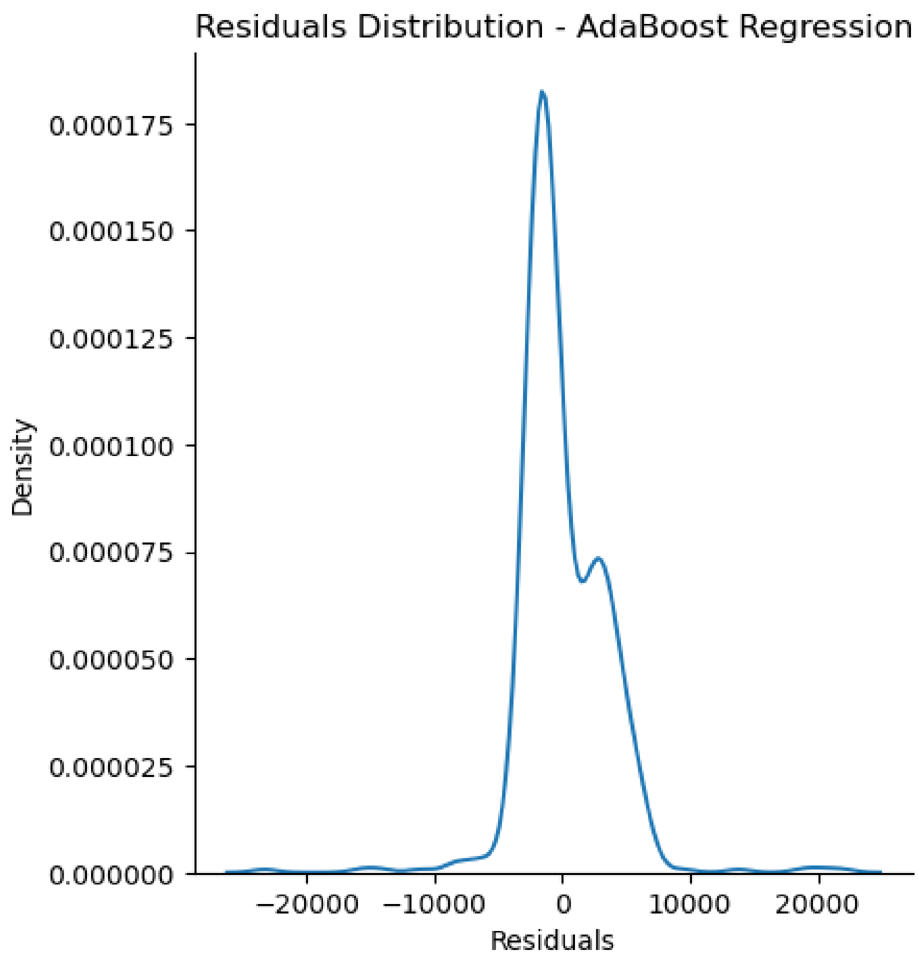
[103]:
```
plt.scatter(Y_test, Y_pred_adaboost, color='orange')
plt.plot(Y_test, Y_test, color='red', linestyle='--')
plt.title('AdaBoost Regression')
```

```
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



[57]:
```
sns.displot(Y_test - Y_pred_adaboost, kind='kde')
plt.title('Residuals Distribution - AdaBoost Regression')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```

```
C:\Users\sarka\anaconda3\envs\Projects\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[58]:
```
print("Accuracy of AdaBoost Regression model: {:.2f}%".format(r2_adaboost *
 ↪100))
```

```
Accuracy of AdaBoost Regression model: 90.63%
```

[59]:
```
pickle.dump(adaboost_regressor,open('adaboost_regressor.pkl','wb'))
```

1.9 Support Vector Machine

[60]:
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in
c:\users\sarka\anaconda3\envs\projects\lib\site-packages (0.12.0)
Requirement already satisfied: numpy>=1.17.3 in
c:\users\sarka\anaconda3\envs\projects\lib\site-packages (from imbalanced-learn)
(1.26.4)
Requirement already satisfied: scipy>=1.5.0 in
```

```
c:\users\sarka\anaconda3\envs\projects\lib\site-packages (from imbalanced-learn)
(1.12.0)
Requirement already satisfied: scikit-learn>=1.0.2 in
c:\users\sarka\anaconda3\envs\projects\lib\site-packages (from imbalanced-learn)
(1.3.0)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\sarka\anaconda3\envs\projects\lib\site-packages (from imbalanced-learn)
(1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\sarka\anaconda3\envs\projects\lib\site-packages (from imbalanced-learn)
(2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

[61]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest, f_regression
from imblearn.over_sampling import RandomOverSampler
```

[62]:
```
svm_regressor = SVR()
```

[63]:
```
svm_regressor.fit(X_train_scaled, Y_train)
```

[63]:
```
SVR()
```

[64]:
```
# # Making predictions on the test data
Y_pred_svm = svm_regressor.predict(X_test_scaled)
```

[65]:
```
# SVM Hyperparameter Tuning
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel':␣
 ↪['linear', 'rbf', 'poly']}
svm_regressor = SVR()
grid_search = GridSearchCV(svm_regressor, param_grid, scoring='r2', cv=5)
grid_search.fit(X_train_scaled, Y_train)
best_params = grid_search.best_params_
print("Best Parameters for SVM:", best_params)

# Feature Selection
selector = SelectKBest(score_func=f_regression, k=7)
X_train_selected = selector.fit_transform(X_train_scaled, Y_train)
X_test_selected = selector.transform(X_test_scaled)

# SVM with best parameters
svm_regressor = SVR(**best_params)
svm_regressor.fit(X_train_selected, Y_train)
Y_pred_svm = svm_regressor.predict(X_test_selected)
r2_svm = metrics.r2_score(Y_test, Y_pred_svm)
print("Accuracy of SVM Regression model after rectifications: {:.2f}%".
 ↪format(r2_svm * 100))
```

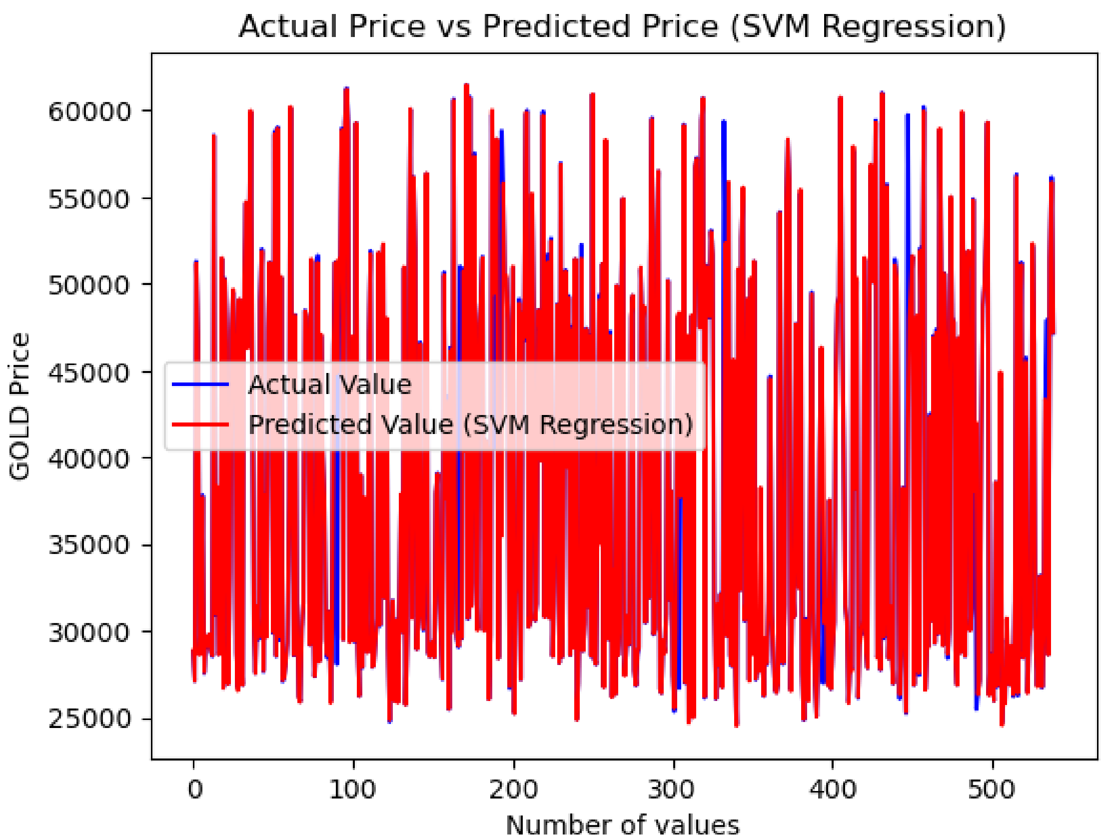
```
# Handle Data Imbalance
oversampler = RandomOverSampler(random_state=42)
X_train_resampled, Y_train_resampled = oversampler.
 ↪fit_resample(X_train_selected, Y_train)

# SVM after handling data imbalance
svm_regressor = SVR(**best_params)
svm_regressor.fit(X_train_resampled, Y_train_resampled)
Y_pred_svm = svm_regressor.predict(X_test_selected)
r2_svm = metrics.r2_score(Y_test, Y_pred_svm)
print("Accuracy of SVM Regression model after handling data imbalance: {:.2f}%".
 ↪format(r2_svm * 100))
```

```
Best Parameters for SVM: {'C': 10, 'gamma': 1, 'kernel': 'linear'}
Accuracy of SVM Regression model after rectifications: 95.63%
Accuracy of SVM Regression model after handling data imbalance: 95.63%
```

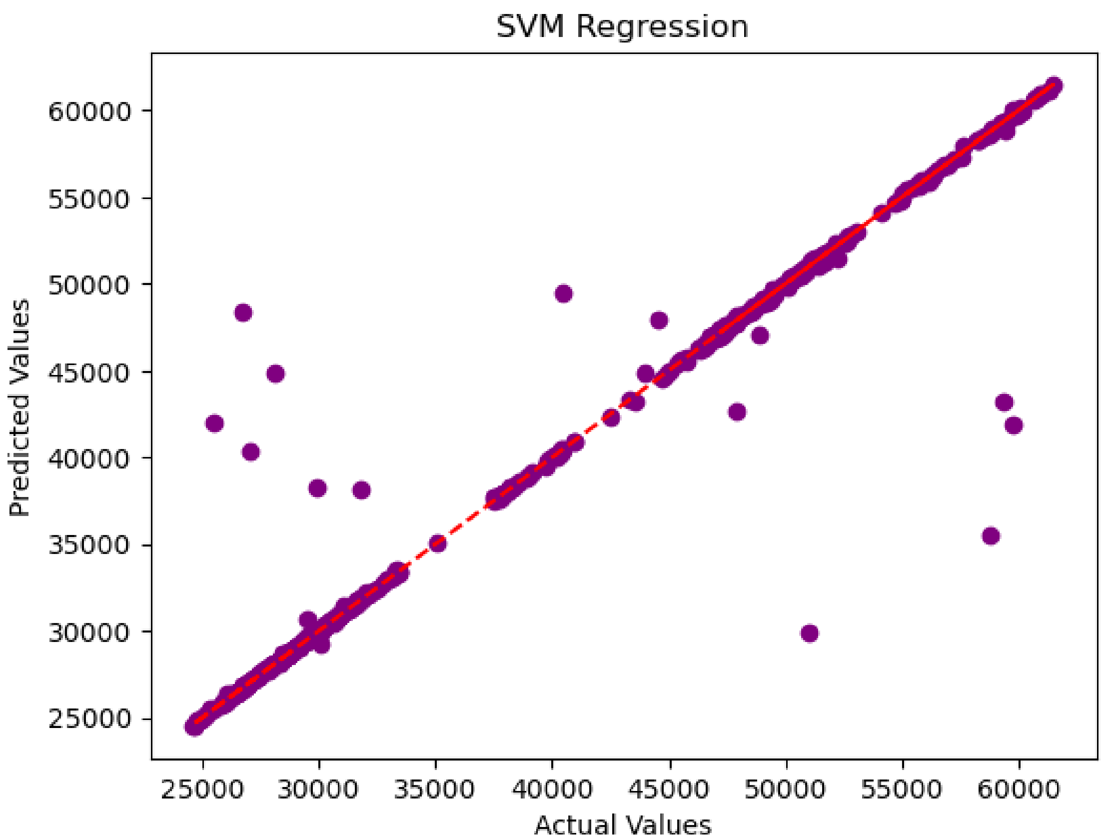
[66]:
```
r2_svm = r2_score(Y_test, Y_pred_svm)
mae_svm = mean_absolute_error(Y_test, Y_pred_svm)
mse_svm = mean_squared_error(Y_test, Y_pred_svm)
print("MAE (SVM Regression): {:.2f}".format(mae_svm))
print("MSE (SVM Regression): {:.2f}".format(mse_svm))
print("R squared error (SVM Regression):", r2_svm)
```

```
MAE (SVM Regression): 411.91
MSE (SVM Regression): 5594331.29
R squared error (SVM Regression): 0.9562949091107288
```

[67]:
```
# Plotting Actual vs Predicted values
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_svm, color='red', label='Predicted Value (SVM Regression)')
plt.title('Actual Price vs Predicted Price (SVM Regression)')
plt.xlabel('Number of values')
plt.ylabel('GOLD Price')
plt.legend()
plt.show()
```

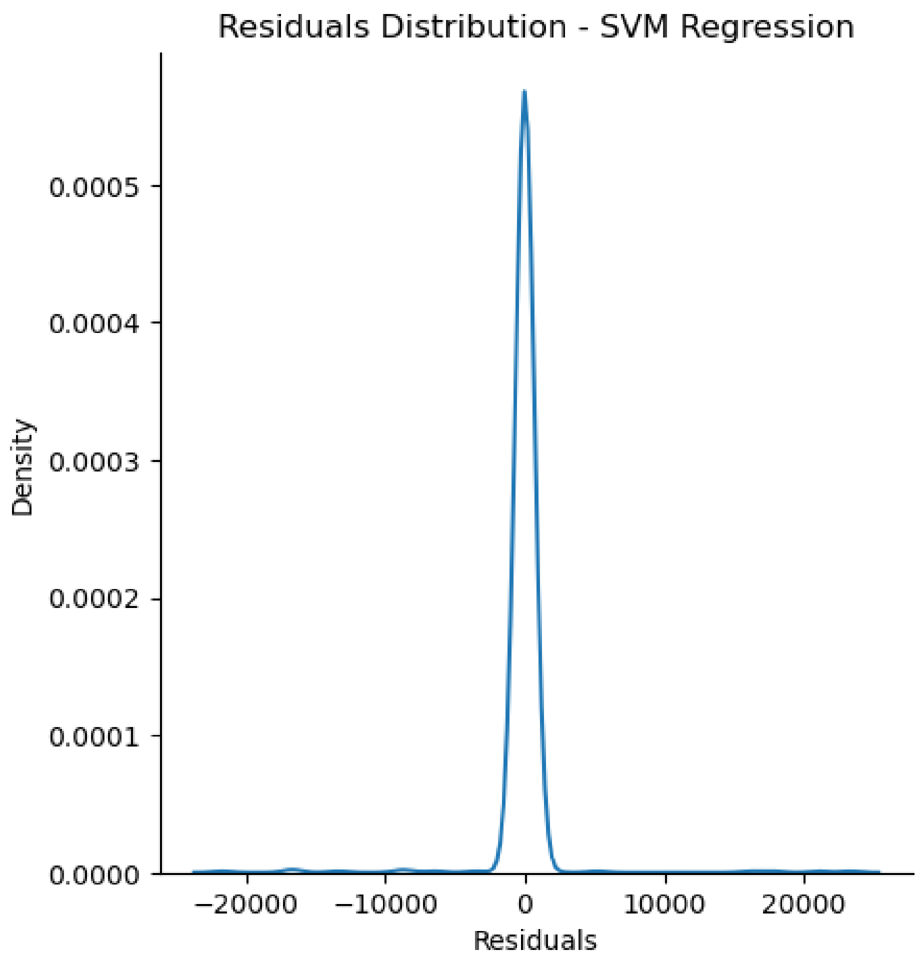


[68]:
```
plt.scatter(Y_test, Y_pred_svm, color='purple')
plt.plot(Y_test, Y_test, color='red', linestyle='--')
plt.title('SVM Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



[69]:
```
sns.displot(Y_test - Y_pred_svm, kind='kde')
plt.title('Residuals Distribution - SVM Regression')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```

```
C:\Users\sarka\anaconda3\envs\Projects\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[70]:
```
print("Accuracy of AdaBoost Regression model: {:.2f}%".format(r2_svm * 100))
```

```
Accuracy of AdaBoost Regression model: 95.63%
```

[71]:
```
pickle.dump(svm_regressor,open('svm_regressor.pkl','wb'))
```

1.10 GRADIENT BOOST REGRESSION

[ ]:
```

```

[72]:
```
# Instantiating the Gradient Boosting Regression model
gradient_boost_regressor = GradientBoostingRegressor()
```

[73]:
```
gradient_boost_regressor.fit(X_train, Y_train)
```

[73]:
```
GradientBoostingRegressor()
```

[74]:

```python
Y_pred_gradient_boost = gradient_boost_regressor.predict(X_test)
```

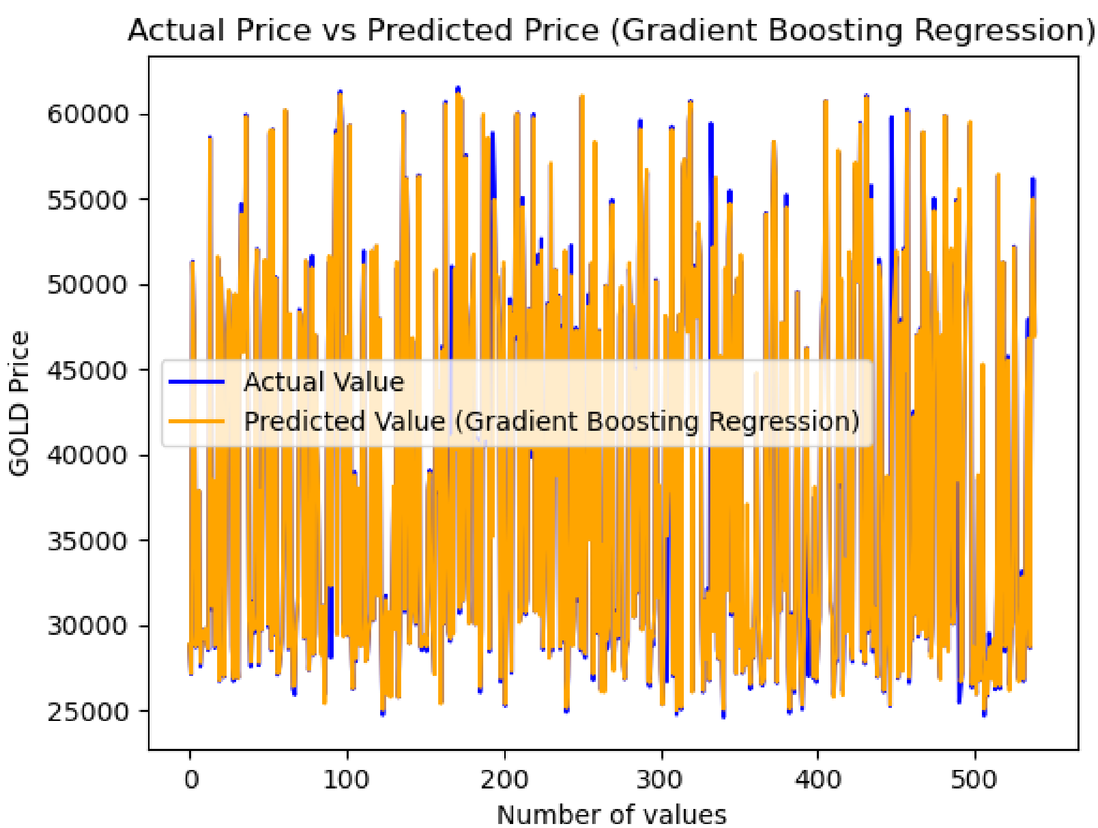
[75]:

```python
r2_gradient_boost = metrics.r2_score(Y_test, Y_pred_gradient_boost)
Y_pred_gradient_boost = gradient_boost_regressor.predict(X_test)
mae_gradient_boost = mean_absolute_error(Y_test, Y_pred_gradient_boost)
mse_gradient_boost = mean_squared_error(Y_test, Y_pred_gradient_boost)
print("MAE (Gradient Boosting Regression): {:.2f}".format(mae_gradient_boost))
print("MSE (Gradient Boosting Regression): {:.2f}".format(mse_gradient_boost))
print("R squared error (Gradient Boosting Regression):", r2_gradient_boost)
```

```
MAE (Gradient Boosting Regression): 490.27
MSE (Gradient Boosting Regression): 5674536.85
R squared error (Gradient Boosting Regression): 0.9556683121389324
```

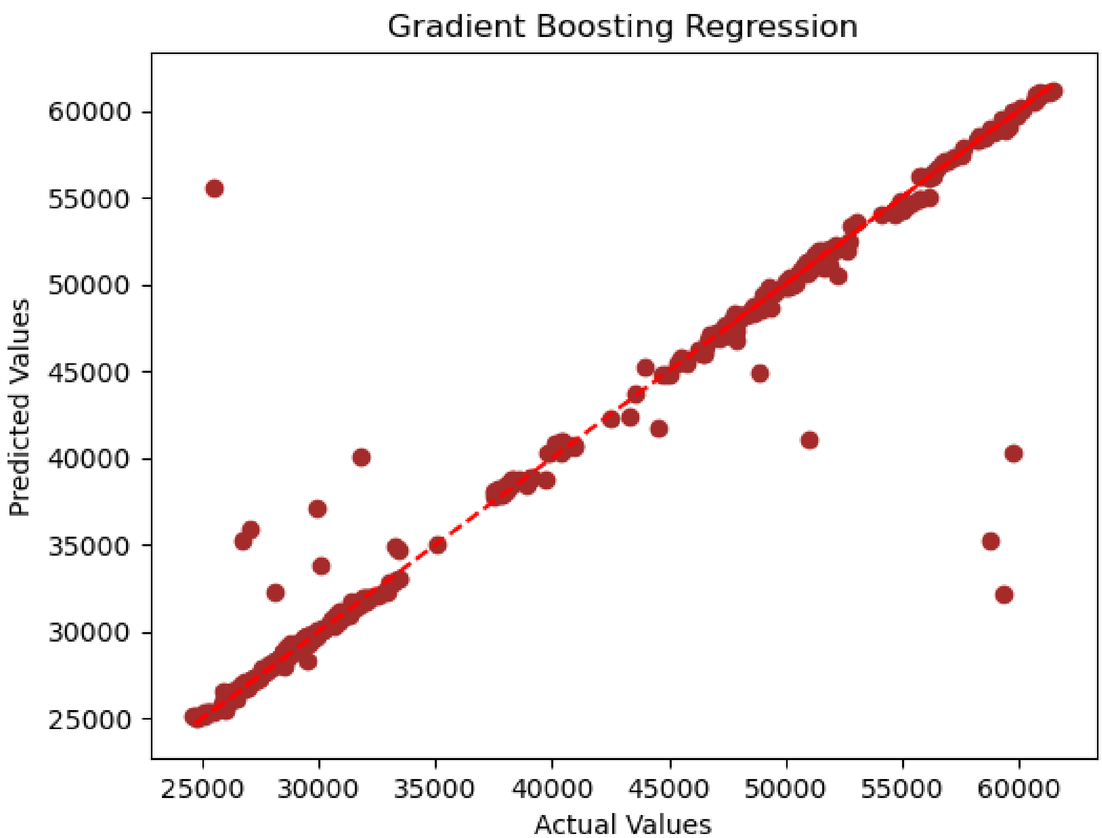
[76]:

```python
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_gradient_boost, color='orange', label='Predicted Value␣
↪(Gradient Boosting Regression)')
plt.title('Actual Price vs Predicted Price (Gradient Boosting Regression)')
plt.xlabel('Number of values')
plt.ylabel('GOLD Price')
plt.legend()
plt.show()
```



[77]:

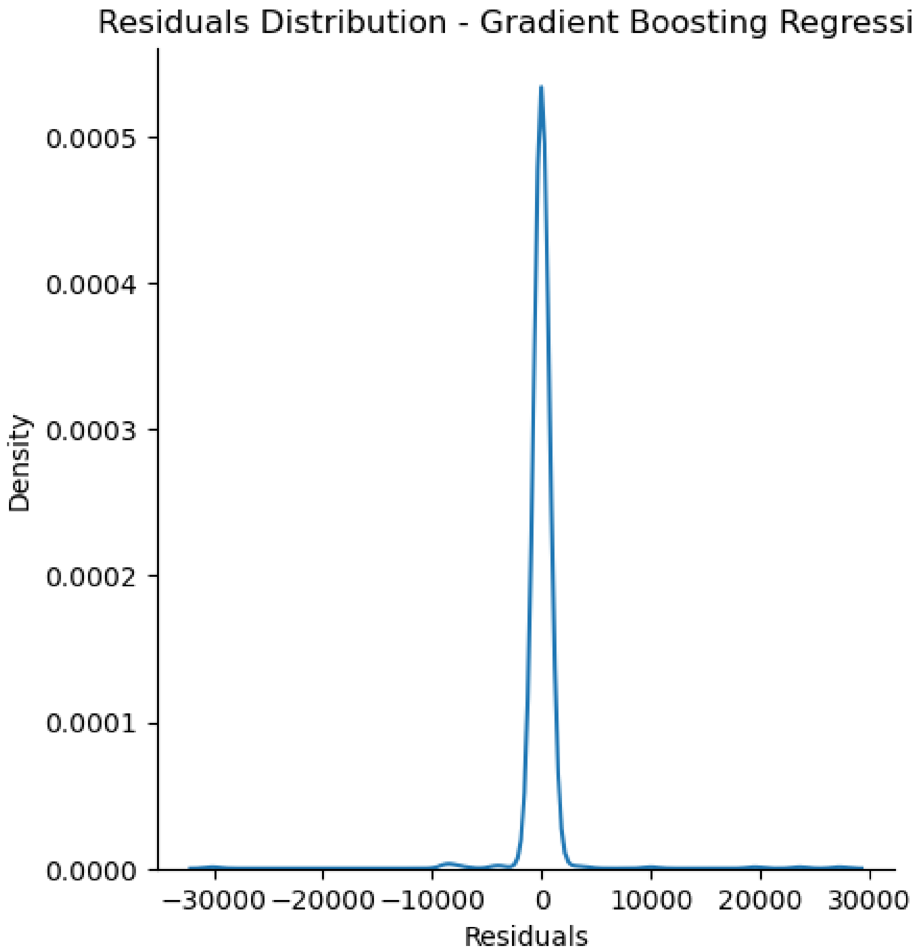
```
plt.scatter(Y_test, Y_pred_gradient_boost, color='brown')
plt.plot(Y_test, Y_test, color='red', linestyle='--')
plt.title('Gradient Boosting Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



[78]:

```
sns.displot(Y_test - Y_pred_gradient_boost, kind='kde')
plt.title('Residuals Distribution - Gradient Boosting Regression')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```

```
C:\Users\sarka\anaconda3\envs\Projects\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[79]:
```
print("Accuracy of Gradient Boost Regression model: {:.2f}%".
  ↪format(r2_gradient_boost * 100))
```

```
Accuracy of Gradient Boost Regression model: 95.57%
```

[80]:
```
pickle.dump(gradient_boost_regressor,open('gradient_boost_regressor.pkl','wb'))
```

2 RESULT SUMMARY

[81]:
```
import pandas as pd

# Create a dictionary to store the metrics of each regression model
metrics_data = {
    'Regression': ['Linear', 'Random Forest', 'XGBoost', 'AdaBoost', 'SVM',␣
  ↪'Gradient Boost'],
    'MAE': [mae_lr, mae_rf, mae_xgb, mae_adaboost, mae_svm, mae_gradient_boost],
```

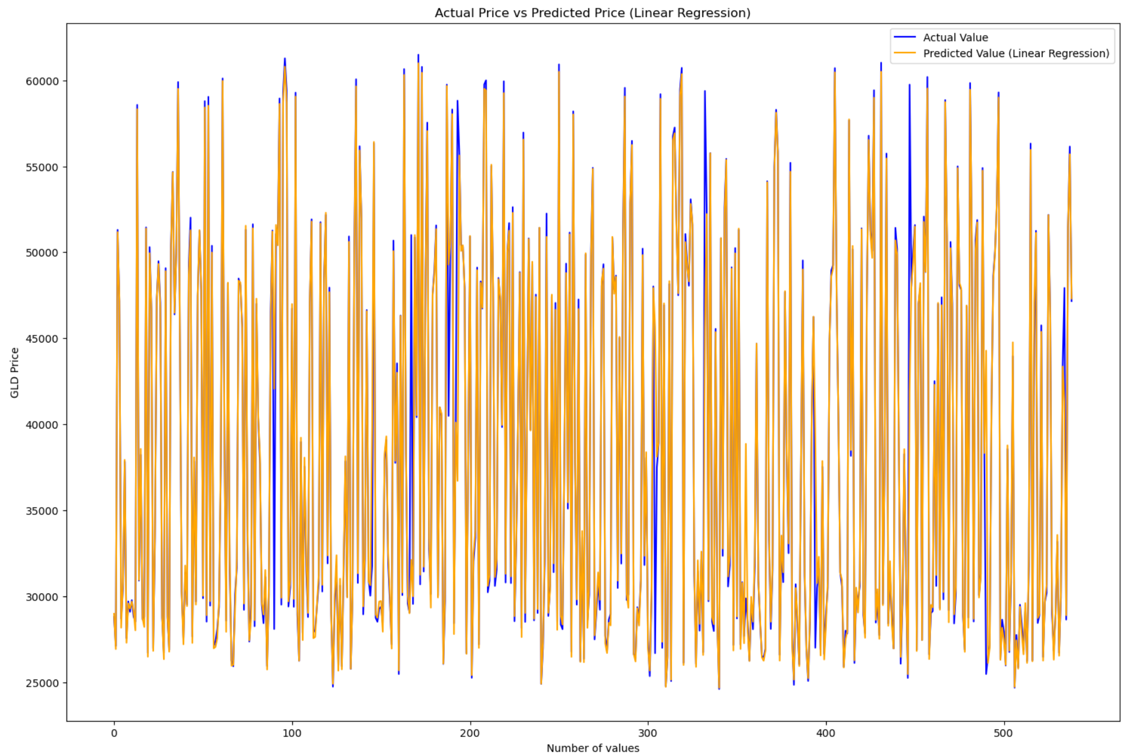
```
    'MSE': [mse_lr, mse_rf, mse_xgb, mse_adaboost, mse_svm, mse_gradient_boost],
    'R2': [error_score, error_scoreR, r2_xgb, r2_adaboost, r2_svm,␣
 ↪r2_gradient_boost],
    'Accuracy': [error_score * 100, error_scoreR * 100, r2_xgb * 100,␣
 ↪r2_adaboost * 100, r2_svm * 100, r2_gradient_boost * 100]
}

# Create a DataFrame from the dictionary
metrics_df = pd.DataFrame(metrics_data)

# Display the DataFrame
print(metrics_df)
```

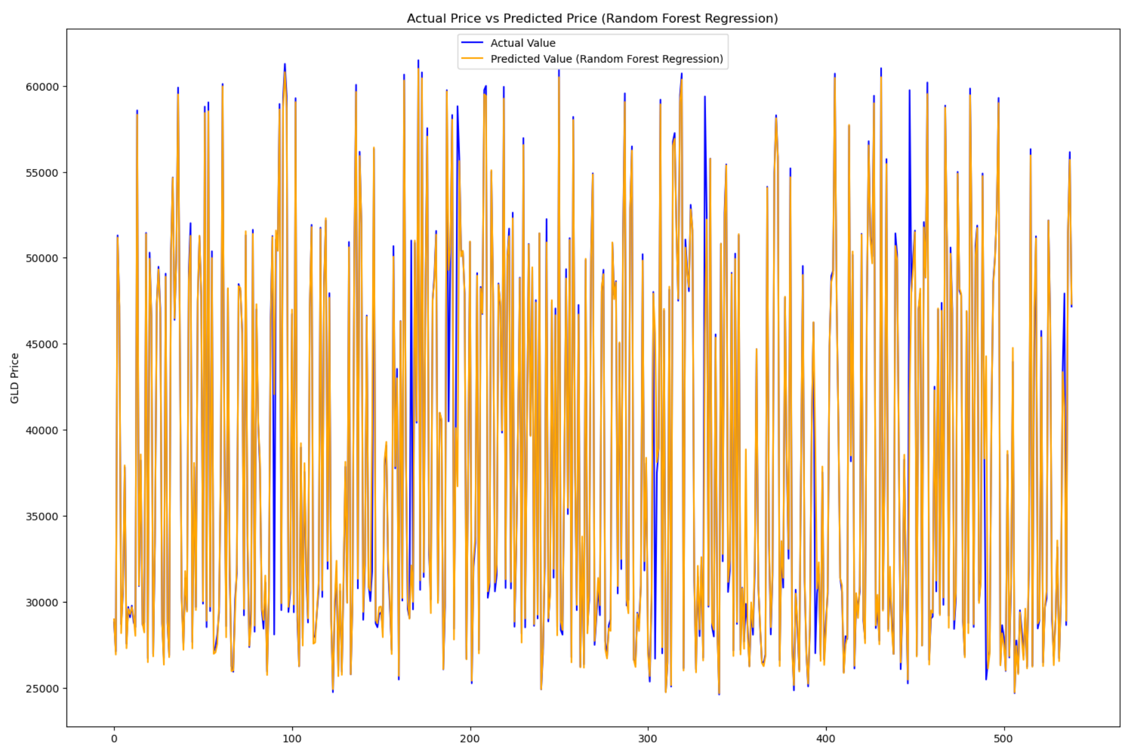
```
       Regression          MAE           MSE        R2   Accuracy
0          Linear   599.701140  5.098138e+06  0.960171  96.017136
1   Random Forest   440.977551  5.208386e+06  0.959310  95.931006
2         XGBoost   529.312616  6.631200e+06  0.948194  94.819449
3        AdaBoost  2457.561945  1.199081e+07  0.906323  90.632314
4             SVM   411.913631  5.594331e+06  0.956295  95.629491
5  Gradient Boost   490.268889  5.674537e+06  0.955668  95.566831
```

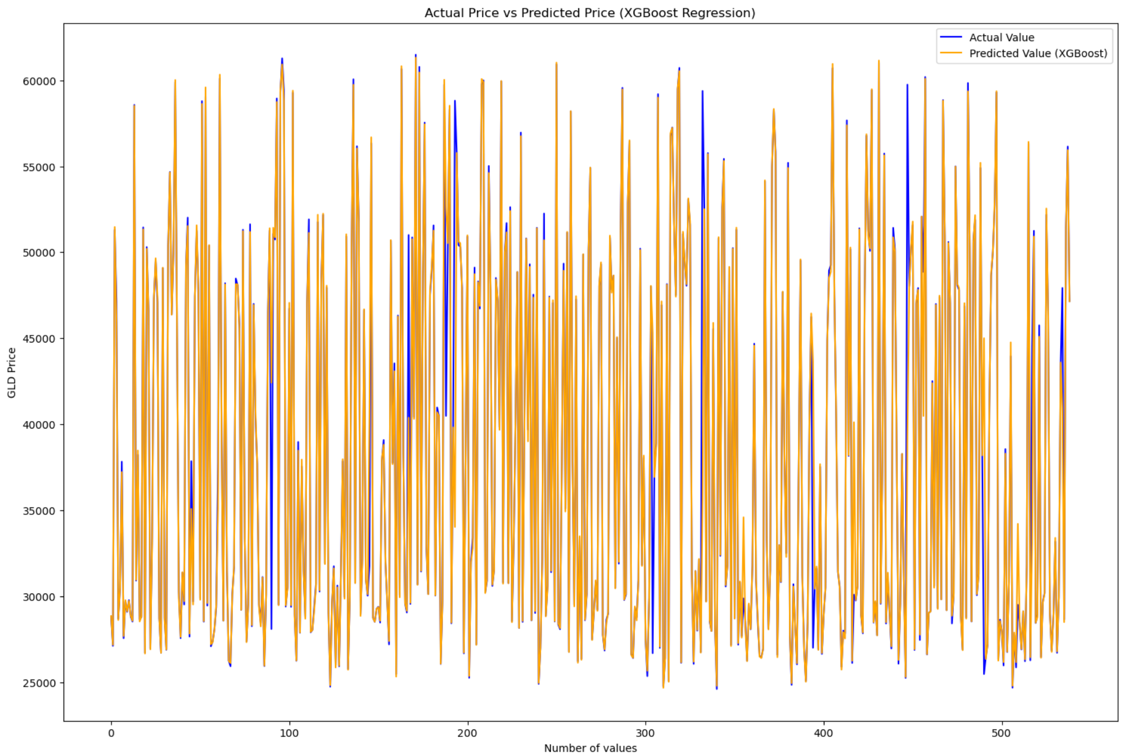
2.1 PLOT

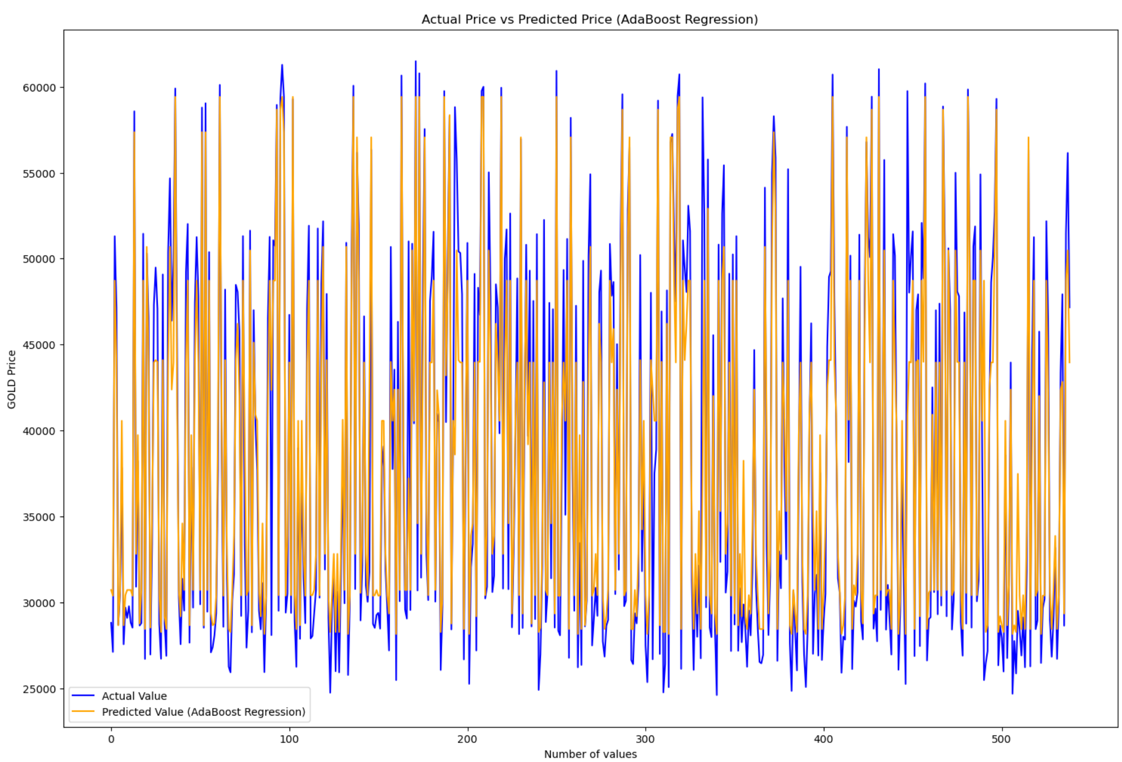
[82]:
```
plt.figure(figsize=(18, 12))



plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(test_data_prediction, color='orange', label='Predicted Value (Linear␣
 ↪Regression)')
plt.title('Actual Price vs Predicted Price (Linear Regression)')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()



plt.figure(figsize=(18, 12))
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(test_data_prediction, color='orange', label='Predicted Value (Random␣
 ↪Forest Regression)')
plt.title('Actual Price vs Predicted Price (Random Forest Regression)')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```

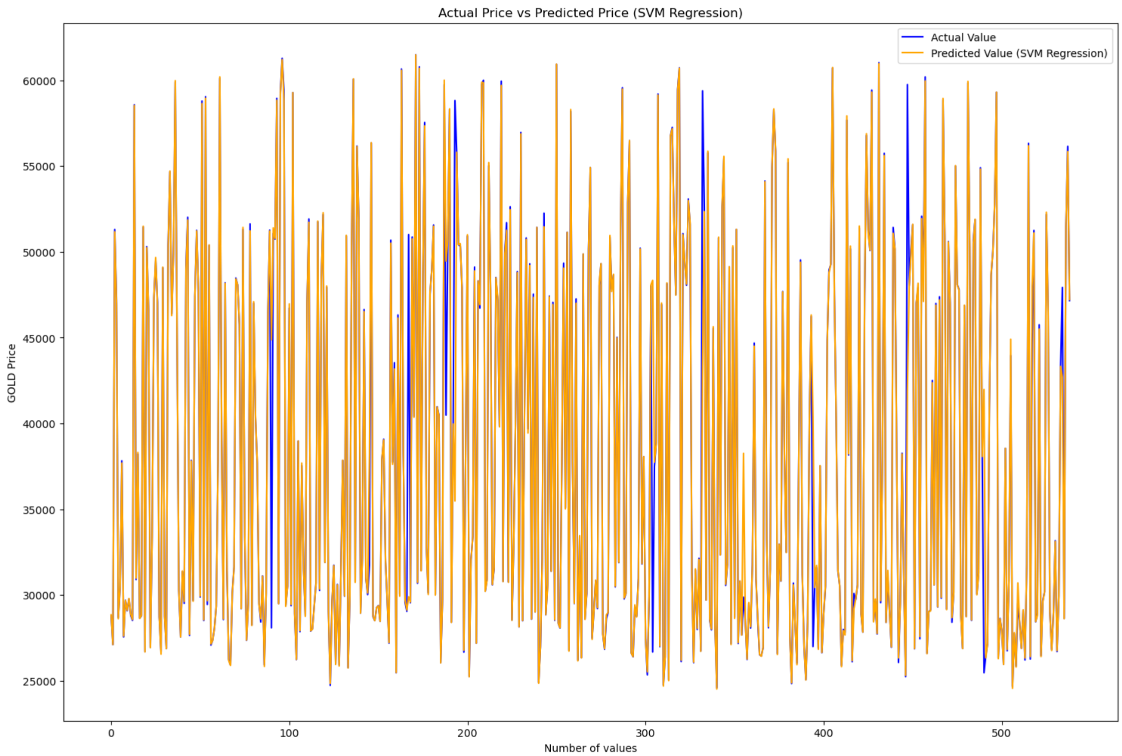
```
plt.figure(figsize=(18, 12))
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_xgb, color='orange', label='Predicted Value (XGBoost)')
plt.title('Actual Price vs Predicted Price (XGBoost Regression)')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()


plt.figure(figsize=(18, 12))
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_adaboost, color='orange', label='Predicted Value (AdaBoost␣
↪Regression)')
plt.title('Actual Price vs Predicted Price (AdaBoost Regression)')
plt.xlabel('Number of values')
plt.ylabel('GOLD Price')
plt.legend()
plt.show()


plt.figure(figsize=(18, 12))
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_svm, color='orange', label='Predicted Value (SVM Regression)')
plt.title('Actual Price vs Predicted Price (SVM Regression)')
plt.xlabel('Number of values')
plt.ylabel('GOLD Price')
plt.legend()
plt.show()


plt.figure(figsize=(18, 12))
plt.plot(Y_test.values, color='blue', label='Actual Value')
plt.plot(Y_pred_gradient_boost, color='orange', label='Predicted Value␣
↪(Gradient Boosting Regression)')
plt.title('Actual Price vs Predicted Price (Gradient Boosting Regression)')
plt.xlabel('Number of values')
plt.ylabel('GOLD Price')
plt.legend()
plt.show()
```

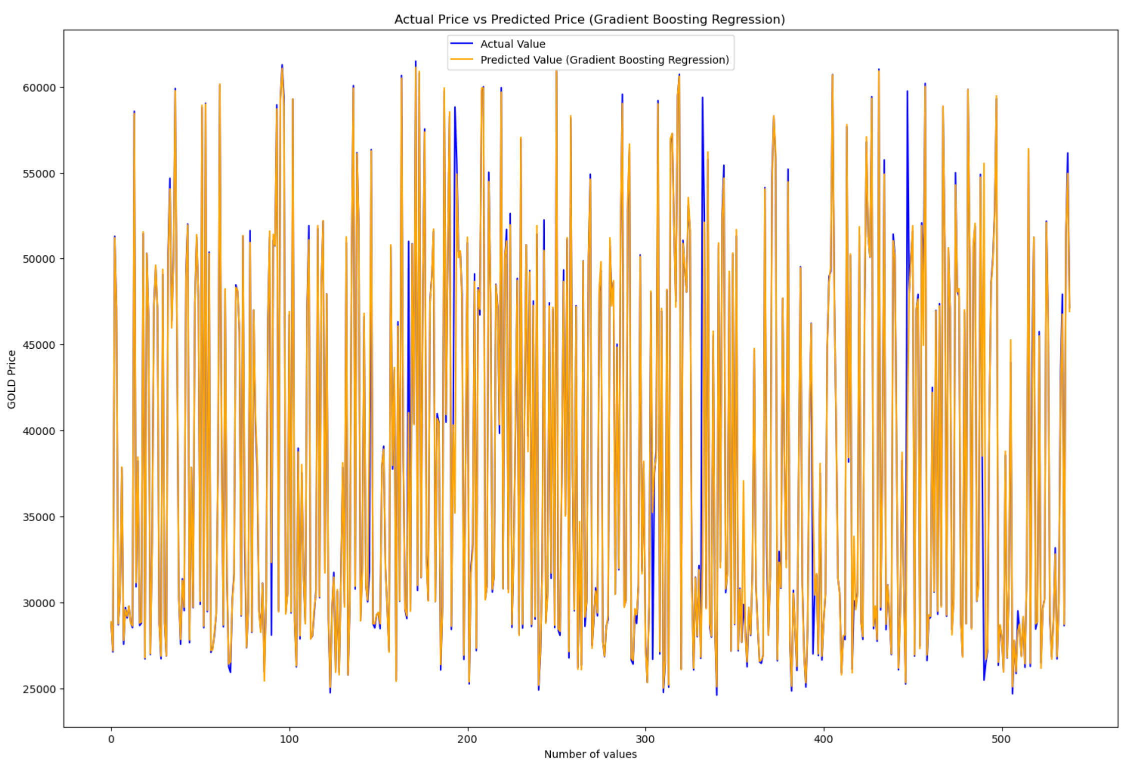

Actual Price vs Predicted Price (Linear Regression)
Actual Value
Predicted Value (Linear Regression)
GLD Price
Number of values
25000
30000
35000
40000
45000
50000
55000
60000
0
100
200
300
400
500


Actual Price vs Predicted Price (Random Forest Regression)
Actual Value
Predicted Value (Random Forest Regression)
GLD Price
Number of values
25000
30000
35000
40000
45000
50000
55000
60000
0
100
200
300
400
500


Actual Price vs Predicted Price (XGBoost Regression)
Actual Value
Predicted Value (XGBoost)
GLD Price
60000
55000
50000
45000
40000
35000
30000
25000
0
100
200
300
400
500
Number of values


Actual Price vs Predicted Price (AdaBoost Regression)
Actual Value
Predicted Value (AdaBoost Regression)
GOLD Price
60000
55000
50000
45000
40000
35000
30000
25000
0
100
200
300
400
500
Number of values

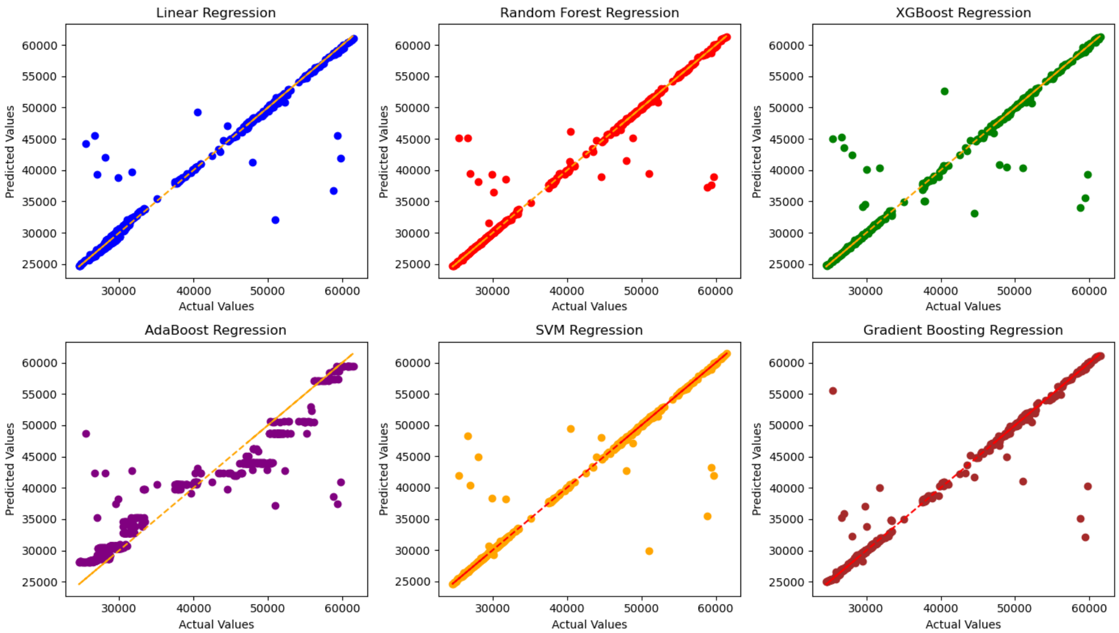

Actual Price vs Predicted Price (SVM Regression)
Actual Value
Predicted Value (SVM Regression)
GOLD Price
Number of values
60000
55000
50000
45000
40000
35000
30000
25000
0
100
200
300
400
500


Actual Price vs Predicted Price (Gradient Boosting Regression)
Actual Value
Predicted Value (Gradient Boosting Regression)
GOLD Price
Number of values
60000
55000
50000
45000
40000
35000
30000
25000
0
100
200
300
400
500

2.2 SCATTERPLOT

[83]:
```
# Create subplots for scatterplots of each regression model
plt.figure(figsize=(14, 8))

# Linear Regression Scatterplot
plt.subplot(2, 3, 1)
plt.scatter(Y_test, test_data_prediction, color='blue')
plt.plot(Y_test, Y_test, color='orange', linestyle='--')
plt.title('Linear Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# Random Forest Regression Scatterplot
plt.subplot(2, 3, 2)
plt.scatter(Y_test, test_data_predictionR, color='red')
plt.plot(Y_test, Y_test, color='orange', linestyle='--')
plt.title('Random Forest Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# XGBoost Regression Scatterplot
plt.subplot(2, 3, 3)
plt.scatter(Y_test, Y_pred_xgb, color='green')
plt.plot(Y_test, Y_test, color='orange', linestyle='--')
plt.title('XGBoost Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# AdaBoost Regression Scatterplot
plt.subplot(2, 3, 4)
plt.scatter(Y_test, Y_pred_adaboost, color='purple')
plt.plot(Y_test, Y_test, color='orange', linestyle='--')
plt.title('AdaBoost Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# SVM Regression Scatterplot
plt.subplot(2, 3, 5)
plt.scatter(Y_test, Y_pred_svm, color='orange')
plt.plot(Y_test, Y_test, color='red', linestyle='--')
plt.title('SVM Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# Gradient Boosting Regression Scatterplot
plt.subplot(2, 3, 6)
plt.scatter(Y_test, Y_pred_gradient_boost, color='brown')
```

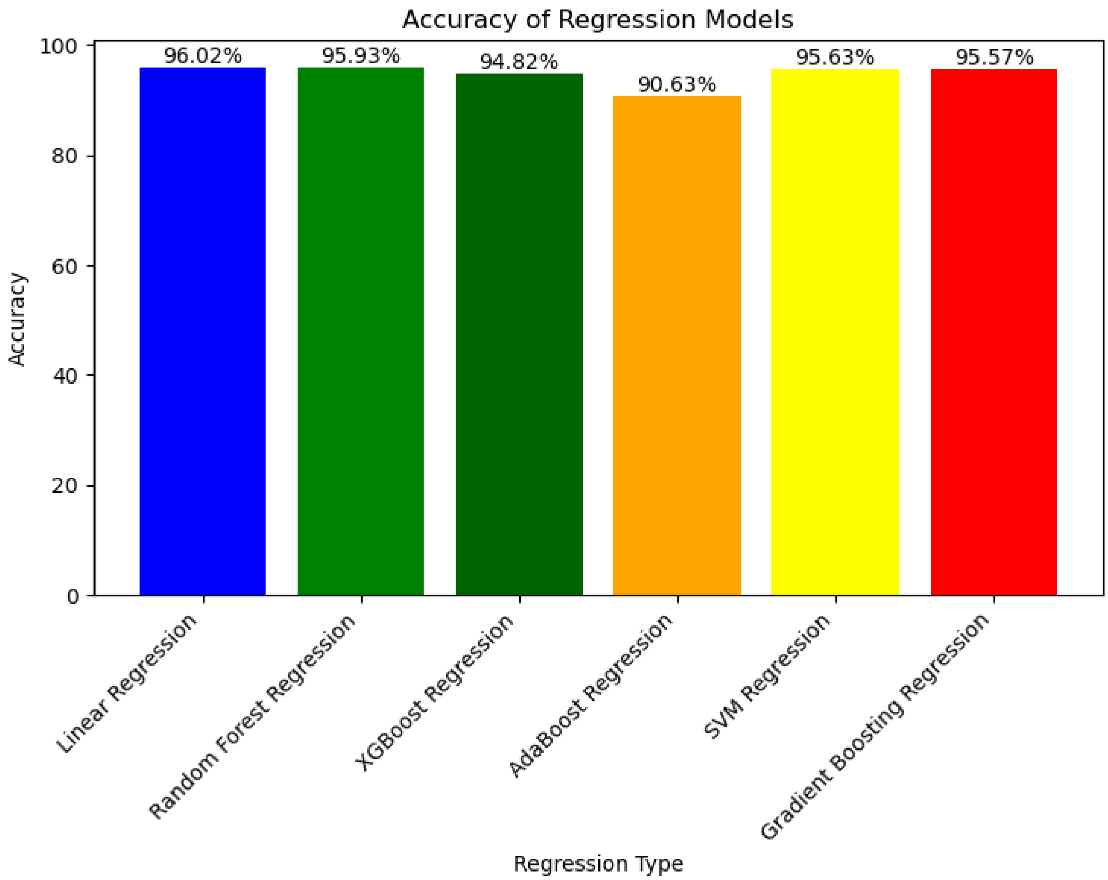
```
plt.plot(Y_test, Y_test, color='red', linestyle='--')
plt.title('Gradient Boosting Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

plt.tight_layout()
plt.show()
```



2.3 BAR GRAPH

[84]:
```
import matplotlib.pyplot as plt

# Data
models = ['Linear Regression', 'Random Forest Regression', 'XGBoost Regression',
          'AdaBoost Regression', 'SVM Regression', 'Gradient Boosting␣
 ↪Regression']
accuracy = [error_score * 100, error_scoreR * 100, r2_xgb * 100, r2_adaboost *␣
 ↪100, r2_svm * 100, r2_gradient_boost * 100]

# Define colors for each bar
colors = ['blue', 'green', 'darkgreen', 'orange', 'yellow', 'red']

# Create bar plot
plt.figure(figsize=(7.5, 6))

bars = plt.bar(models, accuracy, color=colors)
```

```
plt.xlabel('Regression Type')
plt.ylabel('Accuracy')
plt.title('Accuracy of Regression Models')
plt.xticks(rotation=45, ha='right')

# Annotate each bar with accuracy percentage
for bar, acc in zip(bars, accuracy):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{acc:.2f}%',
 ↪ha='center', va='bottom')

plt.tight_layout()

# Show plot
plt.show()
```



[ ]:

[ ]: