A Project Activity Report
Submitted for Artificial Intelligence (UCS411)

TOPIC : GOLD PRICE PREDICTION MODEL

Using Random Forest Regression Algorithm

Submitted by:

Arpit Sagar (102003130)

Rupinderpal Singh (102003167)

Submitted to :

Niyaz Wani

DEPARTMENT OF COMPUTER SCIENCE and ENGINEERING

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,

(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB

INDIA

Jan-June 2022

ABSTRACT:

The Satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance has been a source of inspiration throughout the course of this project. We take this opportunity to express our gratitude to all those who have helped us in this project.

Our Sincere thanks to our guide **Niyaz Wani Sir** for giving us their support and constant encouragement for completion of the project successfully.

INTRODUCTION:

Historically, gold had been used as a form of currency in various parts of the world. In present times, precious metals like gold are held with central banks of all countries to guarantee re-payment of foreign debts, and also to control inflation which results in reflecting the financial strength of the country.

Forecasting rise and fall in the daily gold rates, can help investors to decide when to buy (or sell) the commodity.

We in this project would forecast gold rates using the most comprehensive set of features and would apply various machine learning algorithms for forecasting and compare their results. We also identify the attributes that highly influence the gold rates.

PROJECT OVERVIEW:

Machine learning algorithms are used to train and model the collected data.

In this minor project, we have used the **Random Forest Regression** supervised learning algorithm to predict the gold prices for future using a given data set. It uses **ensemble learning** method for regression.

Using the gold prices dataset we have tried to first train the model and then tested it using predict function .Finding the accuracy percentage and plotting the corresponding graphs we can find the utility of this model in predicting or forecasting gold prices.

The work flow of our project is described as below :

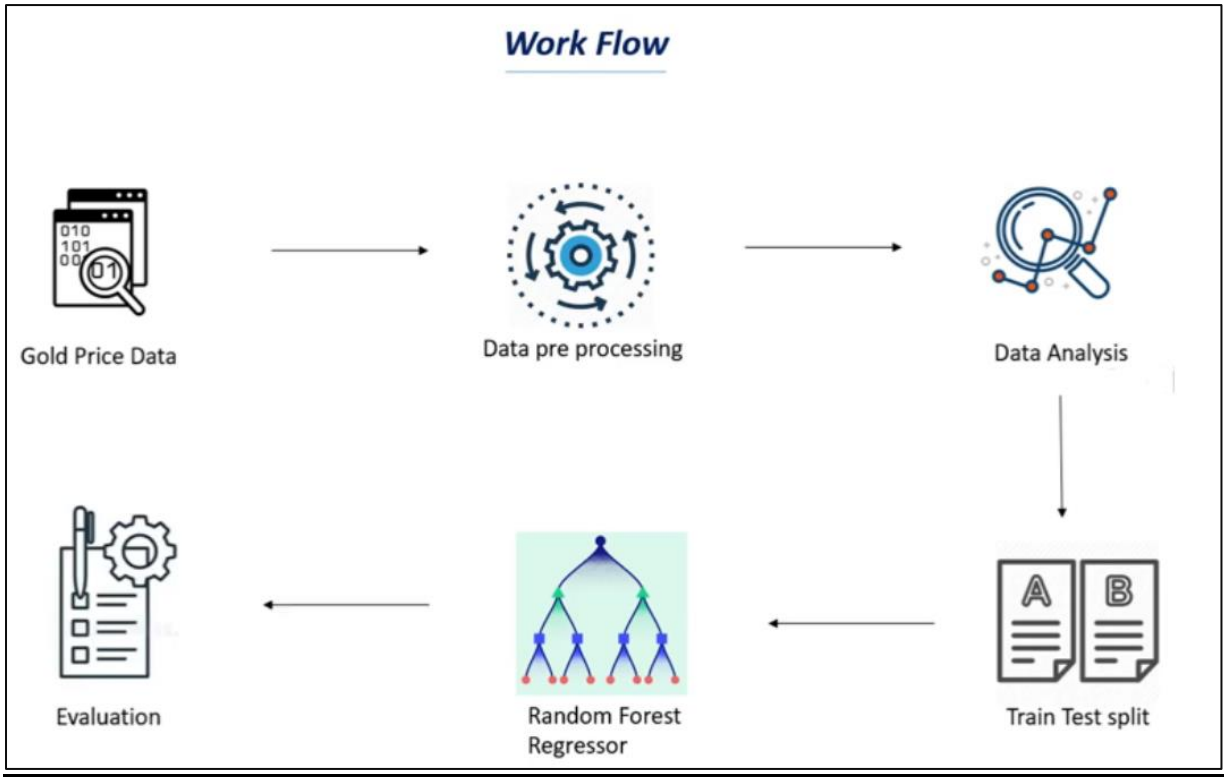
1)Collecting gold price dataset for analysis. Source : Kaggle

2)Performing Data pre-processing

3)Train Test split

4)Prediction using decision tree

5)Evaluation parameters

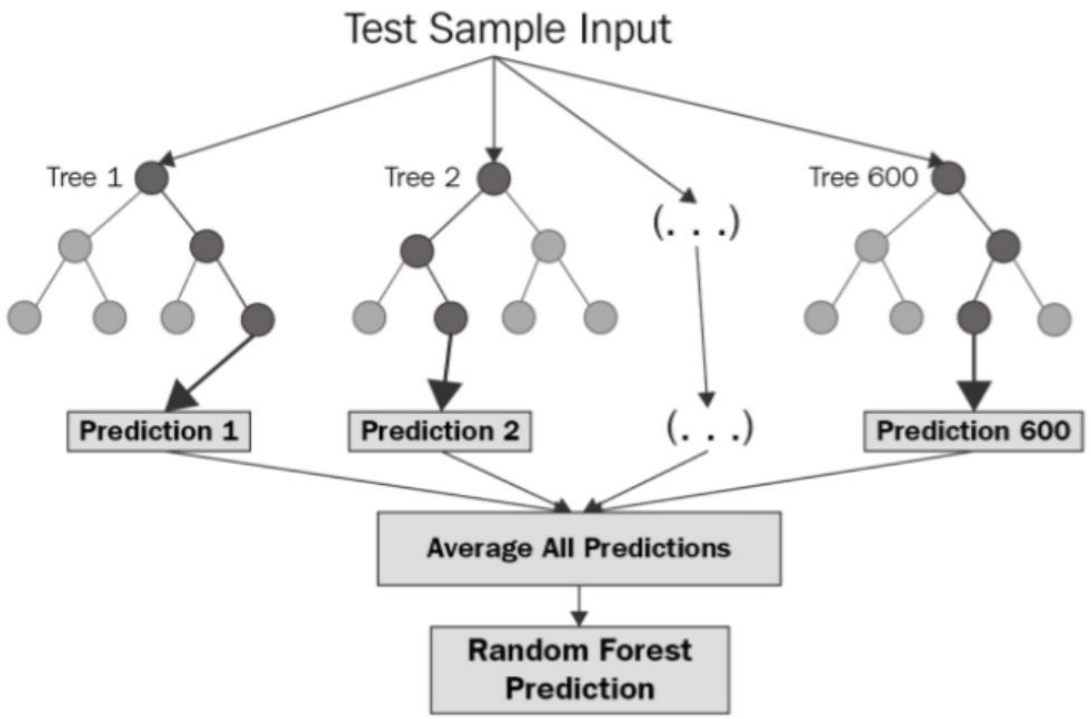


Random Forest Regression Algorithm :

Random Forest is an ensemble learning model useful for solving both regression and classification problems.

It operates using multiple decision trees and makes the final prediction using the bagging method.

To simplify, it builds a 'forest' with multiple decision trees, each trained on different data subsets, and merges the results together to come up with more accurate predictions. **Th**is supervised learning algorithm that uses **ensemble learning** method for regression is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.



The diagram below shows the structure of a Random Forest.The trees run in parallel with no interaction amongst them. The steps followed are:

1)Pick at random *k* data points from the training set.

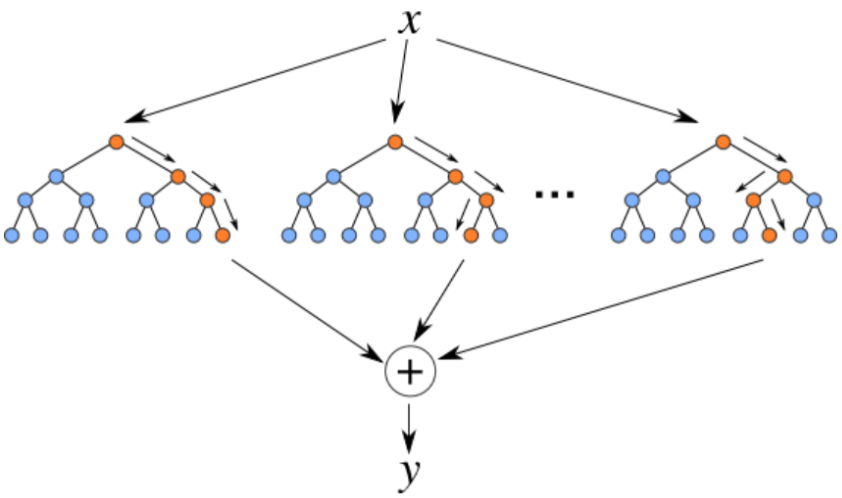
2)Build a decision tree associated to these *k* data points.

3)Choose the number *N* of trees you want to build and repeat steps 1 and 2.

For a new data point, make each one of your *N*-tree trees predict the value of *y* for the data point in question and assign the new data point to the average across all of the predicted *y* values.

A Random Forest Regression model is powerful and accurate. It usually performs great on many problems, including features with non-linear relationships.

Disadvantages, however, include the following: there is no interpretability, overfitting may easily occur, we must choose the number of trees to include in the model.



Building The Model :

- **Importing the dependencies :**

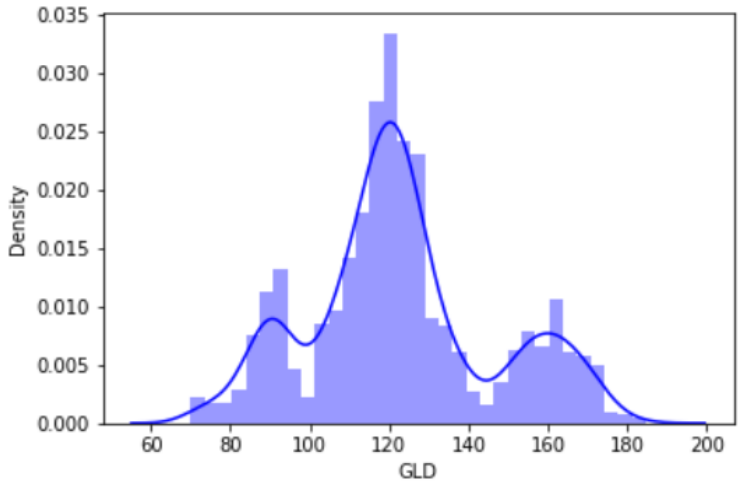
  We have to import the necessary dependencies and libraries in this project like numpy(for linear algebra), pandas(data processing), matplotlib(plotting graphs), sklearn(statistical graphs in python).

- **Data preprocessing and collection:**

  This includes reading csv data ,data cleaning,checking missing values and statistical measures.

  Plotting the distribution of gold prices :

```
# checking the distribution of the GLD Price
sns.distplot(gold_data['GLD'],color='blue')
```



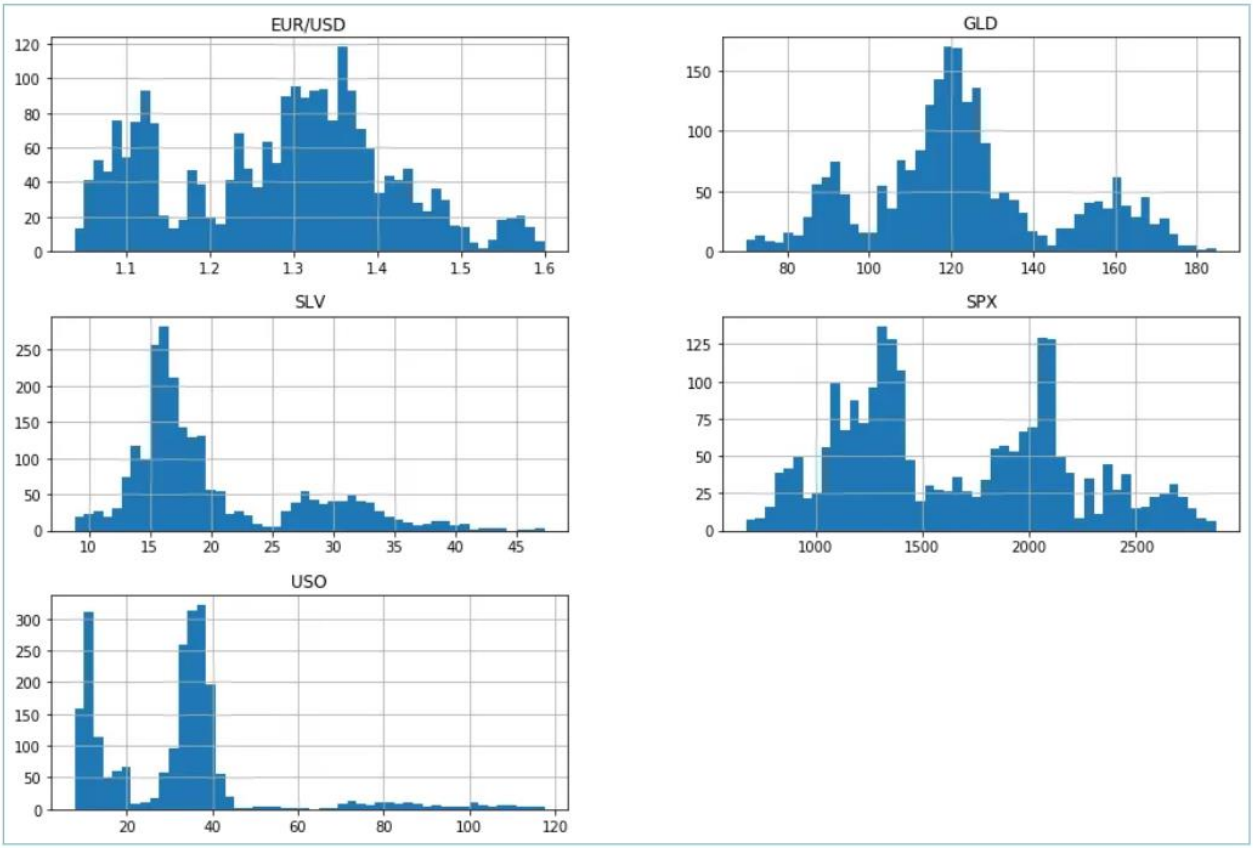
➢ About the dataset used :

Gold price dataset (gld_price_data.csv)is used from kaggle which includes columns like SPX,USO,SLV which are the parameters and factors in the world market on which the gold prices depend.

	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

Printing histograms to see layout of values for each feature

```
1. import matplotlib.pyplot as plt
2. df.hist(bins=50, figsize=(15, 10))
3. plt.show()
```



➢ Correlation:

1. Positive Correlation
2. Negative Correlation

Positive Correlation: Two features (variables) can be positively correlated with each other. It means that when the value of one variable increase then the value of the other variable(s) also increases.

Negative Correlation: Two features (variables) can be negatively correlated with each other. It means that when the value of one variable increase then the value of the other variable(s) decreases.

Implementation:

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdec99d79d0>
```

- **Splitting X and Y into training and testing variables :**

  We will be splitting the data into four variables, viz., X_train, Y_train, X_test, Y_test.

  X_train: contains a random set of values from variable ‘ X ‘

  Y_train: contains the output (the price of Gold) of the corresponding value of X_train.

  X_test: contains a random set of values from variable ‘ X ‘, excluding the ones from X_train(as they are already taken).

  Y_train: contains the output (the price of Gold) of the corresponding value of X_test. test_size: represents the ratio of how the data is distributed among X_trai and X_test (Here 0.2 means that the data will be segregated in the X_train and X_test variables in an 80:20 ratio).

Gold_Price_Prediction MODEL.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on May 15

+ Code + Text

```
# This is formatted as code
```

```
from collections.abc import Iterable
from collections import defaultdict
import warnings
from itertools import chain, combinations
from math import ceil, floor
import numbers
from abc import ABCMeta, abstractmethod
from inspect import signature
import numpy as np
from scipy.special import comb
from object_detection.utils import indexable, check_random_state, _safe_indexing
def train_test_split
( *arrays,

test_size=None,
train_size=None,
random_state=None,
shuffle=True,
stratify=None,

):
n_arrays = len(arrays) if n_arrays == 0: raise ValueError("At least one array required as input")
arrays = indexable(*arrays)
n_samples = _num_samples(arrays[0]) n_train, n_test = _validate_shuffle_split( n_samples, test_size, train_size, default_test_size=0.25 )
if shuffle is False: if stratify is not None: raise ValueError("Stratified train/test split is not implemented for shuffle=False")

    train = np.arange(n_train)
    test = np.arange(n_train, n_train + n_test)

else: if stratify is not None: CVClass = StratifiedShuffleSplit else: CVClass = ShuffleSplit

    cv = CVClass(test_size=n_test, train_size=n_train, random_state=random_state)

    train, test = next(cv.split(X=arrays[0], y=stratify))

return list( chain.from_iterable( (_safe_indexing(a, train), _safe_indexing(a, test)) for a in arrays ) )
```

➢ Building a Decision Tree and Use of predict function:

Model Training: Random Forest Regressor

```
[ ]  regressor = RandomForestRegressor(n_estimators=100)#REGRESSOR IS THE NAME OF THE MODEL
```

The parameter *n_estimators* creates *n* number of trees in your random forest, where *n* is the number we pass in. We passed in 100. The *.fit()* function allows us to train the model, adjusting weights according to the data values in order to achieve better accuracy. After training, our model is ready to make predictions, which is called by the *.predict()* method.

+ Code + Text

Build a decision tree

def build_tree(train, max_depth, min_size, n_features):

```
root = get_split(train, n_features)
split(root, max_depth, min_size, n_features, 1)
return root
```

Make a prediction with a decision tree

def predict(node, row):

```
if row[node['index']] < node['value']:
    if isinstance(node['left'], dict):
        return predict(node['left'], row)
    else:
        return node['left']
else:
    if isinstance(node['right'], dict):
        return predict(node['right'], row)
    else:
        return node['right']
```

Evaluating Prediction Accuracy :

Now we must assess the performance of our model.

R^2 score tells us how well our model is fitted to the data by comparing it to the average line of the dependent variable. If the score is closer to 1, then it indicates that our model performs well versus if the score is farther from 1, then it indicates that our model does not perform so well.
We achieved an accuracy score of approximately 99%.

Calculate accuracy percentage

def accuracy_metric(actual, predicted):

```
correct = 0

for i in range(len(actual)):
    if actual[i] == predicted[i]:
        correct += 1
return correct / float(len(actual)) * 100.0
```

```
[ ]  # R squared error
     error_score = metrics.r2_score(Y_test, test_data_prediction)
     print("R SQUARED ERROR IS : ", error_score)
     #here it comes out to be 98.9 per cent accuracy
```

```
     R SQUARED ERROR IS :  0.9898940153752216
```

Conclusion :

As in this project, we first train a machine learning model (here we use the random forest regression supervised learning algorithm), then use the trained model for prediction of gold prices on test dataset and we can see the accuracy through the plotted graph wherein the observation shows actual prices and predicted prices almost overlap each other.