# Introduction

The project is to apply Ensemble Models into the prediction of coffee price. This document explains the workflow, analyzes result and provide future development.

First, before running the code, you must install all libraries via:  
pip install -r requirements.txt

# Support function

I have created some function that can help the analytical process and visualize result. I document it for understanding and reuse:

## 1. normalized(data, columns, method="minmax")

**Description:** Normalizes the specified columns of a dataset using either Min-Max scaling or Z-score standardization.

**Parameters:**

* **data** (*pd.DataFrame*) – The dataset.
* **columns** (*list of str*) – Column names that need to be normalized.
* **method** (*str, default="minmax"*) – Normalization technique to apply:
  + "minmax" → Scales values between 0 and 1.
  + "zscore" → Standardizes values to have mean = 0 and standard deviation = 1.

**Returns:** (*pd.DataFrame*) – A new DataFrame with normalized values for the selected columns.

## 2. showdata(data, start=0, end=None, normalize=True, method="minmax")

**Description:** Displays a line graph for values in wanted columms

**Parameters:**

* **data** (*pd.DataFrame*) – The dataset to display.
* **start** (*int, default=0*) – The starting index of rows to display.
* **end** (*int, optional*) – The ending index (default is None, which shows all rows after start).
* **normalize** (*bool, default=True*) – Whether to apply normalization before displaying.
* **method** (*str, default="minmax"*) – The normalization method ("minmax" or "zscore").

**Returns**: (None) -print a graph for wanted values

## 3. plot\_predictions(y\_test, y\_pred)

**Description:**  
Plots actual vs. predicted values for evaluating model performance.

**Parameters:**

* **y\_test** (*array-like*) – The true target values.
* **y\_pred** (*array-like*) – The predicted values from the model.

**Returns:**

* (*None*) – Displays a line graph comparing y\_test and y\_pred.

## 4. analyze(model)

**Description:**  
show the weights of features in a model.

**Parameters:**

* **model** (*sklearn estimator*) – A trained machine learning model

**Returns:** (*None*) – A bar chart of weight values

A graph with red and blue lines

AI-generated content may be incorrect.

# Preparing training data

## Feature extraction

This step is to decide the features used for the prediction. Features will be extracted without normalizing (you can always change that by using functions). The list of feature:

* + **Year**: track time to catch any time-related trends
  + **Time**: Catch any Cycle Trend. Using Forier Transformation with R = mean(Price) /1.5
  + **Exponential Moving Average (EMA)** is a weighted moving average that gives to capture moving trends. This helps smooth out the input values by exponentally emphasizing the recent trends. We use EMA in a week and a month to support prediction
  + **Mean Price of week**

From the original data, we extract those features to different columms. Then you can preview data of features via function calling:

showdata(df,4700,4950, True)

A graph showing a number of data

AI-generated content may be incorrect.

II. Training Data

Split into 2 subset: training set and validation set

Ratio: 0.2 🡺 the first 80% of the data is used for training, and the rest for testing

Since this is time series, we do not shuffle

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, shuffle=False)

# Training base model

## Random Forest Regression

* + A tree based machine learning model. I add no parameters (this model will emphasize generation

rf = RandomForestRegressor()

## Gradient Boosting Regression (GBR)

A machine learning technique that builds an ensemble of decision trees sequentially, where each new tree corrects the errors of the previous trees using gradient descent.

To optimize the Model, I use the following hyper-parameters, optimizing their values via GridSearchCV

* learning\_rate=0.05: Controls weight of each tree to the final prediction
  + lower value more trees
* max\_depth=5: Maximum depth of each decision tree.
  + The deeper the tree, the more data we memories
* max\_features='sqrt':
  + Number of features considered for each split in a tree.
  + 'sqrt' means using square root of total features
* min\_samples\_leaf=10: Minimum number of samples required in a leaf node.
* min\_samples\_split=20: Minimum samples required to split an internal node.
* n\_estimators=100: Number of trees (weak learners) in the ensemble.
* subsample=0.9: Fraction of training data randomly selected for each tree.

## III. Extreme Gradient Boosting Regression (XGBR)

An optimized gradient boosting library designed for **speed and performance**. It improves upon traditional Gradient Boosting Machines (GBMs) by adding **regularization, parallel processing, and efficient memory usage**.

From that, I use the same params as GBR, finetuning via optuna, then add more params that exclusive to XGBR:

 **gamma=5.222152554368784**: Minimum loss reduction required to split a node.

* This value will help us solve overfitting problems explained afterwards.

 **reg\_alpha=0.24931830144036302**: **L1 regularization** (Lasso penalty) → Shrinks coefficients to remove less useful features.

 **reg\_lambda=0.03556395419506893**: **L2 regularization** (Ridge penalty) → Prevents extreme weight values.

 **min\_child\_weight=3**: Minimum sum of instance weights needed in a leaf node.

* Higher values prevent overly complex trees.

 **booster='dart'**: Type of boosting algorithm used.

* 'dart' adds **random dropout** to trees, improving generalization.

# Evaluating base model

After training and testing, here is the result of model, based on simple metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | MSE | RMSE | R² | Accuracy (%)  (1 – MAPE) |
| RFR | 8.319994 | 2.884440 | 0.910003 | 98.116410 |
| GBR | 20.376128 | 4.513992 | 0.779592 | 96.785816 |
| XGBR | 9.963435 | 3.156491 | 0.892226 | 97.913408 |

Time cost: XGBR > RFR ~ GBR. XGBR took 10s while others compile nearly immediately

Based on result, RFR > XGBR > GBR. By plot out predictions with seaborn, following conclusions are made:

* + The data works well with a more generalized model like RFR 🡺 Linear model may have good performance
  + GBR shows overfitting tendency, even with well modified Parameters via GridCV
  + XGBR is better dealing with overfitting thanks to L1 and L2 regulization.
  + We see a slightly better performance when changing booster of XGBR from gbtree to dart, to improve genelization while still keeping the sufficient amount of tree for boosting technique

# Training Stack

From the training of previous Emsemble techniques, we have on overview of data and how models work well with them. >95% performance shows optimistic result, and with stack, we can combine all its strength.

Base model of Stack:

* + Gradient Boosting Regression: Gradient-based tree model to capture non-linear and complex pattern
  + Random Forest Regression: A tree-based ML works well with generalization. Provide stability for Stack Model while capture non-lineality
  + Linear Regression: A simple model to capture linear patterns.

Meta – Learner:

* + This is the model that receive values from base model and intepret result base on them.

meta\_learner = LassoCV(cv=5)

Lasso (**Least Absolute Shrinkage and Selection Operator**) is a **linear regression model with L1 regularization**. The objective function for Lasso is:

A math equation with black text

AI-generated content may be incorrect.

The lasso can shrink coefficient to zero, leading to feature selection

A screenshot of a computer

AI-generated content may be incorrect.

# Evaluate stack

After training and testing, here is the result of all models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | MSE | RMSE | R² | Accuracy (%)  (1 – MAPE) |
| Stack | 5.232812 | 2.287534 | 0.943397 | 98.482827 |
| RFR | 8.319994 | 2.884440 | 0.910003 | 98.116410 |
| GBR | 20.376128 | 4.513992 | 0.779592 | 96.785816 |
| XGBR | 9.963435 | 3.156491 | 0.892226 | 97.913408 |

A result model is proven to perform slightly better than all models.

Conclusion: Though there is some small mix of non linear pattern, the data is highly linear thanks to smoothened features. Features important score shows that.  
A graph with a blue line

AI-generated content may be incorrect.

Some of the other plot as well:

A graph showing a graph of a price

AI-generated content may be incorrect.

A graph with red lines

AI-generated content may be incorrect.