MPhil in Economics and Data Science

Module: D400 - Research Computing

Candidate Number (BGN): 3378E

Deadline Date: 19/12/2024

I confirm that this is entirely my own work and has not previously been submitted for assessment, and I have read and understood the University's and Faculty's definition of Plagiarism (please see links below)

Plagiarism and Academic Misconduct

Plagiarism and correct referencing in dissertations Moodle Econ_Update (003).pdf

Actual word count: 1,574

Gold ETF Price Prediction Project Report

1. Motivation

In the history, a lot of countries use gold as legal currency. Nowadays, gold is also seen as a kind of safe currency and can be used to hedge against inflation and geopolitical risks. But the price fluctuation of gold is unexpected and affected by several factors. Therefore, this project aims to figure out the most important features that affect Gold ETF adjusted closing prices using machine learning.

2. Explanatory Data Analysis

2.1 Dataset Overview

The raw dataset contains 1,718 rows and 81 columns. There are 58 float variables (such as Open, High, Low), 22 int variables (such as Volume), and 1 object variable (date).

The missingno matrix plot shows that there is no missing value within the dataset, while the outlier plot shows that there are some outliers for several features.

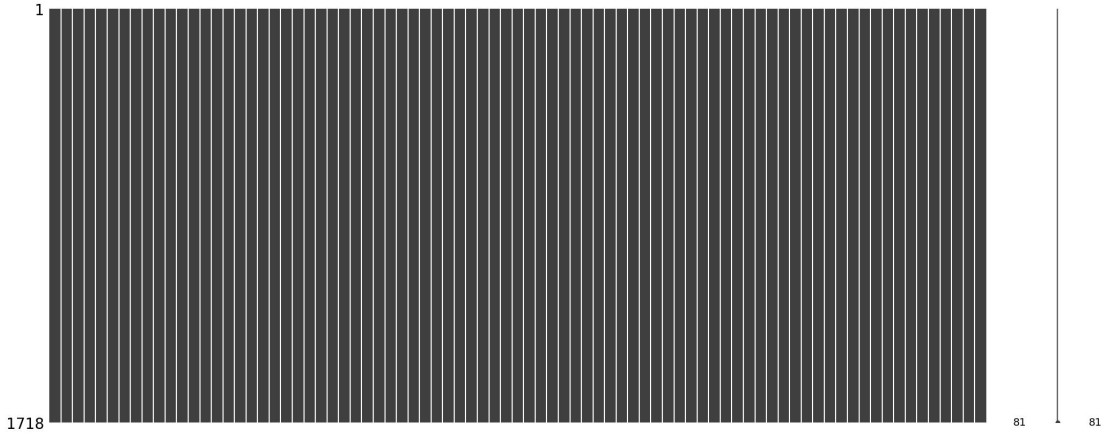

Figure 2-1 Missingno Matrix Plot

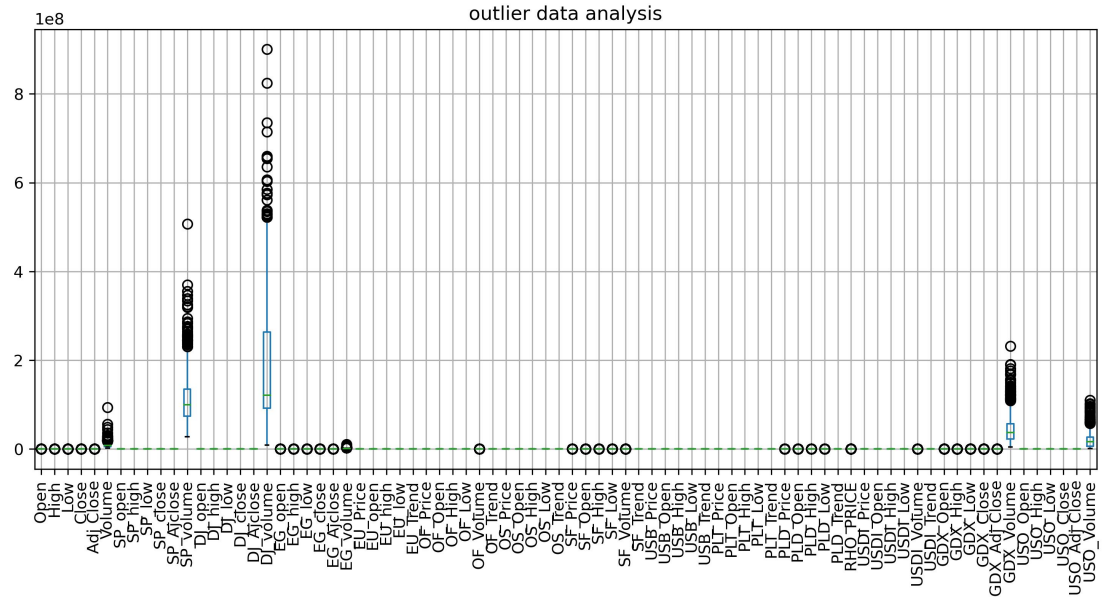

Figure 2-2 Outlier Data Analysis Plot

2.2 Target Variable: Adj Close Price

The target variable (Adj Close Price) is float type, with mean of 127.32, max of 173.61, min of 100.50, and standard deviation of 17.40.

1) Frequency Distribution

The histogram of the target variable reveals a two-peaked distribution: One peak around 120, and another peak around 160. The shape suggests the presence of two normal distributions, indicating potential segmentation or structural changes over history.

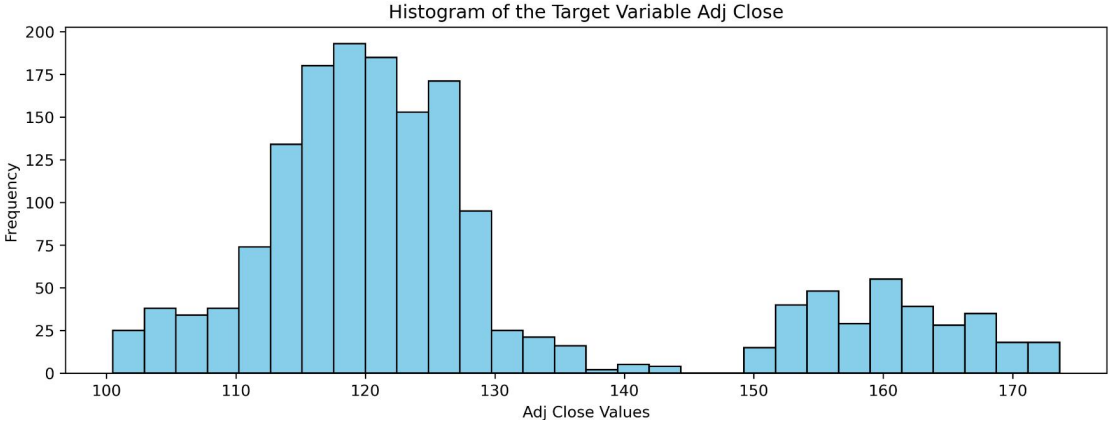

Figure 2-3 Frequency Histogram Plot

2) Time Series Trend

The time series trend plot of Adj Close Price shows a trend of price fluctuation. The price initially decreased from approximately 170 to nearly 100, and then increased back to around 120. This

time series trend suggests a non-linear pattern with significant variation over time.

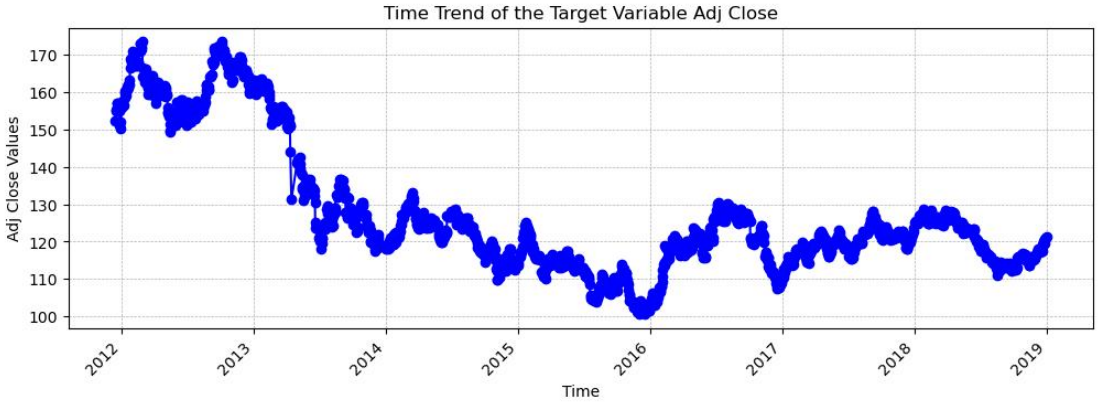

Figure 2-4 Time Series Trend Plot

2.3 Correlation

From the heatmap of correlation coefficients between different variables and the plot of correlation coefficients between other variables and Adj Close Price, it can be seen that there are several variables highly linearly correlated with the target variable (Adj Close Price), either positively or negatively. These highly correlated variables are likely to have a significant influence on predicting the target variable, and bring out a multicollinearity problem.

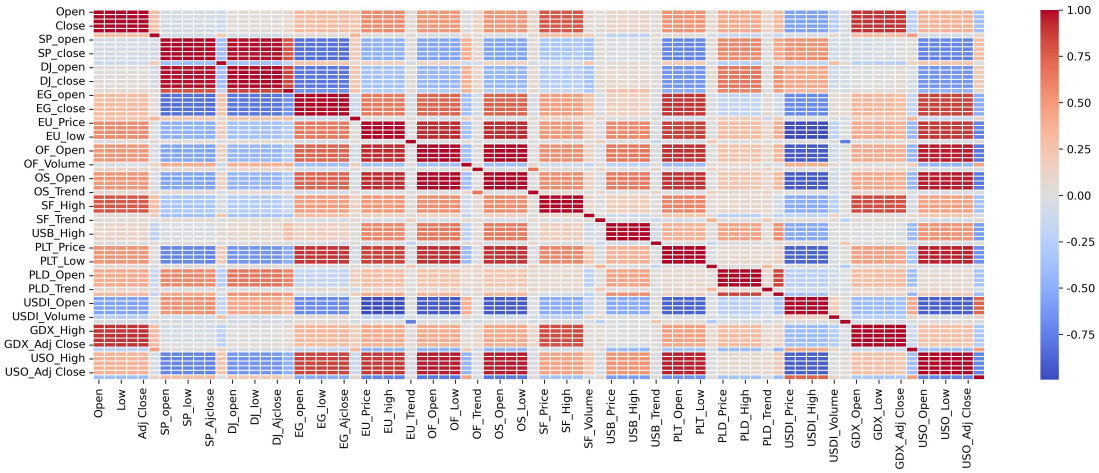

Figure 2-5 Correlation Coefficient Heatmap

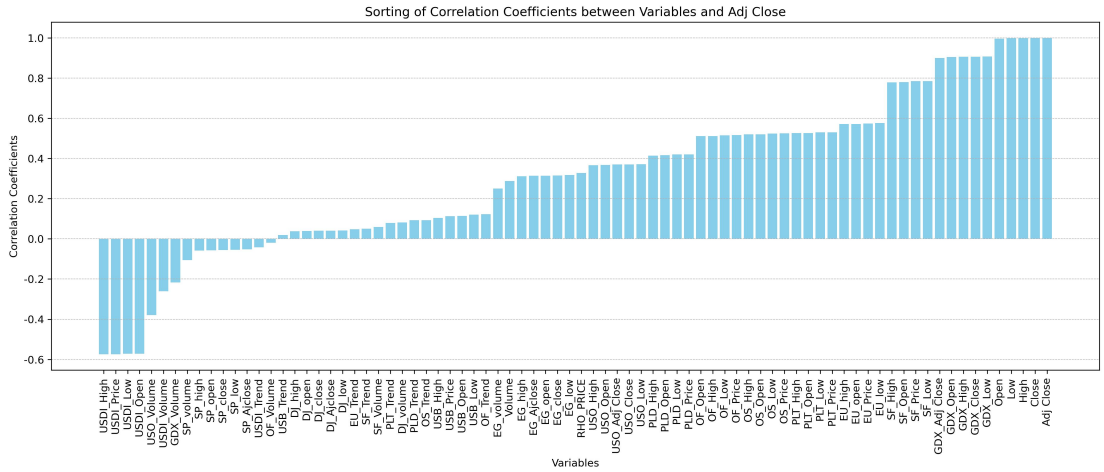

Figure 2-6 Correlation Coefficient Between Variables and Adj Close Price

2.4 Cleaning Steps

Before proceeding to model training, I performed several data cleaning steps to ensure the dataset's quality and suitability for analysis:

1) Convert date colume to datetime type
2) Check missing values
3) Remove outliers
4) Delete variables with high correlation coefficients

3. Feature Selection Approach

My feature selection consists of two main steps. After the two steps, I deleted 23 features in total.

3.1 Lasso-based Feature Selection

I defined a function named “lasso_features_selection”. Using this function, I applied LassoCV to the target variable (Adj Close Price) and all other variables to calculate the optimal alpha (regularization strength) and get the coefficients of each feature. Those features with zero coefficients were eliminated, and those with non-zero coefficients were retained.

```
# List the columns with zero importance from Lasso Regression (to be deleted)
coef=pd.DataFrame([lasso_coef]).T
coef.index=X_train.columns
coef.columns=['importance']
coef=coef.sort_values(by='importance')
drop_lasso_features=coef[coef['importance']==0].index
print("Drop columns:",drop_lasso_features)
```

✓ 0.0s Python

```
Drop columns: Index(['EU_low', 'USB_High', 'USB_Open', 'EU_high', 'EU_open', 'EU_Price',
       'USO_Adj Close', 'DJ_Ajclose', 'USB_Low'],
      dtype='object')
```

Figure 3-1 Code: Lasso-based Feature Selection

3.2 Correlation Coefficient Feature Selection

I calculated the correlation coefficients between all other features and the target variable, and then features with a correlation coefficient larger than 0.6 were deleted. By doing so, the multicollinearity problem can be well avoided and I can ensure model robustness.

```
# List the columns with too high correlation (to be deleted)
drop_corr_feature=sort_corr[sort_corr>0.6].index
print("Drop columns:",drop_corr_feature)
```

✓ 0.0s Python

```
Drop columns: Index(['SF_High', 'SF_Open', 'SF_Price', 'SF_Low', 'GDX_Adj Close', 'GDX_Open',
       'GDX_High', 'GDX_Close', 'GDX_Low', 'Open', 'Low', 'High', 'Close',
       'Adj Close'],
      dtype='object')
```

Figure 3-2 Code: Correlation Coefficient Feature Selection

4. Feature Engineering Approach

4.1 Set up GLM and LGBM modelling pipelines

When constructing GLM and LGBM models, I designed separate pipelines. These pipelines consist of two main steps: feature scaling (both used StandardScaler) and model fitting.

For GLM model pipeline, I chose the ElasticNet model because it combines L1 and L2 regularization, which can reduce the level of overfitting. And for LGBM model pipeline, I used the LGBMRegressor, which is always used to predict continuous numeric values and suits for large datasets.

4.2 Feature Transformer: StandardScaler

I used StandardScaler in both the GLM and LGBM pipelines to transform the features into a standard normal distribution with a mean of 0 and a standard deviation of 1. By doing so, the scaling effects of features could be removed and the model performance could be improved. Besides, scaling can also help to accelerate the convergence and ensure the stability during model training.

```python
# This part is setting up my model pipelines for both GLM and LGBM
# GLM model pipeline
def glm_pipeline():
    glm_model = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('glm', ElasticNet(alpha=0.5, l1_ratio=0.01))
    ])
    return glm_model

# LGBM model pipeline
def lgbm_pipeline():
    lgbm_model = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('lgbm', LGBMRegressor(n_estimators=20))
    ])
    return lgbm_model
```

Figure 4-1 Code: Model Pipelines and StandardScaler

5. Evaluation Approach

My model evaluation function focused on 4 key performance metrics for regression models, including Root Mean Squared Error (RMSE), R-squared (R^2), Mean Absolute Percentage Error (MAPE), and Mean Absolute Error (MAE). By returning and printing these metrics, the function can help users to assess how well the models are performing and compare the performance between tuned models and original ones.

In addition to calculating these metrics, my evaluation function also provided a visualization of the prediction vs actual values, making it easier to visually inspect how well the model predictions match the actual values.

```python
#  Model evaluation function
def model_metrics(true, pred, title, if_show):
    rmse = np.sqrt(mean_squared_error(true, pred))
    r2 = r2_score(true, pred)
    mape = mean_absolute_percentage_error(true, pred)
    mae = mean_absolute_error(true, pred)
    if if_show:
        plt.figure(figsize=(12, 4), dpi=300)
        plt.plot(range(len(pred)), pred, label="pred", c='green')
        plt.scatter(range(len(pred)), true, label="true", c='red')
        plt.legend()
        plt.grid()
        plt.title(title)
        plt.savefig(f"save/{title}.png")
        plt.show()
    return rmse, r2, mape, mae
```

Figure 5-1 Code: Model Evaluation Function

6. Hyperparameter tuning & Selected Parameters

I performed hyperparameter tuning for GLM and LGBM models using grid search with k-fold cross-validation.

I first defined a function named “grid_search_model” that performs k-fold cross-validation with grid search for hyperparameter tuning. This function would return best_model (the model that performs best based on the hyperparameters) and best_params (the specific hyperparameter set that leads to the best model). In my case, I used 5-fold cross-validation, which means the dataset was split into five parts, and the model was trained and evaluated five times, each time using a different fold as the test set and the remaining four folds as the training set.

Then during GLM and LGBM model tuning, I set up two parameter grids (alpha and L1_ratio for GLM, and learning_rate, n_estimators, num_leaves, and min_child_weight for LGBM) and passed the base GLM and LGBM models along with the grids to the grid_search_model function.

Finally the best parameters for each model showed up. For the GLM model, the best combination of hyperparameters was: alpha = 0.005, L1_ratio = 0.9. And for the LGBM model, the best combination of hyperparameters was: learning_rate = 0.1, n_estimators = 800, num_leaves = 12, and min_child_weight = 2.

```
# k-fold cross validation function
def grid_search_model(params, model, X, y):
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    grid_search = GridSearchCV(model, params, cv=kf, scoring='neg_mean_squared_error', n_jobs=-1)
    grid_search.fit(X, y)
    # The detailed info of best model, best params and cv results
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
    cv_results = pd.DataFrame(grid_search.cv_results_)
    return best_model, best_params, cv_results
```

Figure 6-1 Code: k-fold cross validation function

```
# Tune the model pipelines
# GLM model tuning: alpha and l1_ratio
glm_params = {
    'alpha': [0.001, 0.01, 0.1, 0.005, 1],
    'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
}
base_glm = ElasticNet()
glm_best_model, glm_best_params, glm_cv_results = grid_search_model(glm_params, base_glm, X, y)
```

Figure 6-2 Code: GLM model tuning

```
# LGBM model tuning: learning_rate, n_estimators, n_leaves, min_child_weight
lgbm_param_grid = {
    'learning_rate': [0.0001, 0.01, 0.001, 0.1],
    'n_estimators': [100, 150, 200, 500, 650, 800],
    'num_leaves': [3, 5, 12],
    'min_child_weight': [2, 5, 10]
}
base_lgbm = LGBMRegressor()
lgbm_best_model, lgbm_best_params, lgbm_cv_results = grid_search_model(lgbm_param_grid, base_lgbm, X, y)
```

Figure 6-3 Code: LGBM model tuning

7. Results Comparison between GLM and LGBM Models

After getting the tuned GLM model and tuned LGBM model, I finished the feature importance analysis.

In order to evaluate the contribution of each feature to the model and identify which features are most critical in predicting the target variable, I extracted importance scores from tuned LGBM model and the absolute values of coefficients from tuned GLM model. By doing so, I got the list of top 5 important features in predicting gold ETF prices for each model.

```
# Get the feature importances of the tuned LGBM and GLM models and create dataframes
imp_lgbm = lgbm_best_model.feature_importances_
imp_glm = np.abs(glm_best_model.coef_)
imp_df = pd.DataFrame([imp_lgbm, imp_glm]).T
imp_df.columns = ['LGBM', 'GLM']
imp_df.index = X.columns

# Get the top 5 important features and plot them
top5_lgbm = imp_df['LGBM'].sort_values(ascending=False).head(5)
top5_glm = imp_df['GLM'].sort_values(ascending=False).head(5)
```

Figure 7-1 Code: Feature Importance Analysis

The result showed that the 5 most important features differ greatly between the two models.

For the tuned LGBM model, the top 5 important features are:

1) USB_Price (US bond Price data),
2) GDX_Volume (Gold Miners ETF volume data),
3) Volume (Gold ETF volume data),
4) DJ_Volume (Dow Jones Index volume data),
5) SP_Volume (S&P 500 Index volume data)

For the tuned GLM model, the top 5 important features are:

1) USB_Price (US bond Price data),

2) USO_Open (Oil ETF USO open data),

3) OS_Trend (Crude Oil WTI USD trend data),

4) OS_Price (Crude Oil WTI USD price data)

5) EU_Trend (EUR USD Exchange rate trend data)

It can be seen that both tuned GLM and tuned LGBM model identified USB_Price (US Bond Price) as the most important feature for predicting the Gold ETF adjusted price. This indicates that the relationship between the US bond price and the gold ETF price is a key factor influencing the model's predictions. However, in addition to the US bond price, the LGBM model emphasizes market volume data as key predictors of the gold ETF price, while the GLM model places greater emphasis on oil-related data and the exchange rate trend.

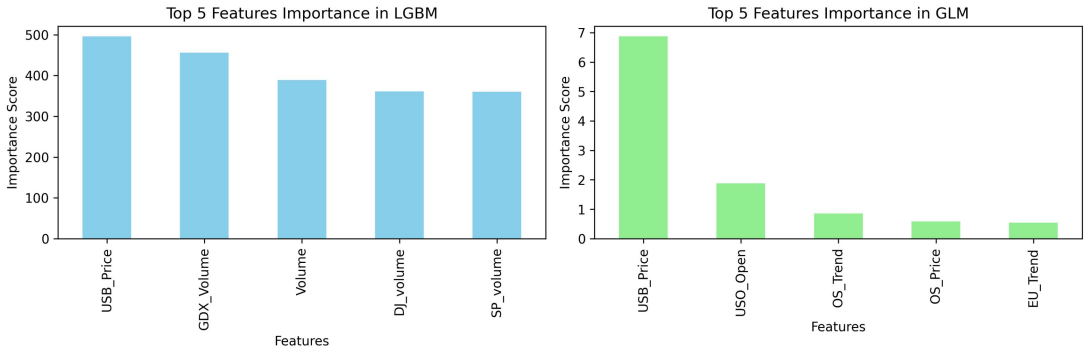

Figure 7-2 Top 5 Important Features

8. Final Performance

The Predicted vs Actual plots of two models before and after tuning below can visually prove the effectiveness of hyperparameter tuning. It is evident from the plots that after tuning, the predicted values align much more closely with the actual values, which highlights the positive impacts of the hyperparameter tuning on model performance, and it is evident that both the tuned GLM and tuned LGBM models perform exceptionally well in predicting the Gold ETF price.

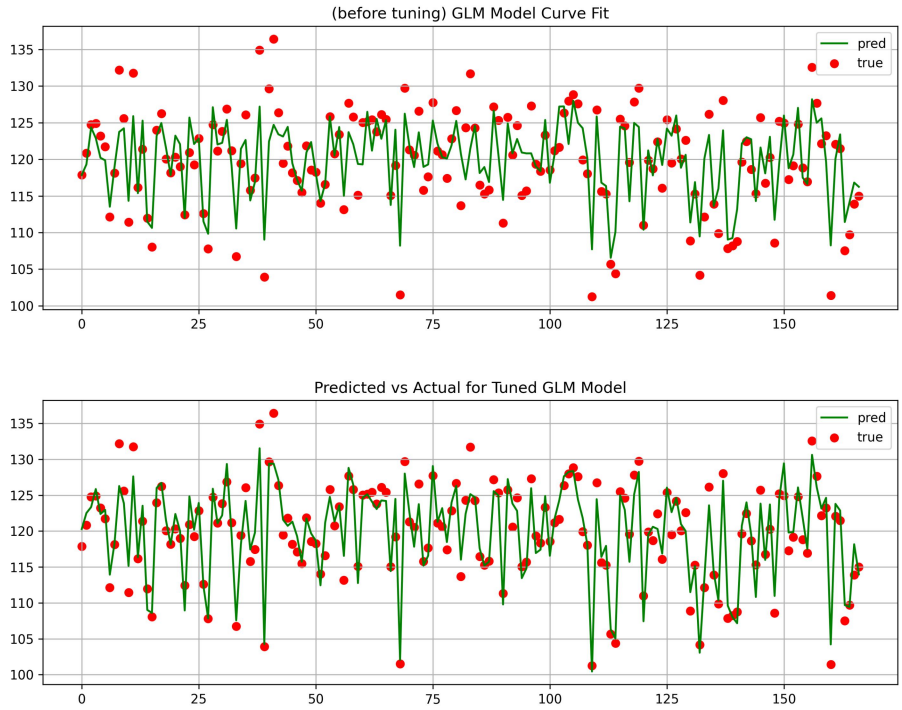

Figure 8-1 GLM: Predicted vs Actual Plots

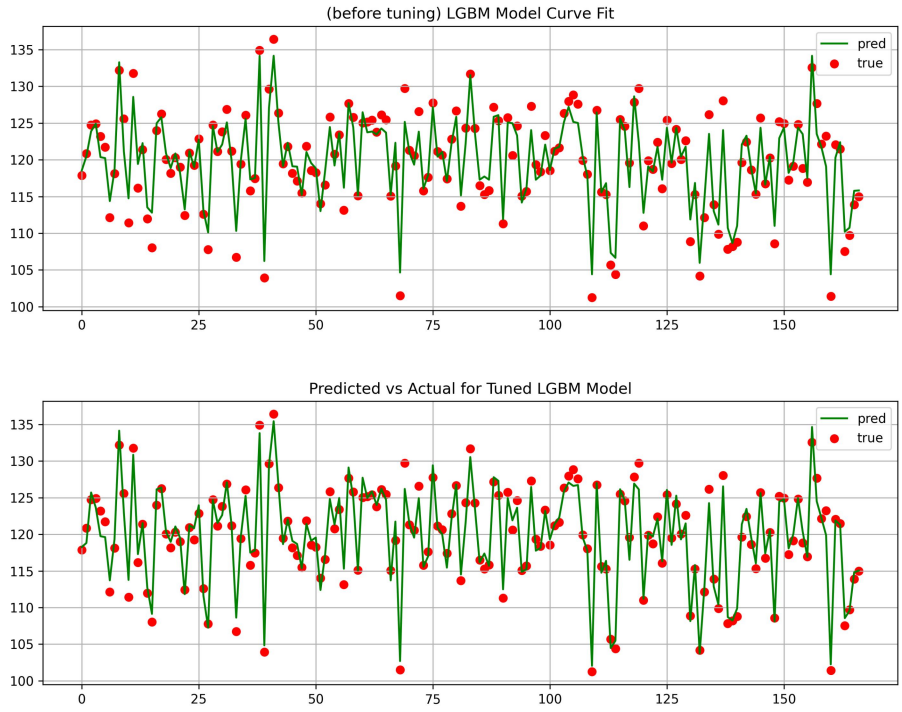

Figure 8-2 LGBM: Predicted vs Actual Plots

Apart from the plots, the evaluation metrics further supported this conclusion, showing a clear improvement in performance after tuning the models (tuned models have lower RMSE, MAPE, MAE, and higher R2 Score).

	Tuned GLM	Untuned GLM	Tuned LGBM	Untuned LGBM
RMSE	2.067	3.387	1.229	1.864
R2 Score	0.908	0.754	0.968	0.926
MAPE	0.013	0.022	0.008	0.012
MAE	1.619	2.599	0.950	1.391

9. Potential Improvement

Although the models performed well, there are still a lot of potential improvement. Here are some suggestions that could improve the models and enhance the overall prediction accuracy.

1) Advanced Tuning Techniques: Recently I am using grid search for model tuning, which can be very slow or inefficient when the dataset or hyperparameter space is large. Therefore, in order to improve the efficiency, some advanced tuning techniques such as RandomizedSearchCV or Bayesian optimization can be considered.
2) Out-of-Sample Testing: In this project, I split dataset into training set and test set, and finished all the model training and tuning process based on the sample data. However, in order to improve the prediction's accuracy and generalization, it is important to evaluate the models on out-of-sample data.