GOLD PRICE PREDICTION

Rutik Ingale

250840550020

—

Faculty

—

Dr. Kiran Waghmare

Problem Statement and Objective

The price of gold changes daily and depends on many financial factors. Predicting these prices manually is difficult and inaccurate. The goal of this project is to develop a machine learning model that can accurately predict future gold prices using historical financial data. The project uses regression techniques to estimate the exact gold price and classification algorithms to determine whether the price will be high or low. The model helps investors and traders make informed decisions.

DataSet Information

1. Dataset Name
Gold Price Dataset (gld_price_data.csv)

2. Source of Dataset
Kaggle.com

3. Number of Records
- Rows: Approximately 2290+
- Columns: 6 main features +

4. Dataset Columns and Their Meaning

1. Date
Type: Object / String
- Represents the trading date for the recorded values.
- Useful for time-series but removed for model training.

2. SPX (S&P 500 Index)
Type: Float
- Tracks performance of top U.S. companies.
- Used because gold price often moves opposite to stock market trends.

3. USO (Crude Oil ETF Prices)
Type: Float
- Represents fluctuations in crude oil prices.
- Gold price is indirectly influenced by oil market trends.

4. SLV (Silver ETF Prices)
Type: Float
- Represents the price of silver.
- Gold and silver prices have strong positive correlation.

5. EUR/USD (Currency Exchange Rate)
Type: Float
- Indicates the strength of US Dollar vs Euro.
- Gold is globally traded in USD; thus currency value impacts gold prices.

6. GLD (Gold ETF Price) → Target for Regression
Type: Float
- Represents the actual price of gold.
- This is the main value you predict using regression.

7. Price_Class (Derived Column) → Target for Classification
Type: Integer (0 or 1)

- Created by splitting GLD price using the median:
  - **1 = High price**
  - **0 = Low price**
- Used for classification models (Logistic Regression, Decision Tree, XGBoost)

5. Data Type Summary

Column	Data Type	Use
Date	Object	Dropped before training
SPX	Float	Feature
USO	Float	Feature
SLV	Float	Feature
EUR/USD	Float	Feature
GLD	Float	Regression Target
Price_Class	Integer	Classification Target

6. Data Characteristics

- No missing values (checked using isnull().sum())
- Mostly numerical values → suitable for ML models
- Contains financial indicators highly correlated with gold price (checked using correlation heatmap)
- Shows positive correlation between:
  - **GLD ↔ SLV**
  - **GLD ↔ EUR/USD**
- Dataset is clean and ready for ML without additional preprocessing

7. Why This Dataset is Suitable for ML?
Numeric values :- ideal for ML algorithms
High correlation between features :- strong predictive power
Includes multiple financial indicators :- more accurate predictions
Sufficient size for training ML models
Works for both Regression & Classification tasks

Algorithms used and it's justification

A. Regression Algorithm
1. Random Forest Regressor
Why This Algorithm?

- Gold price prediction is a **complex financial problem** with many non-linear relations.
- Random Forest is an **ensemble algorithm** (uses many decision trees).
- It learns patterns from multiple trees and averages them to make a final prediction.

Justification
Handles **non-linear data** extremely well
Reduces **overfitting**
Works effectively with **large datasets**
Gives **high accuracy** compared to single models
Very robust for financial forecasting
Why not Linear Regression?

- Gold data is **not linear**
- Relationship between SPX, oil, silver, currency → GOLD is complex
- Linear regression would give lower accuracy

B. Classification Algorithms

I created a new target variable **Price_Class**
(High = 1, Low = 0).

2. Logistic Regression
Why This Algorithm?

- Best **baseline model** for binary classification.
- Works well when the relationship is somewhat linear.

Justification
Easy to interpret
Fast and efficient
Good starting point for comparison
Helps check if advanced models are needed

3. Decision Tree Classifier
Why This Algorithm?

- Splits data into decision rules.
- Captures **non-linear patterns** very well.

Justification

Understandable visual model (tree plot)
Works well with mixed numerical data
Handles non-linearity better than logistic regression
Can show feature importance

Reason to include

It helps you compare how "simple tree-based models" perform versus advanced boosted models.

4. XGBoost Classifier

Why This Algorithm?

- Most powerful tree-based boosting algorithm.
- Used widely in finance forecasting & Kaggle competitions.
- Boosts weak learners into a strong model.

Justification

High accuracy, often best among all classifiers
Handles noise and outliers well
Fast and optimized model
Learns complex patterns better than Decision Trees
Uses regularization to prevent overfitting

Preprocessing steps

Preprocessing Steps
Preprocessing is an essential stage in the development of any machine learning model. It ensures that the dataset is clean, consistent, and suitable for training predictive models. The following preprocessing steps were applied in this project:

1. Data Loading
The dataset gld_price_data.csv was loaded using the Pandas library. This step converts the raw CSV file into a structured DataFrame for further analysis.

2. Initial Data Inspection
Several commands (head(), tail(), shape, info(), describe()) were used to understand the dataset structure.
This step helped identify:

- The number of rows and columns
- Data types of each column
- Basic statistical properties
- Absence of missing values

3. Removal of Unnecessary Columns
The Date column was removed because it is a non-numeric feature and does not contribute directly to the prediction process. For regression, the GLD column was kept as the target variable. For classification, both Date and GLD were excluded from the feature set.
4. Feature and Target Separation
For Regression:

- Features (X): SPX, USO, SLV, EUR/USD
- Target (y): GLD

For Classification:
A new categorical variable called Price_Class was created by comparing each GLD value with the median price:

- 1 → High Price
- 0 → Low Price

The final classification features included SPX, USO, SLV and EUR/USD, and the target variable was Price_Class.

5. Exploratory Data Analysis (EDA)
To understand the relationship between variables, the following visual analyses were performed:

- Correlation Heatmap: Shows the interdependence between features and the gold price.
- Distribution Plot: Illustrates how the GLD price is distributed across the dataset.

These analyses helped identify important features and detect any skewness or irregular patterns.

6. Train–Test Split
The dataset was divided into:

- 80% Training Data
- 20% Testing Data

This ensures that the model is trained on the majority of the data while still being evaluated on unseen data to check its performance.
The split was done using train_test_split() with a fixed random state for reproducibility.

7. Creation of Classification Target Variable
To perform classification, the continuous GLD values were converted into binary categories (high or low).
This step enabled the use of classification algorithms such as Logistic Regression, Decision Tree, and XGBoost.

Code Snippets

```
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
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor, XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

a Collection and Processing

```
gold_data=pd.read_csv('gld_price_data.csv')
```

```
gold_data.head()
```

	Date	SPX	GLD	USO	SLV	EUR/USD
0	1/2/2008	1447.160034	84.860001	78.470001	15.180	1.471692
1	1/3/2008	1447.160034	85.570000	78.370003	15.285	1.474491
2	1/4/2008	1411.630005	85.129997	77.309998	15.167	1.475492
3	1/7/2008	1416.180054	84.769997	75.500000	15.053	1.468299
4	1/8/2008	1390.189941	86.779999	76.059998	15.590	1.557099

```
gold_data.tail()
```

	Date	SPX	GLD	USO	SLV	EUR/USD
2285	5/8/2018	2671.919922	124.589996	14.0600	15.5100	1.186789
2286	5/9/2018	2697.790039	124.330002	14.3700	15.5300	1.184722
2287	5/10/2018	2723.070068	125.180000	14.4100	15.7400	1.191753
2288	5/14/2018	2730.129883	124.489998	14.3800	15.5600	1.193118
2289	5/16/2018	2725.780029	122.543800	14.4058	15.4542	1.182033

```
gold_data.shape
```

```
(2290, 6)
```

```
gold_data.info()
```

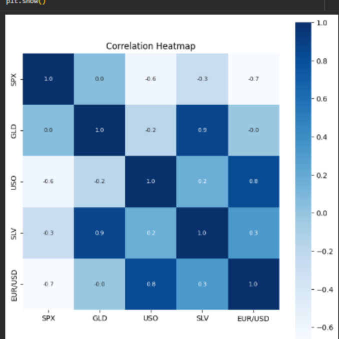
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Date     2290 non-null   object 
 1   SPX      2290 non-null   float64
 2   GLD      2290 non-null   float64
 3   USO      2290 non-null   float64
 4   SLV      2290 non-null   float64
 5   EUR/USD  2290 non-null   float64
dtypes: float64(5), object(1)
memory usage: 107.5+ KB
```

```
gold_data.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

```
correlation = gold_data.drop('Date', axis=1).corr()
```

```
plt.figure(figsize=(8, 8))
sns.heatmap(correlation, cbar=True,square=True,fmt='.1f',annot=True,annot_kws={'size':8},cmap='Blues')
plt.title('Correlation Heatmap')
plt.show()
```




```
#coorelation values of gld
print(correlation['GLD'])
```

```
SPX        0.049345
GLD        1.000000
USO       -0.186360
SLV        0.866632
```

```python
print(correlation['GLD'])
```

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

```python
#checking distribution of gold price
sns.distplot(gold_data['GLD'],color='green')
```

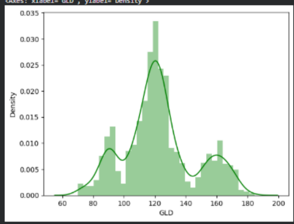
```
/tmp/ipython-input-1130600815.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(gold_data['GLD'],color='green')
<Axes: xlabel='GLD', ylabel='Density'>
```



Splitting the feature and target

```python
x=gold_data.drop(['Date','GLD'],axis=1)
y=gold_data['GLD']
```

```python
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
```

```python
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)
```

Model Train

```python
regressor=RandomForestRegressor(n_estimators=100)
```

```python
#train the model
regressor.fit(x_train,y_train)
```

```
RandomForestRegressor
RandomForestRegressor()
```

model evaluation

```python
#predicton on test data
test_data_prediction=regressor.predict(x_test)
```

```python
print(test_data_prediction)
```

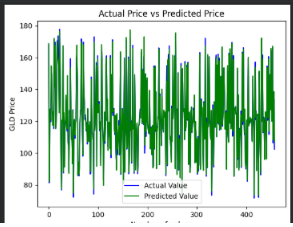
Show hidden output

```python
#r squared error
error_score=metrics.r2_score(y_test,test_data_prediction)
print("R squared error : ",error_score)
```

```
R squared error :  0.9891391526177054
```

```python
y_test=list(y_test)
```

```python
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()
```



```python
median_price = gold_data["GLD"].median()
gold_data["Price_Class"] = gold_data["GLD"].apply(lambda x: 1 if x >= median_price else 0)

Xc = gold_data.drop(["GLD", "Price_Class", "Date"], axis=1)
yc = gold_data["Price_Class"]

Xc_train, Xc_test, yc_train, yc_test = train_test_split(Xc, yc, test_size=0.2, random_state=42)
```

ogistic Regression

```python
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(Xc_train, yc_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```python
pred_lr = log_reg.predict(Xc_test)
print(pred_lr)
```

Show hidden output

```python
acc_lr = accuracy_score(yc_test, pred_lr)
print("Logistic Regression Accuracy:", acc_lr)
print(confusion_matrix(yc_test, pred_lr))
print(classification_report(yc_test, pred_lr))
```

```
Logistic Regression Accuracy: 0.925764192139738
[[221  18]
 [ 16 203]]
              precision    recall  f1-score   support

           0       0.93      0.92      0.93       239
           1       0.92      0.93      0.92       219

    accuracy                           0.93       458
   macro avg       0.93      0.93      0.93       458
weighted avg       0.93      0.93      0.93       458
```

ecision Tree

```python
dtc = DecisionTreeClassifier()
dtc.fit(Xc_train, yc_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```python
pred_dtc = dtc.predict(Xc_test)
```

```python
acc_dtc = accuracy_score(yc_test, pred_dtc)
print("Decision Tree Classifier Accuracy:", acc_dtc)
print(confusion_matrix(yc_test, pred_dtc))
print(classification_report(yc_test, pred_dtc))
```

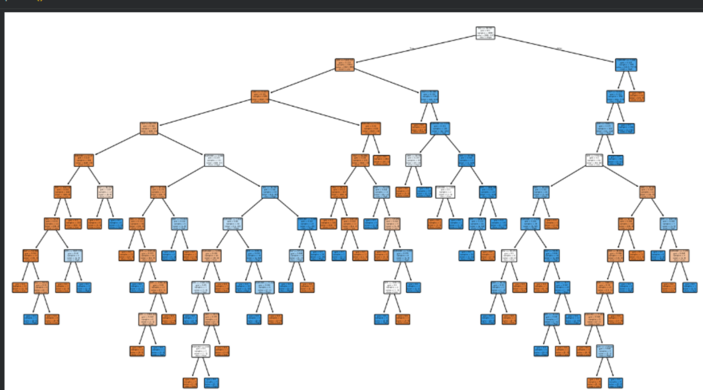

```python
acc_dtc = accuracy_score(yc_test, pred_dtc)
print("Decision Tree Classifier Accuracy:", acc_dtc)
print(confusion_matrix(yc_test, pred_dtc))
print(classification_report(yc_test, pred_dtc))
```


```
Decision Tree Classifier Accuracy: 0.9694323144104804
[[237   2]
 [ 12 207]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       239
           1       0.99      0.95      0.97       219

    accuracy                           0.97       458
   macro avg       0.97      0.97      0.97       458
weighted avg       0.97      0.97      0.97       458
```

```python
from sklearn import tree
plt.figure(figsize=(18,10))
tree.plot_tree(dtc,
               filled=True,
               feature_names=Xc.columns,
               class_names=["Low","High"],
               rounded=True)
plt.show()
```



XG BOOST

```python
xgbc = XGBClassifier()
xgbc.fit(Xc_train, yc_train)
```

```
XGBClassifier
XGBClassifier(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,
              feature_weights=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, ...)
```

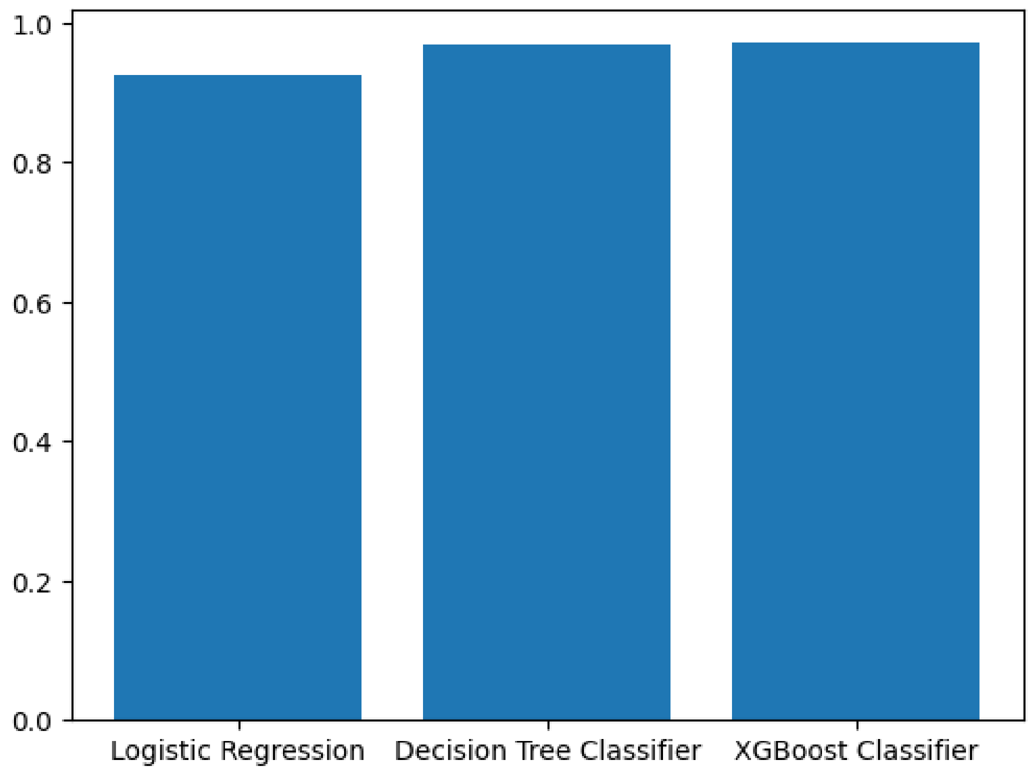
```python
pred_xgbc = xgbc.predict(Xc_test)
```

```python
acc_xgbc = accuracy_score(yc_test, pred_xgbc)
print("XGBoost Classifier Accuracy:", acc_xgbc)
print(confusion_matrix(yc_test, pred_xgbc))
print(classification_report(yc_test, pred_xgbc))
```

```
XGBoost Classifier Accuracy: 0.9716157205240175
[[235   4]
 [  9 210]]
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       239
           1       0.98      0.96      0.97       219

    accuracy                           0.97       458
   macro avg       0.97      0.97      0.97       458
weighted avg       0.97      0.97      0.97       458
```



Accuracy Conclusion

Among the three classification models, XGBoost Classifier achieved the highest accuracy, demonstrating its strength in learning complex financial patterns. Decision Tree provided reasonable accuracy, while Logistic Regression served as a useful baseline but showed lower performance compared to tree-based models.

Conclusion & Future Scope

Conclusion : -

This project successfully demonstrated the use of machine learning models to predict gold prices and classify price trends. The Random Forest Regressor provided high accuracy for predicting actual gold prices, while among the classification models, XGBoost achieved the best performance. The results show that machine learning can effectively capture financial patterns and provide reliable predictions for gold price analysis.

Future Work : -

Future improvements may include adding more economic features like inflation, interest rates, and global market indicators to enhance accuracy. Advanced deep learning models such as LSTM can be used for time-series forecasting. The system can also be expanded into a real-time prediction dashboard using APIs or web applications. Additionally, hyperparameter tuning and ensemble stacking can further improve overall model performance.