MATLAB CODE

Predicting Gold Prices : A Comparison of Linear Regression and Random Forest

Step 1: Load and preprocess the data

```
% Loading the dataset containing gold price information

data = readtable('gold_price.csv');
```

```
% Display the first 5 rows

disp('First 5 Rows of the Dataset:');
```

First 5 Rows of the Dataset:

```
disp(data(1:5, :));
```

Date	Open	High	Low	Close	AdjClose	Volume	SP_open	SP_high	SP_low	SP_close	SP_Ajclose	SP_volume	DJ_open	DJ_high	DJ_low	DJ_close	DJ_Ajclose	DJ_volume	EG_open	EG_high	EG_low	EG_close	EG_Ajclose	EG_volume	EU_Price	EU_open	EU_high	EU_low	EU_Trend	OF_Price	OF_Open	OF_High	OF_Low	OF_Volume	OF_Trend	OS_Price	OS_Open	OS_High	OS_Low	OS_Trend	SF_Price	SF_Open	SF_High	SF_Low	SF_Volume	SF_Trend	USB_Price	USB_Open	USB_High	USB_Low	USB_Trend	PLT_Price	PLT_Open	PLT_High	PLT_Low	PLT_Trend	PLD_Price	PLD_Open	PLD_High	PLD_Low	PLD_Trend	RHO_PRICE	USDI_Price	USDI_Open	USDI_High	USDI_Low	USDI_Volume	USDI_Trend	GDX_Open	GDX_High	GDX_Low	GDX_Close	GDX_AdjClose	GDX_Volume	USO_Open	USO_High	USO_Low	USO_Close	USO_AdjClose	USO_Volume
2011-12-15	154.74	154.95	151.71	152.33	152.33	2.1522e+07	123.03	123.2	121.99	122.18	105.44	1.9911e+08	11825	11968	11825	11869	11869	1.3693e+08	74.55	76.15	72.15	72.9	70.432	7.879e+05	1.3018	1.2982	1.3051	1.2957	1	105.09	104.88	106.5	104.88	14330	1	93.42	94.91	96	93.33	0	53604	54248	54248	52316	1.1944e+05	1	1.911	1.911	1.911	1.911	1	1414.7	1420.3	1423.3	1376.8	0	618.85	614.7	615	614.6	1	1425	80.341	80.565	80.63	80.13	22850	0	53.01	53.14	51.57	51.68	48.974	2.0606e+07	36.9	36.94	36.05	36.13	36.13	1.2617e+07

2011-12-16 154.31 155.37 153.9 155.23 155.23 1.8124e+07 122.23 122.95 121.3
121.59 105.6 2.2048e+08 11870 11968 11819 11866 11866 3.8952e+08 73.6
75.1 73.35 74.9 72.364 8.966e+05 1.3035 1.302 1.3087 1.2997 1 103.35
103.51 104.56 102.46 1.4008e+05 0 93.79 93.43 94.8 92.53 1 53458
53650 54030 52890 65390 0 1.851 1.851 1.851 1.851 0 1420.2
1414.8 1431.8 1400.7 1 623.65 622.6 623.45 622.3 1 1400 80.249
80.175 80.395 79.935 13150 0 52.5 53.18 52.04 52.68 49.922
1.6285e+07 36.18 36.5 35.73 36.27 36.27 1.2579e+07

2011-12-19 155.48 155.86 154.36 154.87 154.87 1.2547e+07 122.06 122.32 120.03
120.29 104.47 1.839e+08 11867 11926 11735 11766 11766 1.3517e+08 69.1
69.8 64.2 64.7 62.509 2.0967e+06 1.2995 1.3043 1.3044 1.2981 0 103.64
103.63 104.57 102.37 1.4788e+05 1 94.09 93.77 94.43 92.55 1 52961
53400 53400 52544 67280 0 1.81 1.81 1.81 1.81 0 1411.1
1422.7 1427.6 1404.6 0 608.8 626 630 608.6 0 1400 80.207
80.3 80.47 80.125 970 0 52.49 52.55 51.03 51.17 48.491
1.512e+07 36.39 36.45 35.93 36.2 36.2 7.4182e+06

2011-12-20 156.82 157.43 156.58 156.98 156.98 9.1363e+06 122.18 124.14 120.37
123.93 107.63 2.2542e+08 11769 12117 11769 12104 12104 1.6518e+08 66.45
68.1 66 67 64.732 8.753e+05 1.3079 1.3003 1.3133 1.2994 1 106.73
104.3 107.27 103.91 1.7024e+05 1 95.55 96.39 99.7 96.39 1 53487
52795 53575 52595 55130 1 1.927 1.927 1.927 1.927 1 1434.8
1409 1436.5 1408.2 1 626.65 622.45 622.45 622.45 1 1400 80.273
80.89 80.94 80.035 22950 1 52.38 53.25 52.37 52.99 50.215
1.1645e+07 37.3 37.61 37.22 37.56 37.56 1.0042e+07

2011-12-21 156.98 157.53 156.13 157.16 157.16 1.1996e+07 123.93 124.36 122.75
124.17 107.84 1.9423e+08 12104 12120 11999 12108 12108 1.6325e+08 67.1
69.4 66.9 68.5 66.181 8.376e+05 1.3045 1.3079 1.3197 1.3024 0 107.71
107.15 108.17 106.16 1.4509e+05 1 99.01 97.54 99.26 96.81 1 53148
53519 54184 52937 75950 0 1.97 1.97 1.97 1.97 1 1429
1434.4 1453.8 1417.7 0 635.9 625.7 641.5 623.8 1 1400 80.35
80.105 80.445 79.55 24140 1 53.15 53.43 52.42 52.96 50.187
8.7243e+06 37.67 38.24 37.52 38.11 38.11 1.0728e+07

```
%The dataset serves as the foundation of the analysis. Without
loading the data correctly, further steps such as feature
selection and model training cannot proceed.
```

Step 2: Data Preprocessing

```
% Selected relevant features for predicting Gold Close Price

features = {'SP_close', 'DJ_close', 'USDI_Price', 'OF_Price',
'EU_Price', 'PLT_Price'};

target = 'Close'; % Target variable: Gold Close Price

%Identifing predictor variables (e.g., stock prices, currency
rates) that could influence the target variable (Gold Close
Price).
```

```
%Feature selection is critical for training accurate models. Irrelevant or redundant features can degrade model performance and its complicate analysis.
```

```
% Extract predictors (X) and target (Y)

X = data{:,features};

Y = data{:, target};
```

```
% Handle Missing Values: Remove rows with missing data

validRows = ~any(ismissing(X), 2) & ~ismissing(Y);

X = X(validRows, :);

Y = Y(validRows);

%Removes rows with missing data to ensure models are trained on complete and reliable information.

%Missing values can introduce bias or mistakes in version predictions. Cleaning the facts guarantees that the effects mirror the proper relationships among predictors and the goal variable.
```

```
% Check for Missing Values

missingValues = sum(ismissing(data));

disp('Number of Missing Values in Each Column:');
```

Number of Missing Values in Each Column:

```
disp(missingValues);
```

```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
```

Step 3: Exploratory Data Analysis

```
% Correlation Analysis to identify relationships among features

selectedFeatures = {'Close', 'SP_close', 'DJ_close', 'USDI_Price', 'OF_Price', 'EU_Price', 'PLT_Price'};

selectedData = data(:, selectedFeatures);
```

```
% Calculate Correlation Matrix

numericData = table2array(selectedData);

correlationMatrix = corr(numericData, 'Rows', 'pairwise');

%Computes and visualizes the correlations among capabilities and
the goal variable the use of a heatmap.

%Understanding the relationships among capabilities enables
perceive which predictors have the most powerful affect at the
goal. It additionally famous multicollinearity, that can have an
effect on version performance.

% Create Heatmap

figure;

h = heatmap(selectedFeatures, selectedFeatures,
correlationMatrix, ...

   'Colormap', cool, 'ColorbarVisible', 'on', 'CellLabelFormat',
'%.4f'); % Display values with 4 decimals
```

```
% Customize Heatmap Appearance

title('Correlation Heatmap');

blueColormap = [linspace(0.9, 0, 256)', linspace(0.9, 0, 256)',
ones(256, 1)];
```

```
% Visualize Target Variable (Gold Close Price)

figure;

histogram(data.Close, 30, 'FaceColor', 'g');

xlabel('Gold Close Price');

ylabel('Frequency');

title('Distribution of Gold Close Prices');

% Plot Time Series of Gold Prices

figure;

plot(data.Date, data.Close, 'LineWidth', 1.5);

datetick('x', 'yyyy');
```

```
xlabel('Date'); ylabel('Gold Close Price');
title('Time Series of Gold Close Prices');
grid on;
```

```
% Feature Relationships
% Scatter Plot: Gold Close Price vs Selected Features
figure;
for i = 1:length(selectedFeatures)
   subplot(2, 2, i);
   scatter(data{:, selectedFeatures{i}}, data.Close, 'filled');
   xlabel(selectedFeatures{i});
   ylabel('Gold Close Price');
   title(['Gold Close Price vs ', selectedFeatures{i}]);
end
```

```
Error using subplot (line 293)
Index exceeds number of subplots.
```

```
% Boxplot to Analyze Outliers
figure;
boxplot(data.Close);
title('Boxplot of Gold Close Prices to Identify Outliers');
```

Step 4: Train-Test Split

```
%Divides the dataset into training (80%) and testing (20%) subsets to evaluate model performance on unseen data.
%A proper split ensures that the models are evaluated on data they haven't seen during training, providing a realistic measure of their generalization ability.
```

```
% Created an 80-20 train-test split
rng(42); % Seed for reproducibility
```

```
cv = cvpartition(size(X, 1), 'HoldOut', 0.2);

XTrain = X(training(cv), :);

yTrain = Y(training(cv));

XTest = X(test(cv), :);

yTest = Y(test(cv));
```

Step 5: Normalize Features

```
%Standardizes the predictors to have zero mean and unit variance using the training set statistics.

%Normalization prevents features with larger scales from dominating model training. This is especially important for linear models but less critical for Random Forest.
```

```
% Normalize using training data statistics

[XTrain, mu, sigma] = normalize(XTrain);

XTest = normalize(XTest, 'center', mu, 'scale', sigma);
```

```
function metrics = calculateRegressionMetrics(yTrue, yPred)

   % Function to calculate regression performance metrics

   % yTrue: Ground truth values

   % yPred: Predicted values
```

```
   % Mean Squared Error

   mse = mean((yTrue - yPred).^2);


   % Root Mean Squared Error

   rmse = sqrt(mse);


   % Mean Absolute Error

   mae = mean(abs(yTrue - yPred));
```

```
    % R-squared (Coefficient of Determination)
    ssTotal = sum((yTrue - mean(yTrue)).^2);
    ssRes = sum((yTrue - yPred).^2);
    r2 = 1 - (ssRes / ssTotal);

    % Store metrics in a structure
    metrics = struct('MSE', mse, 'RMSE', rmse, 'MAE', mae, 'R2',
r2);
end
```

Step 6: Train and Evaluate Models Using 5-Fold Cross-Validation

```
% Initialize structure for metrics
metrics = struct();
cvPartition = cvpartition(size(XTrain, 1), 'KFold', 5);
%Splits the training data into 5 subsets for cross-validation,
ensuring that each model is evaluated on multiple folds.
%Cross-validation provides a robust estimate of model performance
by reducing the risk of overfitting to a single split.
```

```
% Initialize predictions
linearTrainPred = zeros(size(yTrain));
linearTestPred = zeros(size(yTest));
rfTrainPred = zeros(size(yTrain));
rfTestPred = zeros(size(yTest));
```

```
% Loop over cross-validation folds
for fold = 1:cvPartition.NumTestSets
```

```
    % Training and validation split for current fold
    trainIdx = training(cvPartition, fold);
    valIdx = test(cvPartition, fold);

    % Training data for the fold
    XFoldTrain = XTrain(trainIdx, :);
    yFoldTrain = yTrain(trainIdx);
    XFoldVal = XTrain(valIdx, :);
    yFoldVal = yTrain(valIdx);

    % Train Linear Regression
    mdlLinear = fitlm(XFoldTrain, yFoldTrain);
    %Trains a Linear Regression version, which assumes a linear
dating among predictors and the goal variable.
    %Linear Regression is a baseline version that is
straightforward and interpretable, making it beneficial for
knowledge how predictors have an impact on the goal.
    yFoldLinearPred = predict(mdlLinear, XFoldVal);

    % Train Random Forest
    mdlRF = fitrensemble(XFoldTrain, yFoldTrain, 'Method', 'Bag',
...
        'NumLearningCycles', 50, 'Learners', templateTree());
    %Trains a Random Forest model, that is an ensemble of
selection timber that captures non-linear relationships and
interactions among features.
    %Random Forest frequently outperforms easier fashions like
Linear Regression, specifically whilst the relationships among
variables are complicated or non-linear.
    yFoldRFPred = predict(mdlRF, XFoldVal);

    % Evaluate on validation set
```

```
    metrics.Linear.Fold{fold} =
calculateRegressionMetrics(yFoldVal, yFoldLinearPred);

    metrics.RF.Fold{fold} = calculateRegressionMetrics(yFoldVal,
yFoldRFPred);

end
```

Step 7: Train Final Models on Full Training Set

```
% Linear Regression

mdlLinearFinal = fitlm(XTrain, yTrain);
```

```
% Random Forest

mdlRFFinal = fitrensemble(XTrain, yTrain, 'Method', 'Bag',
'NumLearningCycles', 50, 'Learners', templateTree());
```

```
% Predictions on Training and Test Sets

linearTrainPred = predict(mdlLinearFinal, XTrain);

linearTestPred = predict(mdlLinearFinal, XTest);

rfTrainPred = predict(mdlRFFinal, XTrain);

rfTestPred = predict(mdlRFFinal, XTest);
```

```
% Evaluate Final Models

metrics.Linear.Final.Train = calculateRegressionMetrics(yTrain,
linearTrainPred);

metrics.Linear.Final.Test = calculateRegressionMetrics(yTest,
linearTestPred);

metrics.RF.Final.Train = calculateRegressionMetrics(yTrain,
rfTrainPred);

metrics.RF.Final.Test = calculateRegressionMetrics(yTest,
rfTestPred);
```

```
% Step 7.1: Evaluate Models on Entire Training Data

% Use trained models to predict on the full training data

disp('=== Model Performance on Entire Training Set ===');
```

=== Model Performance on Entire Training Set ===

```
% Predictions
linearFullTrainPred = predict(mdlLinearFinal, XTrain);  % Linear Regression
rfFullTrainPred = predict(mdlRFFinal, XTrain);          % Random Forest
```

```
% Evaluate Models
metrics.Linear.FullTrain = calculateRegressionMetrics(yTrain, linearFullTrainPred);
metrics.RF.FullTrain = calculateRegressionMetrics(yTrain, rfFullTrainPred);
```

```
% Display Results
disp('Linear Regression Performance on Full Training Data:');
```

Linear Regression Performance on Full Training Data:

```
disp(metrics.Linear.FullTrain);
```

MSE: 32.6698

RMSE: 5.7158

MAE: 4.7681

R2: 0.8940

```
disp('Random Forest Performance on Full Training Data:');
```

Random Forest Performance on Full Training Data:

```
disp(metrics.RF.FullTrain);
```

MSE: 1.8858

RMSE: 1.3733

MAE: 0.8941

R2: 0.9939

```
% Visualization: Predicted vs Actual on Training Data
figure('Name', 'Predicted vs Actual (Training Data)');
subplot(1, 2, 1);
```

```
scatter(yTrain, linearFullTrainPred, 'b.');
title('Linear Regression: Predicted vs Actual (Train)');
xlabel('True Values');
ylabel('Predicted Values');
grid on;
```

```
subplot(1, 2, 2);
scatter(yTrain, rfFullTrainPred, 'r.');
title('Random Forest: Predicted vs Actual (Train)');
xlabel('True Values');
ylabel('Predicted Values');
grid on;
```

```
% Residual Analysis for Entire Training Data
figure('Name', 'Residuals on Training Data');
subplot(1, 2, 1);
scatter(yTrain, linearFullTrainPred - yTrain, 'b.');
title('Linear Regression Residuals (Full Train)');
xlabel('True Values');
ylabel('Residuals');
grid on;
```

```
subplot(1, 2, 2);
scatter(yTrain, rfFullTrainPred - yTrain, 'r.');
title('Random Forest Residuals (Full Train)');
xlabel('True Values');
ylabel('Residuals');
grid on;
```

Step 8: Compare Models

```
fprintf('\n=== Final Model Performance Comparison ===\n\n');
```

=== Final Model Performance Comparison ===

```
comparisonTable = table({'Linear Regression'; 'Random Forest'}, ...
    [metrics.Linear.Final.Train.RMSE, metrics.Linear.Final.Test.RMSE]', ...
    [metrics.RF.Final.Train.RMSE, metrics.RF.Final.Test.RMSE]', ...
    'VariableNames', {'Model', 'Train_RMSE', 'Test_RMSE'});
disp(comparisonTable);
```

Model	Train_RMSE	Test_RMSE
{'Linear Regression'}	5.7158	1.3733
{'Random Forest' }	5.9076	2.3927

```
% Visualize RMSE Comparison
figure('Name', 'Model RMSE Comparison');
bar([metrics.Linear.Final.Train.RMSE, metrics.Linear.Final.Test.RMSE; ...
    metrics.RF.Final.Train.RMSE, metrics.RF.Final.Test.RMSE]');
set(gca, 'XTickLabel', {'Train', 'Test'});
legend('Linear Regression', 'Random Forest');
ylabel('RMSE');
title('RMSE Comparison for Models');
grid on;
```

Step 9: Residual Analysis

```
% Residual Plots
%Residual analysis helps identify systematic errors or outliers that models might struggle with.
```

```
figure('Name', 'Residual Plots');
```

```
subplot(1, 2, 1);

scatter(yTrain, linearTrainPred - yTrain, 'b.');

title('Linear Regression Residuals (Train)');

xlabel('True Values');

ylabel('Residuals');

grid on;
```

```
subplot(1, 2, 2);

scatter(yTest, rfTestPred - yTest, 'r.');

title('Random Forest Residuals (Train)');

xlabel('True Values');

ylabel('Residuals');

grid on;
```

Step 10: ROC Curve Comparison

```
% Threshold for Binary Classification

threshold = mean(yTest);

trueLabels = (yTest >= threshold);

linearPredLabels = (linearTestPred >= threshold);

rfPredLabels = (rfTestPred >= threshold);
```

```
% Calculate ROC Curves

[XLinear, YLinear, TLinear, AUCLinear] = perfcurve(trueLabels,
linearTestPred, 1);

[XRF, YRF, TRF, AUCRF] = perfcurve(trueLabels, rfTestPred, 1);
```

```
% Plot ROC Curves

figure('Name', 'ROC Curves');

plot(XLinear, YLinear, 'b', 'LineWidth', 1.5);

hold on;
```

```
plot(XRF, YRF, 'r', 'LineWidth', 1.5);
hold off;
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve for Models');
legend(sprintf('Linear Regression (AUC: %.2f)', AUCLinear), ...
      sprintf('Random Forest (AUC: %.2f)', AUCRF), 'Location', 
'Best');
grid on;
```

Step 11: Confusion Matrices

```
% Confusion Matrix: Linear Regression
figure('Name', 'Confusion Matrix - Linear Regression');
confusionchart(categorical(yTest >= threshold), 
categorical(linearPredLabels), ...
   'Title', 'Confusion Matrix - Linear Regression (Test Data)');
```

```

```

```
% Confusion Matrix: Random Forest
figure('Name', 'Confusion Matrix - Random Forest');
confusionchart(categorical(yTest >= threshold), 
categorical(rfPredLabels), ...
   'Title', 'Confusion Matrix - Random Forest (Test Data)');
```

Step 12: Training Time Comparison

```
% Measure training times
linearTrainingTime = timeit(@() fitlm(XTrain, yTrain));
rfTrainingTime = timeit(@() fitrensemble(XTrain, yTrain, 
'Method', 'Bag', 'NumLearningCycles', 50, 'Learners', 
templateTree()));
```

```

```

```
% Display training times
```

```
disp(['Training Time (Linear Regression): ',
num2str(linearTrainingTime), ' seconds']);
```

Training Time (Linear Regression): 0.01044 seconds

```
disp(['Training Time (Random Forest): ', num2str(rfTrainingTime),
' seconds']);
```

Training Time (Random Forest): 0.43277 seconds

```
% Visualize Training Time Comparison

figure('Name', 'Training Time Comparison');

bar([linearTrainingTime, rfTrainingTime]);

set(gca, 'XTickLabel', {'Linear Regression', 'Random Forest'});

ylabel('Training Time (seconds)');

title('Training Time Comparison');

grid on;
```

Step 13: Predicted vs True Values

```
figure('Name', 'Predicted vs True Values');

subplot(1, 2, 1);

scatter(yTest, linearTestPred, 'b.');

title('Linear Regression: Predicted vs True');

xlabel('True Values');

ylabel('Predicted Values');

grid on;
```

```
subplot(1, 2, 2);

scatter(yTest, rfTestPred, 'r.');

title('Random Forest: Predicted vs True');

xlabel('True Values');

ylabel('Predicted Values');

grid on;
```

Helper Function: Calculate Regression Metrics

```
% Initialize metrics storage
resultsTable = table({'Linear Regression'; 'Random Forest'}, ...
   'VariableNames', {'Model'});
% Initialize metrics storage
resultsTable = table({'Linear Regression'; 'Random Forest'}, ...
   'VariableNames', {'Model'});
```

```

```

```
% Calculate and Store Precision
resultsTable.Precision = [calculatePrecision(yTest,
linearTestPred, threshold), ...
                         calculatePrecision(yTest, rfTestPred,
threshold)]';
```

```

```

```
% Calculate and Store Recall
resultsTable.Recall = [calculateRecall(yTest, linearTestPred,
threshold), ...
                      calculateRecall(yTest, rfTestPred,
threshold)]';
```

```

```

```
% Calculate and Store F1 Score
resultsTable.F1Score = [calculateF1Score(yTest, linearTestPred,
threshold), ...
                       calculateF1Score(yTest, rfTestPred,
threshold)]';
```

```

```

```
% Calculate and Store AUC
resultsTable.AvgAUC = [AUCLinear, AUCRF]';
```

```

```

```
% Display the Results Table
disp('--- Final Metrics Table ---');
```

--- Final Metrics Table ---

```
disp(resultsTable);
```

Model	Precision	Recall	F1Score	AvgAUC
{'Linear Regression'}	0.71186	0.84848	0.77419	0.9444
{'Random Forest' }	0.94186	0.81818	0.87568	0.97831

```
% --- Helper Functions ---
function acc = calculateAccuracy(yTrue, yPred)
   % Rounding predictions to nearest integer for accuracy
calculation
   yPred = round(yPred);
   acc = sum(yPred == yTrue) / numel(yTrue) * 100; % Accuracy in
percentage
end
```

```
function precision = calculatePrecision(yTrue, yPred, threshold)
   % Converting to binary classification
   yPred = yPred >= threshold;
   yTrue = yTrue >= threshold;
   TP = sum((yTrue == 1) & (yPred == 1)); % True Positives
   FP = sum((yTrue == 0) & (yPred == 1)); % False Positives
   precision = TP / (TP + FP);
end
```

```
function recall = calculateRecall(yTrue, yPred, threshold)
   % Converting to binary classification
   yPred = yPred >= threshold;
   yTrue = yTrue >= threshold;
   TP = sum((yTrue == 1) & (yPred == 1)); % True Positives
   FN = sum((yTrue == 1) & (yPred == 0)); % False Negatives
```

```
    recall = TP / (TP + FN);
end
```

```
function f1 = calculateF1Score(yTrue, yPred, threshold)
    precision = calculatePrecision(yTrue, yPred, threshold);
    recall = calculateRecall(yTrue, yPred, threshold);
    f1 = 2 * (precision * recall) / (precision + recall);
end
```