IMPROVING A MODEL USING FEATURE SELECTION METHODS

Khush Domadiya
Chenghui Tan (Sabrina)
Mohanasundaram Murugesan
Amisha Farhana Shaik

APPROACH

- Split the dataset into **training (2004 and earlier)** and **test (2005)** sets.

```
train_data = Smarket[Smarket['Year'] < 2005]
test_data = Smarket[Smarket['Year'] == 2005]

# Display the number of rows in each dataset
print("Training data shape:", train_data.shape)
print("Test data shape:", test_data.shape)

# Optionally, display the first few rows of each dataset
print("\nTraining Data (2004 and earlier):\n", train_data.head())
print("\nTest Data (2005):\n", test_data.head())
```

```
Training data shape: (998, 9)
Test data shape: (252, 9)

Training Data (2004 and earlier):
   Year   Lag1   Lag2   Lag3   Lag4   Lag5  Volume  Today Direction
0  2001  0.381 -0.192 -2.624 -1.055  5.010  1.1913  0.959        Up
1  2001  0.959  0.381 -0.192 -2.624 -1.055  1.2965  1.032        Up
2  2001  1.032  0.959  0.381 -0.192 -2.624  1.4112 -0.623      Down
3  2001 -0.623  1.032  0.959  0.381 -0.192  1.2760  0.614        Up
4  2001  0.614 -0.623  1.032  0.959  0.381  1.2057  0.213        Up

Test Data (2005):
      Year   Lag1   Lag2   Lag3   Lag4   Lag5  Volume  Today Direction
998   2005 -0.134  0.008 -0.007  0.715 -0.431  0.7869 -0.812      Down
999   2005 -0.812 -0.134  0.008 -0.007  0.715  1.5108 -1.167      Down
1000  2005 -1.167 -0.812 -0.134  0.008 -0.007  1.7210 -0.363      Down
1001  2005 -0.363 -1.167 -0.812 -0.134  0.008  1.7389  0.351        Up
1002  2005  0.351 -0.363 -1.167 -0.812 -0.134  1.5691 -0.143      Down
```

LINEAR REGRESSION MODEL TO PREDICT THE PERCENTAGE RETURN.

- Dataset: S&P 500 daily returns data from 2001 to 2005
- Predictor variables : 'Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume'
- Train Test Split : Training dataset of all days from 2004 and earlier. The test set is data from 2005
- Dependent variable : Today i.e the percentage return on that day

CODE FOR LINEAR REGRESSION

```python
# Load the dataset
Smarket = load_data('Smarket')

# Filter the data
train_data = Smarket[Smarket['Year'] <= 2004]
test_data = Smarket[Smarket['Year'] == 2005]
# Features and target variable
features = ['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume']
X_train = train_data[features]
y_train = train_data['Today']
# Add constant to the training features
X_train = sm.add_constant(X_train)

# Initialize and fit the OLS model using statsmodels
ols_model = sm.OLS(y_train, X_train).fit()

# Prepare the test data
X_test = test_data[features]  # Select features for the test set
y_test = test_data['Today']   # Actual values for comparison

# Add constant to the test features
X_test = sm.add_constant(X_test)

# Predict the 'Today' variable using the OLS model
y_pred = ols_model.predict(X_test)

# Print predicted values and compare with actual
#print("Predicted values:", y_pred.values)
#print("Actual values:", y_test.values)

#summary
print(ols_model.summary())

# Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the performance metrics
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Create a dataframe with the features and constant
X_vif = sm.add_constant(X_train)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1])]

print(vif_data)
```

LINEAR REGRESSION RESULTS

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Today   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                 -0.004
Method:                 Least Squares   F-statistic:                    0.3626
Date:                Wed, 25 Sep 2024   Prob (F-statistic):              0.903
Time:                        10:57:34   Log-Likelihood:                -1620.8
No. Observations:                 998   AIC:                             3256.
Df Residuals:                     991   BIC:                             3290.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0146      0.205     -0.071      0.943      -0.417       0.388
Lag1          -0.0217      0.032     -0.684      0.494      -0.084       0.041
Lag2          -0.0106      0.032     -0.332      0.740      -0.073       0.052
Lag3          -0.0033      0.032     -0.104      0.917      -0.066       0.059
Lag4          -0.0060      0.032     -0.188      0.851      -0.068       0.056
Lag5          -0.0394      0.031     -1.254      0.210      -0.101       0.022
Volume         0.0110      0.147      0.075      0.940      -0.278       0.300
==============================================================================
Omnibus:                       49.299   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              143.022
Skew:                           0.165   Prob(JB):                     8.77e-32
Kurtosis:                       4.825   Cond. No.                         11.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Mean Squared Error: 0.42
R-squared: 0.00
```

	Feature	VIF
0	const	27.656125
1	Lag1	1.003460
2	Lag2	1.005034
3	Lag3	1.006164
4	Lag4	1.008210
5	Lag5	1.002516
6	Volume	1.020027

BEST SUBSET SELECTION METHOD

- It involves testing all possible combinations of predictors (features) to find the subset that provides the best performance.
- Methodology:
  - The process involves evaluating all possible combinations of predictors and selecting the subset that minimizes a specified criterion (e.g., residual sum of squares, Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), etc.).For p p predictors, there are $2p$ 2 p possible subsets, making the method computationally intensive for a large number of predictors.

```python
# Step 1: Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
import statsmodels.api as sm
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error
import os

# Load the Smarket data (ensure all years are included)
from ISLP import load_data
Smarket = load_data('Smarket')

# One-hot encode the 'Direction' column
Smarket = pd.get_dummies(Smarket, columns=['Direction'], drop_first=True)

# Add Day column based on index
Smarket['Day'] = Smarket.index.copy()

# Create Year_Day feature (normalize Day within each year)
year_day_max_dict = Smarket.groupby('Year')['Day'].max().to_dict()
Smarket['Year_Day'] = Smarket.apply(lambda x: x['Year'] + x['Day'] / year_day_max_dict[x['Year']], axis=1)

# Display the updated data to ensure all years are included
print("First 5 rows of the data after preprocessing (including all years):")
print(Smarket.head())

# Step 2: Define the feature matrix X and the target variable Y ('Today' as Y)
X = Smarket[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume', 'Year_Day']]
Y = Smarket['Today']  # Target variable is 'Today' (continuous)
```

WHAT IS THE CP STATISTIC?

- The Cp statistic is a measure used to assess the fit of a regression model. It helps to evaluate how well the model explains the variation in the data while accounting for model complexity (number of predictors).
- A lower Cp value indicates a better-fitting model with fewer predictors, aiming to balance goodness of fit and simplicity.

```python
# Step 3: Define Cp statistic function for regression
def Cp_statistic(sigma2, X, Y, model):
    """
    Cp statistic calculation for regression
    sigma2: residual variance
    X: feature matrix
    Y: target variable (continuous)
    model: fitted model
    """
    n, p = X.shape
    Yhat = model.predict(X)
    RSS = np.sum((Y - Yhat) ** 2)
    Cp = (RSS + 2 * p * sigma2) / n
    return Cp
```

WHAT IS FORWARD STEPWISE SELECTION?

- Forward Stepwise Selection is a feature selection method in which predictors are added to a regression model one at a time, based on which predictor improves the model's performance the most (in this case, measured by the Cp statistic). The process stops when adding more predictors does not significantly improve the model's performance.

```python
# Step 4: Forward Stepwise Selection for regression
def forward_stepwise_selection(X, Y, sigma2):
    """
    Forward Stepwise Selection to find the best regression model
    X: feature matrix
    Y: target variable (continuous)
    sigma2: residual variance
    """
    n_predictors = X.shape[1]
    predictors = []
    best_models = []

    for i in range(n_predictors):
        remaining_predictors = list(set(X.columns) - set(predictors))
        best_cp = np.inf
        best_model = None
        for predictor in remaining_predictors:
            temp_predictors = predictors + [predictor]
            X_temp = X[temp_predictors]
            model = sm.OLS(Y, sm.add_constant(X_temp)).fit()
            cp = Cp_statistic(sigma2, sm.add_constant(X_temp), Y, model)
            if cp < best_cp:
                best_cp = cp
                best_model = model
        predictors.append(best_model.model.exog_names[-1])
        best_models.append(best_model)
        print(f"Step {i+1}: Selected predictors = {predictors}, Cp = {best_cp}")

    return best_models
```

OUTPUT OF SUBSET SELECTION

```
Full model sigma^2 (residual variance): 1.2940657780654643
Step 1: Selected predictors = ['Lag5'], Cp = 1.292795243051345
Step 2: Selected predictors = ['Lag5', 'Year_Day'], Cp = 1.2933751857716926
Step 3: Selected predictors = ['Lag5', 'Year_Day', 'Lag1'], Cp = 1.294471095829016
Step 4: Selected predictors = ['Lag5', 'Year_Day', 'Lag1', 'Lag2'], Cp = 1.2963470868481828
Step 5: Selected predictors = ['Lag5', 'Year_Day', 'Lag1', 'Lag2', 'Lag4'], Cp = 1.2983013586068772
Step 6: Selected predictors = ['Lag5', 'Year_Day', 'Lag1', 'Lag2', 'Lag4', 'Volume'], Cp = 1.30032085792603
Step 7: Selected predictors = ['Lag5', 'Year_Day', 'Lag1', 'Lag2', 'Lag4', 'Volume', 'Lag3'], Cp = 1.302347
7990450822
Regression performance metrics for all models:
   Model                                        Predictors       MSE  \
0      1                                            [Lag5]  1.288654
1      2                                  [Lag5, Year_Day]  1.287164
2      3                            [Lag5, Year_Day, Lag1]  1.286189
3      4                      [Lag5, Year_Day, Lag1, Lag2]  1.285995
4      5                [Lag5, Year_Day, Lag1, Lag2, Lag4]  1.285878
5      6        [Lag5, Year_Day, Lag1, Lag2, Lag4, Volume]  1.285827
6      7  [Lag5, Year_Day, Lag1, Lag2, Lag4, Volume, Lag3]  1.285784

   R-squared  Adjusted R-squared
0   0.001215            0.000415
1   0.002371            0.000770
2   0.003126            0.000726
3   0.003277            0.000074
4   0.003367           -0.000639
5   0.003406           -0.001404
6   0.003440           -0.002177
```

```python
# Step 5: Calculate performance metrics for regression (MSE, R-squared, Adjusted R-squared)
def calculate_regression_metrics(models, X, Y):
    """
    Calculate regression performance metrics for each model:
    - MSE, R-squared, Adjusted R-squared
    """
    model_data = []

    for i, model in enumerate(models):
        X_model = sm.add_constant(X[model.model.exog_names[1:]])
        Yhat = model.predict(X_model)

        # Mean Squared Error
        mse = mean_squared_error(Y, Yhat)

        # OLS model statistics
        rsquared = model.rsquared
        rsquared_adj = model.rsquared_adj

        # Store the model data in a dictionary
        model_data.append({
            'Model': i + 1,
            'Predictors': model.model.exog_names[1:],
            'MSE': mse,
            'R-squared': rsquared,
            'Adjusted R-squared': rsquared_adj
        })

    df_models = pd.DataFrame(model_data)
    #print("Regression performance metrics for all models:")
    #print(df_models)

    return df_models

# Step 6: Split the data into training and test sets based on the Year
train_data = Smarket[Smarket['Year'] <= 2004]
test_data = Smarket[Smarket['Year'] == 2005]
```

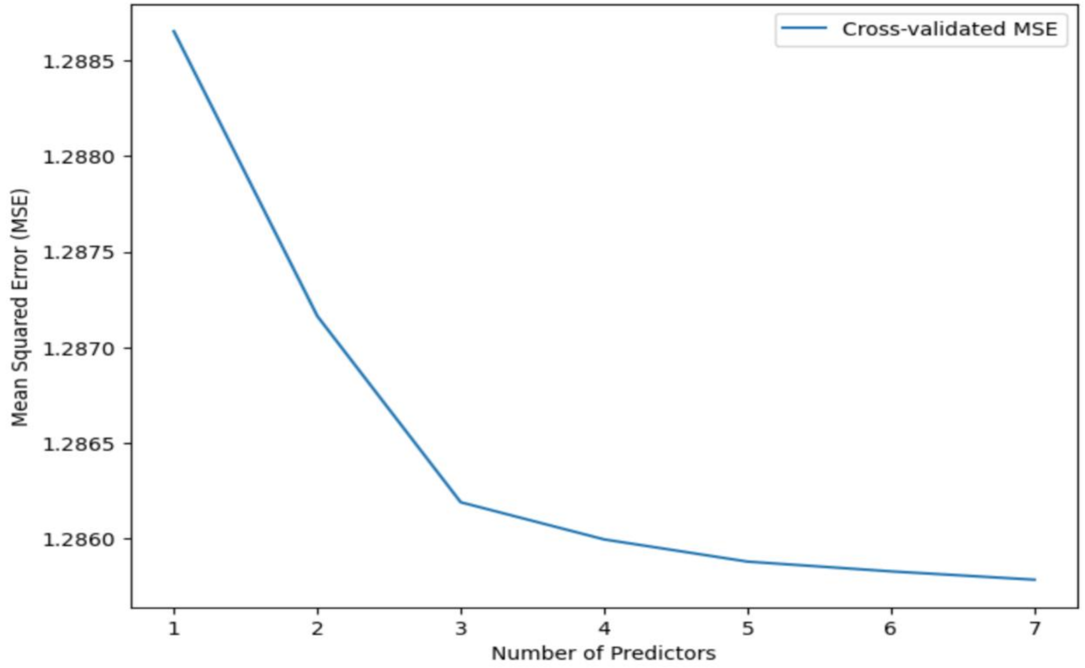
```python
 Step 7: Evaluate each model on training and test data
ef evaluate_on_train_test(models, train_data, test_data, features, target):
   """
   Evaluate each model on both the training and test datasets.
   Calculate the training and test MSE for each model.
   """
   train_mses = []
   test_mses = []

   for i, model in enumerate(models):
       # Extract predictors for current model
       selected_features = model.model.exog_names[1:]  # Ignore constant

       # Train data
       X_train = sm.add_constant(train_data[selected_features])
       Y_train = train_data[target]
       train_pred = model.predict(X_train)
       #train_mse = mean_squared_error(Y_train, train_pred)
       #train_mses.append(train_mse)

       # Test data
       X_test = sm.add_constant(test_data[selected_features])
       Y_test = test_data[target]
       test_pred = model.predict(X_test)
       test_mse = mean_squared_error(Y_test, test_pred)
       test_mses.append(test_mse)

       # Print the results for each model
       print(f"Model {i+1}:")
       print(f"  Predictors: {selected_features}")
       #print(f"  Training MSE: {train_mse}")
       print(f"  Test MSE: {test_mse}")
       print("-" * 40)

   return train_mses, test_mses

 Step 8: Calculate sigma^2 and perform forward stepwise selection
odel_full = sm.OLS(Y, sm.add_constant(X)).fit()
igma2 = model_full.mse_resid
rint(f"Full model sigma^2 (residual variance): {sigma2}")

est_models = forward_stepwise_selection(X, Y, sigma2)

 Step 9: Calculate performance metrics for all regression models
f_models = calculate_regression_metrics(best_models, X, Y)

 Step 10: Evaluate models on train/test data
rain_mses, test_mses = evaluate_on_train_test(best_models, train_data, test_data, features=X.columns, target='Today')

 Step 11: Display the best regression model based on test MSE
est_model_idx = np.argmin(test_mses)
est_model = best_models[best_model_idx]
rint(f"\nBest regression model is Model {best_model_idx+1} with predictors = {best_model.model.exog_names[1:]} and Test MSE = {test_mses[b
```

CROSS-VALIDATION

```python
# Step 6: Cross-validation for regression
def cross_validation(models, X, Y, k=5):
    """
    k-fold cross-validation for regression
    models: all selected models
    X: feature matrix
    Y: target variable (continuous)
    k: number of folds for cross-validation
    """
    kfold = KFold(n_splits=k)
    cv_errors = []

    for i, model in enumerate(models):
        mse_folds = []
        for train_idx, test_idx in kfold.split(X):
            X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
            Y_train, Y_test = Y.iloc[train_idx], Y.iloc[test_idx]
            Yhat = model.predict(sm.add_constant(X_test[model.model.exog_names[1:]]))
            mse = np.mean((Y_test - Yhat) ** 2)
            mse_folds.append(mse)
        cv_error = np.mean(mse_folds)
        cv_errors.append(cv_error)
        print(f"Model {i+1}: Cross-validated MSE = {cv_error}")

    return cv_errors
```

REGRESSION PERFORMANCE METRICS FOR ALL MODELS

```
Regression performance metrics for all models:
   Model                                        Predictors       MSE  \
0      1                                            [Lag5]  1.288654   
1      2                                 [Lag5, Year_Day]  1.287164   
2      3                           [Lag5, Year_Day, Lag1]  1.286189   
3      4                     [Lag5, Year_Day, Lag1, Lag2]  1.285995   
4      5               [Lag5, Year_Day, Lag1, Lag2, Lag4]  1.285878   
5      6       [Lag5, Year_Day, Lag1, Lag2, Lag4, Volume]  1.285827   
6      7  [Lag5, Year_Day, Lag1, Lag2, Lag4, Volume, Lag3]  1.285784   

   R-squared  Adjusted R-squared  
0   0.001215            0.000415  
1   0.002371            0.000770  
2   0.003126            0.000726  
3   0.003277            0.000074  
4   0.003367           -0.000639  
5   0.003406           -0.001404  
6   0.003440           -0.002177  
Model 1: Cross-validated MSE = 1.2886542325615356
Model 2: Cross-validated MSE = 1.2871636700369786
Model 3: Cross-validated MSE = 1.286189074849397
Model 4: Cross-validated MSE = 1.285994560623659
Model 5: Cross-validated MSE = 1.2858783271374485
Model 6: Cross-validated MSE = 1.2858273212116966
Model 7: Cross-validated MSE = 1.2857837570858446
```



OUTPUT COMPARISION

- THE CASE ASSIGNMENT 3 RESULT:
- TEST MSE: 0.41749641
- AFTER SUBSET SELECTION AND VALIDATION RESULT:
- BEST MODEL TEST MSE: 0.419565142

IT SEEMS NOT IMPROVED

BECAUSE IN CASE ASSIGNMENT 3 THE MODEL ONLY TRAIN ONCE

THE MODEL IS NOT VALIDATED

BUT NOW IN CASE ASSIGNMENT 4 THE MODEL VALIDATED, EVEN WITH SLIGHTLY HIGH MSE.

```
Best regression model is Model 7 with predictors = ['Lag5', 'Year_Day', 'Lag1', 'Lag2', 'Lag4', 'Volume',
'Lag3'] and Cross-validated MSE = 1.2857837570858446
Test MSE for the best regression model: 0.41956514212307966
```

LASSO REGRESSION

```python
# Scale the features (important for Lasso, as it is sensitive to feature scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Prepare the test data and scale it
X_test = test_data[features]
y_test = test_data['Today']
X_test_scaled = scaler.transform(X_test)

# Initialize and fit the Lasso model
lasso_model = Lasso(alpha=0.1)  # Adjust alpha for stronger or weaker regularization
lasso_model.fit(X_train_scaled, y_train)

# Predict the 'Today' variable using the Lasso model
y_pred = lasso_model.predict(X_test_scaled)

# Print predicted values and compare with actual
#print("Predicted values:", y_pred)
#print("Actual values:", y_test.values)

# Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the performance metrics
print(f"Mean Squared Error (Lasso): {mse:.2f}")
print(f"R-squared (Lasso): {r2:.2f}")

# Print the coefficients of the Lasso model
print("Lasso Coefficients:", lasso_model.coef_)
```

```
Mean Squared Error (Lasso): 0.42
R-squared (Lasso): -0.00
Lasso Coefficients: [-0. -0. -0. -0. -0.  0.]
```

LASSO FEATURE ENGINEERING

```python
# Create new lag difference features
train_data['LagDiff1'] = train_data['Lag1'] - train_data['Lag2']
train_data['LagDiff2'] = train_data['Lag2'] - train_data['Lag3']
test_data['LagDiff1'] = test_data['Lag1'] - test_data['Lag2']
test_data['LagDiff2'] = test_data['Lag2'] - test_data['Lag3']

# Combine new features with existing features
X_train_new = np.hstack((X_train_poly, train_data[['LagDiff1', 'LagDiff2']].values))
X_test_new = np.hstack((X_test_poly, test_data[['LagDiff1', 'LagDiff2']].values))
y_train = train_data['Today']
y_test = test_data['Today']

# Initialize and fit the Lasso model
lasso_model = Lasso(alpha=0.01)
lasso_model.fit(X_train_new, y_train)

# Predict the 'Today' variable using the Lasso model
y_pred_lasso = lasso_model.predict(X_test_new)

# Print Lasso coefficients
print("Lasso Coefficients:", lasso_model.coef_)

# Calculate performance metrics
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

# Print performance metrics
print(f"Mean Squared Error (Lasso with Feature Engineering): {mse_lasso:.2f}")
print(f"R-squared (Lasso with Feature Engineering): {r2_lasso:.2f}")
```

```
Lasso Coefficients: [-0.         -0.          0.          0.         -0.         -0.
 -0.00966744  0.00355625  0.0511209  -0.01267454 -0.0404856  -0.
  0.01604667  0.01550881 -0.00783062  0.         -0.          0.00749583
 -0.04171899  0.         -0.03264433 -0.00297372  0.03489854 -0.
  0.02532142 -0.02975583 -0.03203691 -0.         -0.01757618]
Mean Squared Error (Lasso with Feature Engineering): 0.44
R-squared (Lasso with Feature Engineering): -0.06
```

RIDGE REGRESSION

```python
# Define the alpha (regularization strength) for Ridge Regression
alpha_value = 100.0  # You can experiment with this value (e.g., 0.1, 1.0, 10.0)

# Initialize the Ridge regression model
ridge_model = Ridge(alpha=alpha_value)

# Fit the Ridge model using the training data
ridge_model.fit(X_train, y_train)

# Predict the 'Today' variable using the Ridge model
y_pred_ridge = ridge_model.predict(X_test)

# Print predicted values and compare with actual
print("Predicted values (Ridge):", y_pred_ridge)

# Calculate performance metrics for Ridge regression
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)
```

```
Ridge Regression Mean Squared Error: 0.42
Ridge Regression R-squared: 0.00
```

ELASTIC NET REGRESSION

```python
# Step 4: Standardize the features (Elastic Net requires scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 5: Define Elastic Net model with Grid Search for hyperparameter tuning
elastic_net = ElasticNet()

# Define the parameter grid for alpha (regularization strength) and l1_ratio (mix between Lasso and Ridge)
param_grid = {
    'alpha': [0.01, 0.1, 1, 10, 100],  # regularization strength
    'l1_ratio': [0.1, 0.5, 0.7, 1.0]   # balance between Lasso and Ridge (L1 and L2 regularization)
}

# Step 6: GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(elastic_net, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train_scaled, y_train)

# Best parameters
best_alpha = grid_search.best_params_['alpha']
best_l1_ratio = grid_search.best_params_['l1_ratio']
print(f"Best alpha: {best_alpha}")
print(f"Best l1_ratio: {best_l1_ratio}")

# Step 7: Train the best Elastic Net model
elastic_net_best = ElasticNet(alpha=best_alpha, l1_ratio=best_l1_ratio)
elastic_net_best.fit(X_train_scaled, y_train)

# Step 8: Predict the 'Today' variable using the trained Elastic Net model
y_pred_en = elastic_net_best.predict(X_test_scaled)
```

```
Best alpha: 0.1
Best l1_ratio: 1.0
Mean Squared Error (Elastic Net): 0.42
R-squared (Elastic Net): -0.00
Mean Squared Error (OLS): 0.42
R-squared (OLS): 0.00
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Today   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                 -0.004
Method:                 Least Squares   F-statistic:                    0.3626
Date:                Mon, 30 Sep 2024   Prob (F-statistic):              0.903
Time:                        15:26:29   Log-Likelihood:                -1620.8
No. Observations:                 998   AIC:                             3256.
Df Residuals:                     991   BIC:                             3290.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0146      0.205     -0.071      0.943      -0.417       0.388
Lag1          -0.0217      0.032     -0.684      0.494      -0.084       0.041
Lag2          -0.0106      0.032     -0.332      0.740      -0.073       0.052
Lag3          -0.0033      0.032     -0.104      0.917      -0.066       0.059
Lag4          -0.0060      0.032     -0.188      0.851      -0.068       0.056
Lag5          -0.0394      0.031     -1.254      0.210      -0.101       0.022
Volume         0.0110      0.147      0.075      0.940      -0.278       0.300
==============================================================================
Omnibus:                       49.299   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              143.022
Skew:                           0.165   Prob(JB):                     8.77e-32
Kurtosis:                       4.825   Cond. No.                         11.0
==============================================================================
```

RECURSIVE FEATURE ELIMINATION

- RFE is a technique used to select features by recursively considering smaller sets of features.
- It uses a model like Linear Regression to evaluate the importance of each feature, eliminating the least important features until the desired number of features is reached.

```
Selected features: Index(['Lag1', 'Lag5', 'Volume'], dtype='object')
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Today   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                 -0.001
Method:                 Least Squares   F-statistic:                    0.6765
Date:                Tue, 01 Oct 2024   Prob (F-statistic):              0.566
Time:                        08:42:28   Log-Likelihood:                -1620.9
No. Observations:                 998   AIC:                             3250.
Df Residuals:                     994   BIC:                             3269.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0230      0.203     -0.113      0.910      -0.422       0.376
Lag1          -0.0215      0.032     -0.679      0.498      -0.084       0.041
Lag5          -0.0391      0.031     -1.247      0.213      -0.101       0.022
Volume         0.0172      0.146      0.118      0.906      -0.269       0.304
==============================================================================
Omnibus:                       50.088   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              145.046
Skew:                           0.174   Prob(JB):                     3.19e-32
Kurtosis:                       4.835   Cond. No.                         10.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Mean Squared Error (RFE): 0.42
R-squared (RFE): 0.00
```

RECURSIVE FEATURE ELIMINATION

```python
# Perform RFE with the model and select a certain number of features (e.g., 3)
n_features_to_select = 3  # You can adjust this number
rfe = RFE(estimator=model, n_features_to_select=n_features_to_select)
rfe.fit(X_train, y_train)

# Get the selected features
selected_features = X_train.columns[rfe.support_]
print("Selected features:", selected_features)

# Prepare the training and test sets with the selected features
X_train_rfe = X_train[selected_features]
X_test_rfe = X_test[selected_features]

# Add constant to the training features for OLS
X_train_rfe = sm.add_constant(X_train_rfe)

# Fit the OLS model
ols_model_rfe = sm.OLS(y_train, X_train_rfe).fit()

# Add constant to the test features for OLS
X_test_rfe = sm.add_constant(X_test_rfe)

# Predict the 'Today' variable using the OLS model
y_pred_rfe = ols_model_rfe.predict(X_test_rfe)

# Print the summary of the RFE model
print(ols_model_rfe.summary())

# Calculate performance metrics
mse_rfe = mean_squared_error(y_test, y_pred_rfe)
r2_rfe = r2_score(y_test, y_pred_rfe)

# Print the performance metrics
print(f"Mean Squared Error (RFE): {mse_rfe:.2f}")
print(f"R-squared (RFE): {r2_rfe:.2f}")
```