Project Title: Predictive Trading Strategy Development for VDE ETF Using Machine Learning

By:

Abhishek Patil

Introduction:

This report details a machine learning-based stock trading strategy that aims to predict key price levels for the next trading day. The core strategy involves buying at the daily lows and selling at the close to capture intraday volatility. Multiple predictive models are trained on historical price data to forecast the opening, high, low, and closing prices for the next day. Trading signals are generated via a random forest classifier to determine optimal positioning. The models are backtested over multi-year market data.

The trading instruments focused on are exchange-traded funds (ETFs) that represent the energy sector. The VDE ETF, which tracks oil and gas stocks, serves as the sample target for modeling. The time range tested spans from 2004 to 2023, covering varied economic environments. Key metrics like the daily price range and returns distribution are derived to quantify opportunity.

The underlying hypothesis is that machine learning algorithms can reliably predict short-term fluctuations in this ETF to facilitate systematic and profitable trading based on the forecasted price levels. Capturing daily swings from low-to-high offers significant reward potential. The combination of data science, financial engineering, and execution automation may unlock this recurring alpha.

This is to decide which machine learning model to use based on the results of this experiment:

```
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LogisticRegression
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
```

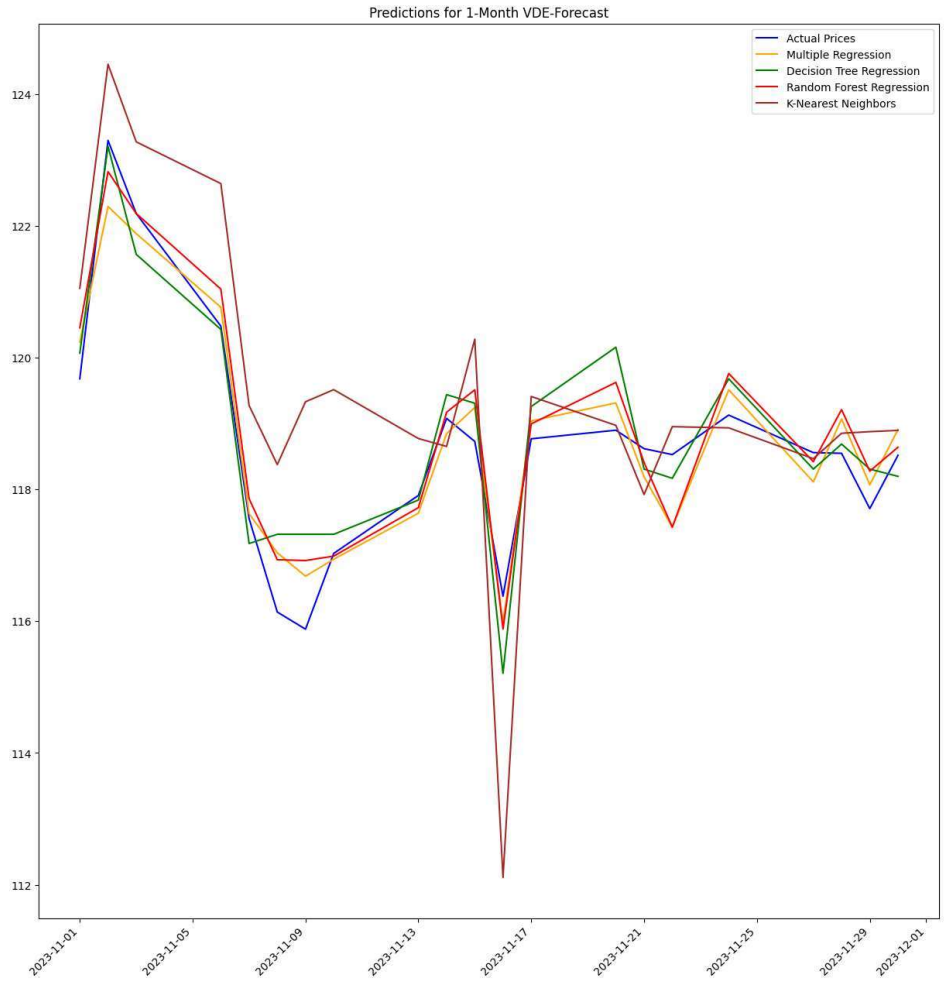
Testing different Machine Learning model and their predictive power for vde across time frame and the strength of my underlying features.

```
# Performance Metrics function
def calculate_metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return mae, mse, rmse, r2, mape


# Download historical data for SPY
vde = yf.download('VDE', start='2000-01-01', end='2023-12-01')
vde['SMA_5'] = vde['Close'].rolling(window=5).mean()
vde['SMA_21'] = vde['Close'].rolling(window=21).mean()
vde['Daily_Return'] = vde['Close'].pct_change()
vde.dropna(inplace=True)
vde["Move"] = vde["High"] - vde["Low"]
vde["Move_per"] = vde["Move"] / vde["Low"]
vde['Target'] = np.where(vde['Daily_Return'] > 0, 1, 0)
vde.tail()
# spy_data.head()

features = ["Open", "High", "Low","Volume","Daily_Return", "Move", "Move_per","SMA_5", "SMA_21"]
X = vde[features]
y = vde['Close']

#List of training end dates
dates = ['2023-11-01', '2023-09-01', '2023-06-01', '2022-12-01']
pred = ['1-Month', '3-Months', '6-Months', '1- Year']
# Create an empty DataFrame to store performance metrics
all_metrics_df = pd.DataFrame(columns=['MAE', 'MSE', 'RMSE', 'R-Squared', 'MAPE'])

# Iterate #through each training end date
for train_end_date, i in zip(dates, pred):
    # Train-test split
    X_train = X[X.index < train_end_date]
    y_train = y[y.index < train_end_date]
    X_test = X[X.index >= train_end_date]
    y_test = y[y.index >= train_end_date]

    # Standardize features
    scaler = StandardScaler()
    X train scaled = scaler.fit transform(X train)
```

```
    X_test_scaled = scaler.transform(X_test)

    # Multiple Regression
    lr_model = LinearRegression()
    lr_model.fit(X_train_scaled, y_train)
    lr_predictions = lr_model.predict(X_test_scaled)

    # Decision Tree Regression
    dt_model = DecisionTreeRegressor()
    dt_model.fit(X_train_scaled, y_train)
    dt_predictions = dt_model.predict(X_test_scaled)

    # Random Forest Regression
    rf_model = RandomForestRegressor()
    rf_model.fit(X_train_scaled, y_train)
    rf_predictions = rf_model.predict(X_test_scaled)

    # K-Nearest Neighbors Regression
    knn_model = KNeighborsRegressor()
    knn_model.fit(X_train_scaled, y_train)
    knn_predictions = knn_model.predict(X_test_scaled)



    # Calculate metrics for each model
    lr_metrics = calculate_metrics(y_test, lr_predictions)
    dt_metrics = calculate_metrics(y_test, dt_predictions)
    rf_metrics = calculate_metrics(y_test, rf_predictions)
    knn_metrics = calculate_metrics(y_test, knn_predictions)


    # Plotting
    plt.figure(figsize=(10, 6))
    plt.plot(y_test.index, y_test, label='Actual Prices', color='blue')
    plt.plot(y_test.index, lr_predictions, label='Multiple Regression', color='orange')
    plt.plot(y_test.index, dt_predictions, label='Decision Tree Regression', color='green')
    plt.plot(y_test.index, rf_predictions, label='Random Forest Regression', color='red')
    plt.plot(y_test.index, knn_predictions, label='K-Nearest Neighbors', color='brown')
    plt.xticks(rotation=45, ha='right')
    plt.title(f'Predictions for {i} VDE-Forecast')
    plt.legend()

    plt.show()

    # Compare Performance Metrics
    models = ['Multiple Regression', 'Decision Tree', 'Random Forest','K-NN']
    metrics_df = pd.DataFrame([lr_metrics, dt_metrics, rf_metrics,knn_metrics],
                          columns=['MAE', 'MSE', 'RMSE', 'R-Squared', 'MAPE'], index=models)

    # Display overall performance metrics
    print(f"\nOverall Performance Metrics for {i} VDE-Forecast:")
    print(f"{metrics_df}\n")


#########################################################################################################
### Residual Analysis

    # Calculate residuals for each model
    lr_residuals = y_test - lr_predictions
    dt_residuals = y_test - dt_predictions
    rf_residuals = y_test - rf_predictions
    knn_residuals = y_test - knn_predictions


    # Plot Residuals
    plt.figure(figsize=(8, 6))

    plt.subplot(2, 2, 1)
    plt.scatter(y_test, lr_residuals, color='orange')
    plt.axhline(y=0, color='black', linestyle='--', linewidth=2)
    plt.title(f'Multiple Regression Residuals ({i})')

    plt.subplot(2, 2, 2)
    plt.scatter(y_test, dt_residuals, color='green')
```

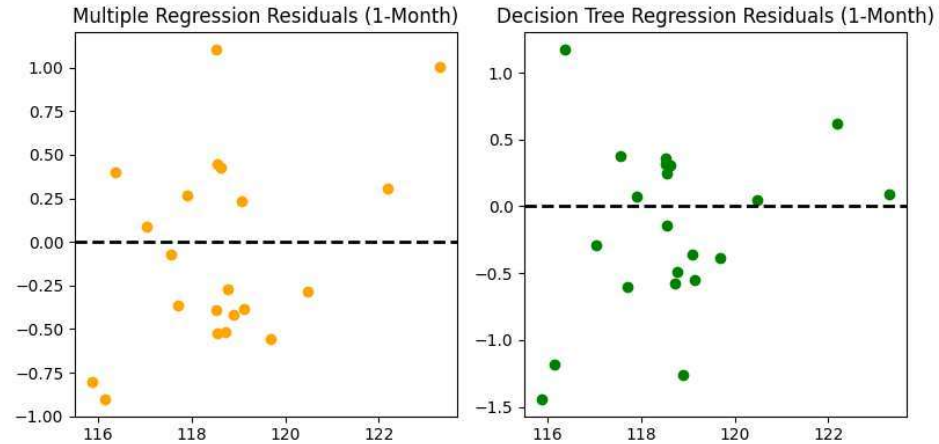
```
plt.scatter(y_test, dt_residuals, color='green')
plt.axhline(y=0, color='black', linestyle='--', linewidth=2)
plt.title(f'Decision Tree Regression Residuals ({i})')

plt.subplot(2, 2, 3)
plt.scatter(y_test, rf_residuals, color='red')
plt.axhline(y=0, color='black', linestyle='--', linewidth=2)
plt.title(f'Random Forest Regression Residuals ({i})')

plt.subplot(2, 2, 4)
plt.scatter(y_test, knn_residuals, color='brown')
plt.axhline(y=0, color='black', linestyle='--', linewidth=2)
plt.title(f'K-Nearest Neighbors Residuals ({i})')



plt.tight_layout()
plt.show()

# Display Residual Statistics
residual_stats = pd.DataFrame({
'Mean Residual': [lr_residuals.mean(), dt_residuals.mean(), rf_residuals.mean(), knn_residuals.mean()],
'Std Dev Residual': [lr_residuals.std(), dt_residuals.std(), rf_residuals.std(), knn_residuals.std()]
}, index=models)

print(f"\nResidual Statistics for {i} VDE-Forecast:")
print(f"{residual_stats}\n")
```

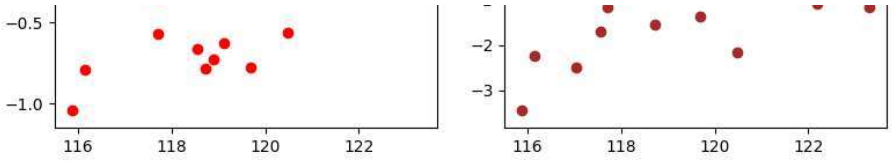
```
[*********************100%%**********************]  1 of 1 completed
```



Overall Performance Metrics for 1-Month VDE-Forecast:

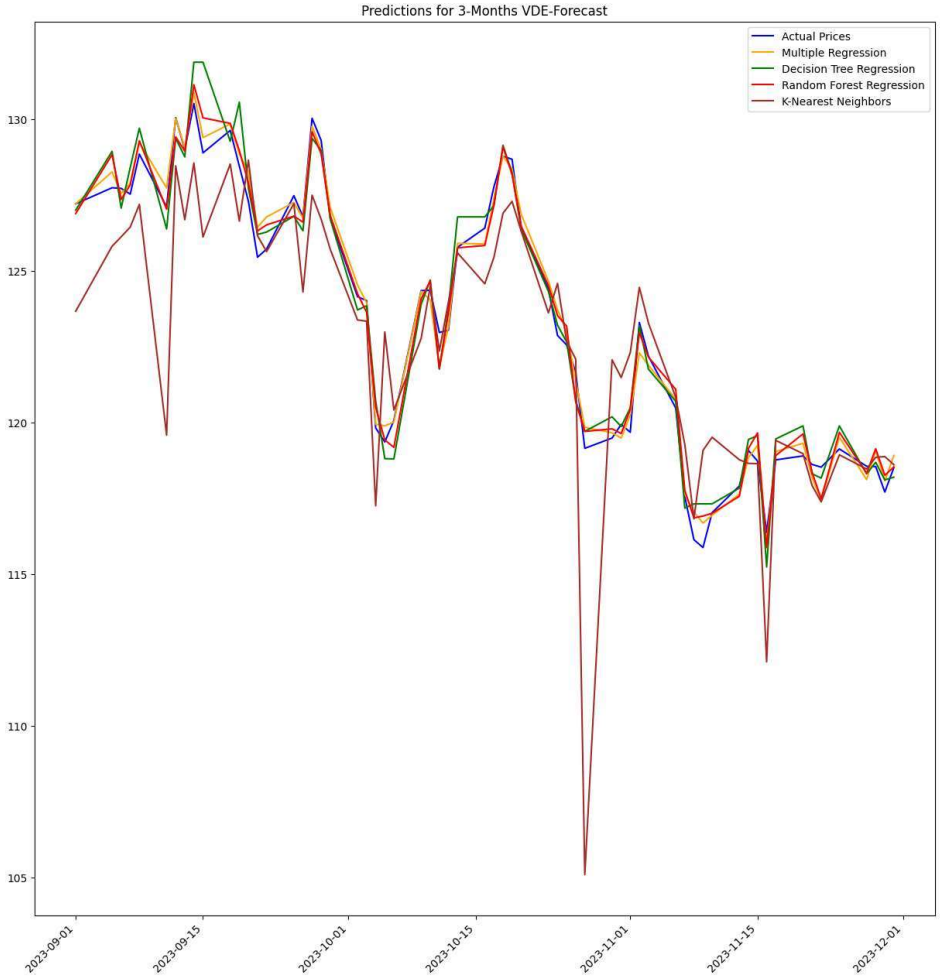
	MAE	MSE	RMSE	R-Squared	MAPE
Multiple Regression	0.464592	0.288573	0.537190	0.904540	0.391393
Decision Tree	0.519049	0.427706	0.653992	0.858515	0.440034
Random Forest	0.473238	0.327941	0.572661	0.891517	0.399838
K-NN	1.274191	2.830366	1.682369	0.063715	1.080192





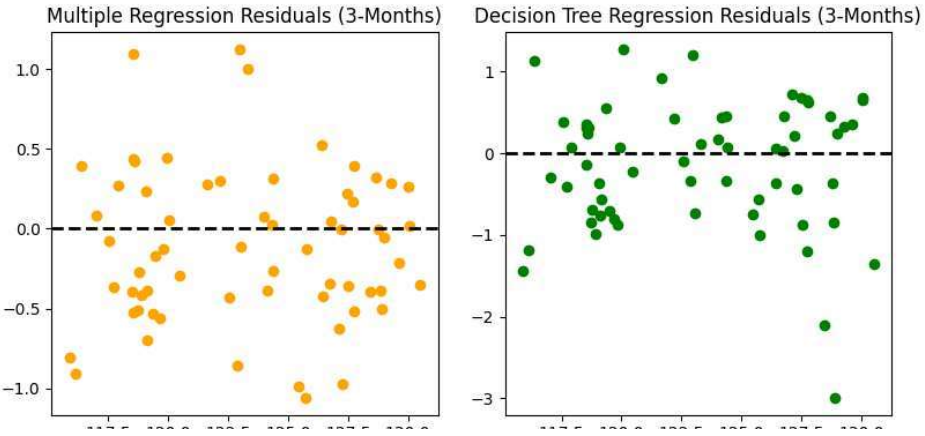
Residual Statistics for 1-Month VDE-Forecast:

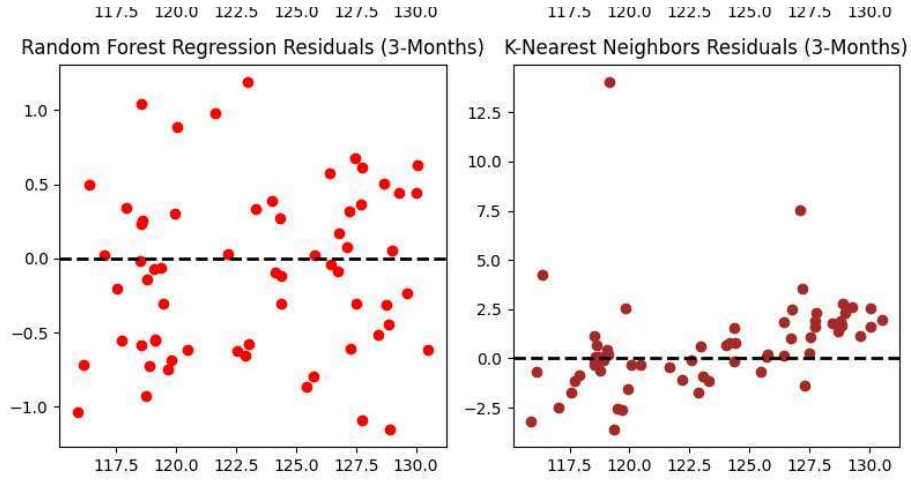
	Mean Residual	Std Dev Residual
Multiple Regression	-0.057679	0.547274
Decision Tree	-0.174285	0.645908
Random Forest	-0.221010	0.541342
K-NN	-0.733429	1.551474



Overall Performance Metrics for 3-Months VDE-Forecast:

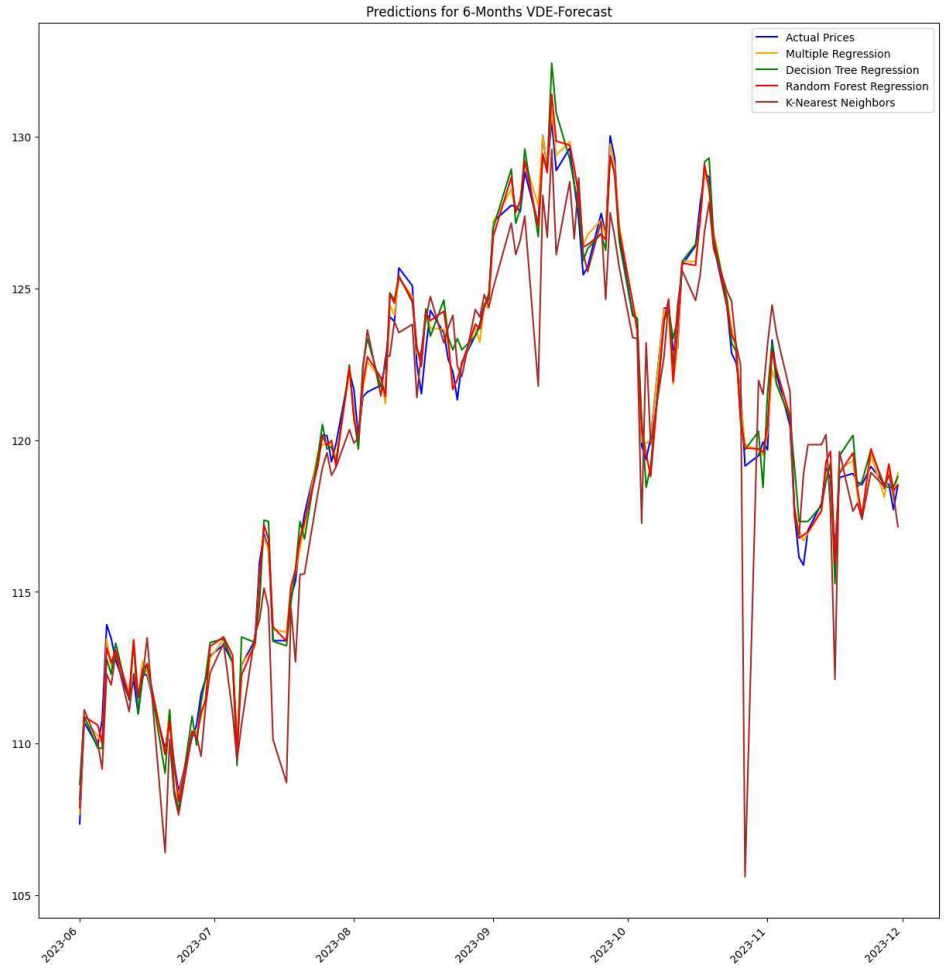
	MAE	MSE	RMSE	R-Squared	MAPE
Multiple Regression	0.400922	0.243617	0.493576	0.987278	0.326958
Decision Tree	0.623969	0.639733	0.799833	0.966593	0.505048
Random Forest	0.470016	0.318967	0.564771	0.983343	0.382071
K-NN	1.643048	6.786417	2.605075	0.645608	1.333476





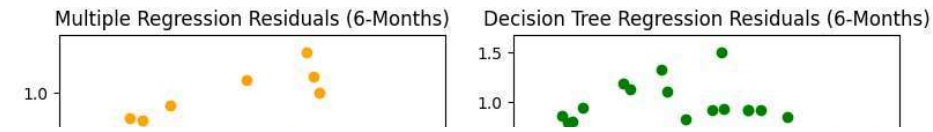
Residual Statistics for 3-Months VDE-Forecast:

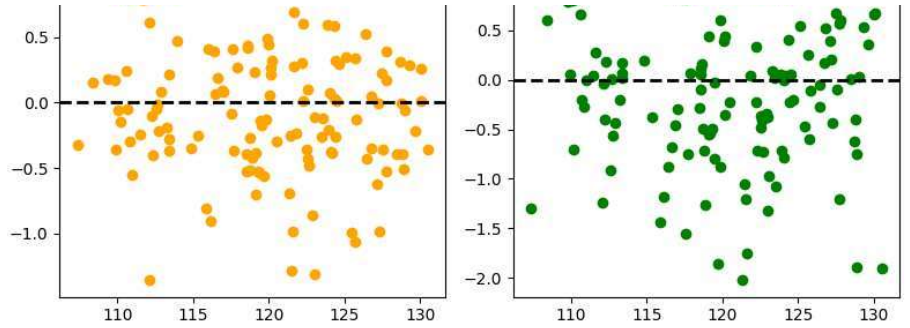
	Mean Residual	Std Dev Residual
Multiple Regression	-0.120882	0.482388
Decision Tree	-0.157618	0.790447
Random Forest	-0.099911	0.560329
K-NN	0.692572	2.531498

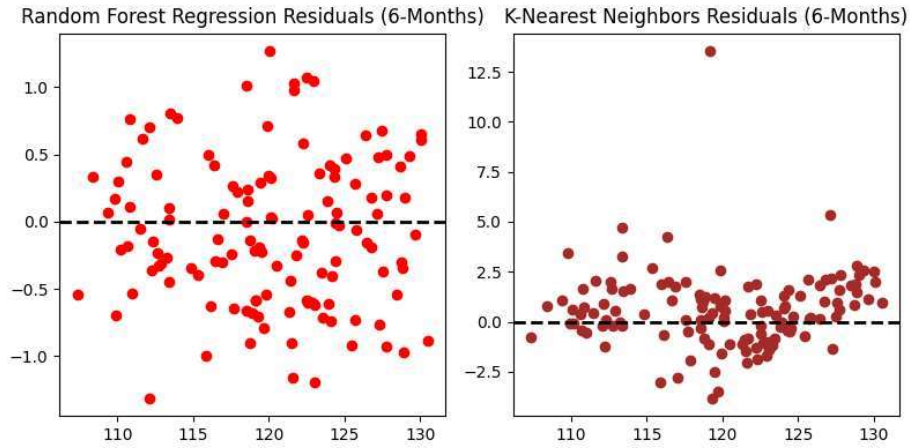


Overall Performance Metrics for 6-Months VDE-Forecast:

	MAE	MSE	RMSE	R-Squared	MAPE
Multiple Regression	0.394216	0.249849	0.499849	0.992689	0.328387
Decision Tree	0.583623	0.561217	0.749144	0.983577	0.487279
Random Forest	0.458448	0.304596	0.551902	0.991087	0.381305
K-NN	1.352945	4.025064	2.006256	0.882216	1.126357

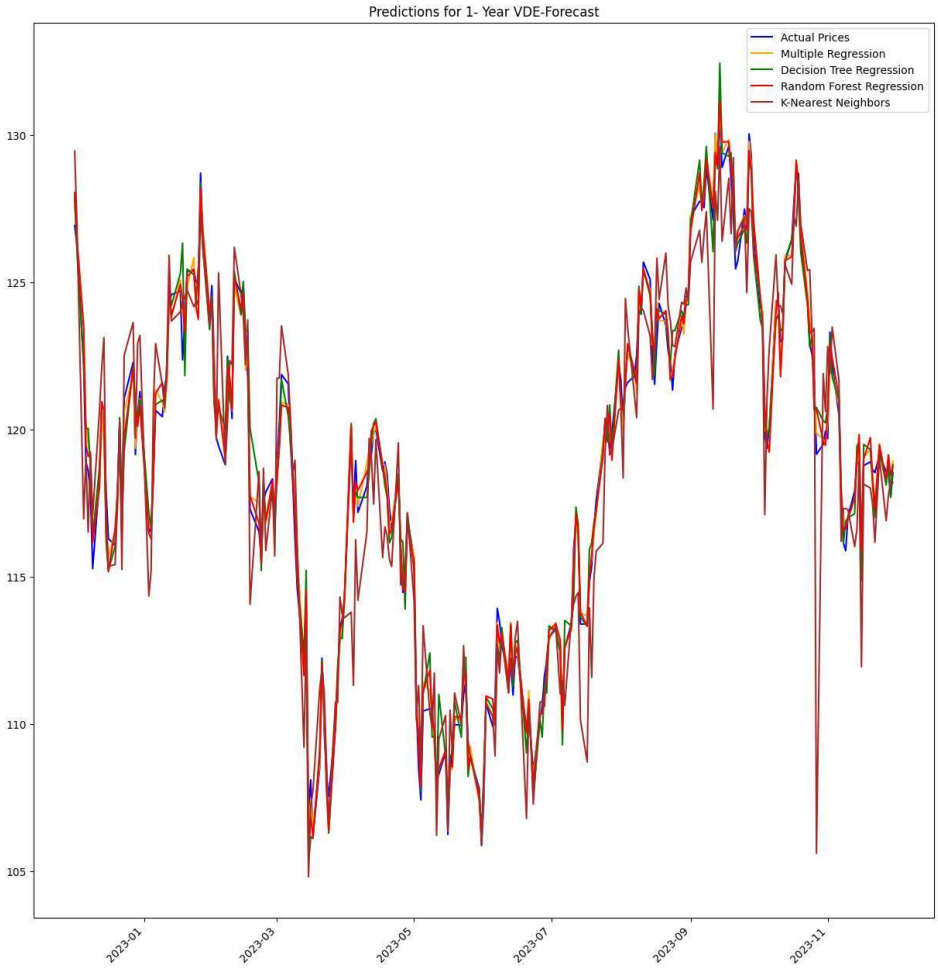






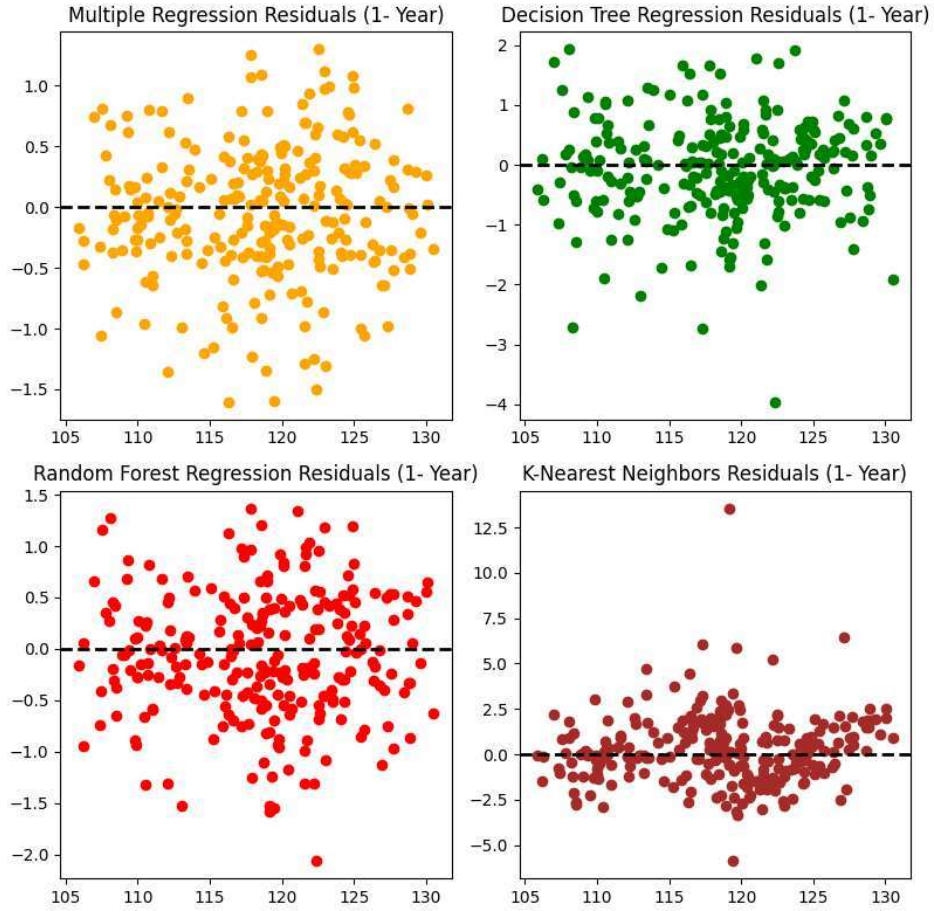
Residual Statistics for 6-Months VDE-Forecast:

	Mean Residual	Std Dev Residual
Multiple Regression	-0.066967	0.497305
Decision Tree	-0.144567	0.737974
Random Forest	-0.084294	0.547587
K-NN	0.582362	1.927478



Overall Performance Metrics for 1- Year VDE-Forecast:

	MAE	MSE	RMSE	R-Squared	MAPE
Multiple Regression	0.437480	0.309554	0.556376	0.991164	0.368915
Decision Tree	0.625378	0.694186	0.833178	0.980185	0.530257
Random Forest	0.493455	0.381037	0.617282	0.989123	0.416010
K-NN	1.390430	3.782882	1.944963	0.892019	1.170747



Residual Statistics for 1- Year VDE-Forecast:

	Mean Residual	Std Dev Residual
Multiple Regression	-0.056535	0.554602
Decision Tree	-0.079163	0.831066
Random Forest	-0.065054	0.615071
K-NN	0.263307	1.930908

Cross Validating the Best Performimg models Multiple Regression and Random Forest

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit

# Define the number of splits for TimeSeriesSplit
n_splits = 10  # You can adjust this based on your preference

# Create TimeSeriesSplit object
tscv = TimeSeriesSplit(n_splits=n_splits)

train_end_date = '2023-11-01'
# Train-test split
X_train = X[X.index < train_end_date]
y_train = y[y.index < train_end_date]
X_test = X[X.index >= train_end_date]
y_test = y[y.index >= train_end_date]

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create empty DataFrames to store performance metrics for each fold
all_metrics_df_fold = pd.DataFrame(columns=['MAE', 'MSE', 'RMSE', 'R-Squared', 'MAPE'])

# Perform cross-validation
for train_index, test_index in tscv.split(X_train_scaled):
    X_train_cv, X_test_cv = X_train_scaled[train_index], X_train_scaled[test_index]
    y_train_cv, y_test_cv = y_train.iloc[train_index], y_train.iloc[test_index]

    # Fit the model on the training data
    lr_model = LinearRegression()
    lr_model.fit(X_train_cv, y_train_cv)

    # Make predictions on the test data
    lr_predictions_cv = lr_model.predict(X_test_cv)

    # Calculate metrics for each fold
    lr_metrics_cv = calculate_metrics(y_test_cv, lr_predictions_cv)
    # Append metrics for this fold to the DataFrame
    all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_i

# Calculate average metrics across all folds
avg_metrics_cv = all_metrics_df_fold.mean()

# Display overall performance metrics for this training end date
print(f"\nOverall Cross-Validated Performance Metrics for Multiple Linear Regression Model VDE-Forecast:")
print(f"{avg_metrics_cv}\n")
```

```
Overall Cross-Validated Performance Metrics for Multiple Linear Regression Model VDE-Forecast:
MAE          0.349411
MSE          0.238329
RMSE         0.474046
R-Squared    0.998008
```

```
MAPE          0.401087
dtype: float64

<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-4-89f15a40e795>:40: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
```

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit

# Define the number of splits for TimeSeriesSplit
n_splits = 10  # You can adjust this based on your preference

# Create TimeSeriesSplit object
tscv = TimeSeriesSplit(n_splits=n_splits)

train_end_date = '2023-11-01'
# Train-test split
X_train = X[X.index < train_end_date]
y_train = y[y.index < train_end_date]
X_test = X[X.index >= train_end_date]
y_test = y[y.index >= train_end_date]

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create empty DataFrames to store performance metrics for each fold
all_metrics_df_fold = pd.DataFrame(columns=['MAE', 'MSE', 'RMSE', 'R-Squared', 'MAPE'])

# Perform cross-validation
for train_index, test_index in tscv.split(X_train_scaled):
    X_train_cv, X_test_cv = X_train_scaled[train_index], X_train_scaled[test_index]
    y_train_cv, y_test_cv = y_train.iloc[train_index], y_train.iloc[test_index]

    # Random Forest Regression
    rf_model = RandomForestRegressor()
    rf_model.fit(X_train_cv, y_train_cv)
    rf_predictions_cv = rf_model.predict(X_test_cv)

    # Calculate metrics for each fold
    rf_metrics_cv = calculate_metrics(y_test_cv, rf_predictions_cv)

    # Append metrics for this fold to the DataFrame
    all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_i

# Calculate average metrics across all folds
avg_metrics_cv = all_metrics_df_fold.mean()

# Display overall performance metrics for this training end date
print(f"\nOverall Cross-Validated Performance Metrics for Random Forest Model VDE Forecast:")
print(f"{avg_metrics_cv}\n")
```

```
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
```

```
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_

Overall Cross-Validated Performance Metrics for Random Forest Model VDE Forecast:
MAE          0.510490
MSE          0.424922
RMSE         0.651860
R-Squared    0.994892
MAPE         0.451880
dtype: float64

<ipython-input-5-30fa4dc17fd2>:39: FutureWarning: The frame.append method is deprecated and will be removed from panda
  all_metrics_df_fold = all_metrics_df_fold.append(pd.Series(lr_metrics_cv, index=all_metrics_df_fold.columns), ignore_
```

The results indicate that both Multiple Regression and Random Forest models perform exceptionally well across various time-based characteristics. We will utilize the identical model to forecast the future prices for our trading approach.

Start coding or generate with AI.

Methodology:

Dataset Foundation:

The machine learning models are implemented on historical stock data for the Vanguard Energy ETF (VDE). This provides exposure to the overall energy sector while diversifying across multiple underlying oil/gas/fuel companies.

The raw price data spans from the inception date of VDE in 2004 to current day, covering nearly 20 years of varying market environments. This practices effective strategy robustness testing across bull/bear cycles.

For each day in the time range, the open, high, low, close prices, along with the daily volume traded constitutes the elementary feature inputs. This OHLCV data forms a standard foundation for quant strategies.

Feature Engineering:

Additionally, an insightful custom metric called Move_per is calculated from raw values, designed to capture intraday swings as a percentage of lows. On each day, the absolute difference between the closing price and low price is measured, then divided by the low to represent swing magnitude. Higher values signal wider upside bounces from oversold zones.

This Move_per indicator compliments other features like volatility and daily returns to encode the concept of tradable intraday range - the very essence of the strategy. Wider ranges with substantive swings between support and resistance extremes tends to enable profitable momentum trading. The feature targets this effect.

By training models on a blend of basic OHLCV data and specialized derivatives like Move_per that are strategically meaningful, the algorithms maximize predictive power from underlying signals while retaining generalization ability to evolving markets. Careful data selection and engineering aims to capture winning edges formulaically.

Random Forest Classifier Model:

The key purpose of the random forest classifier is to analyze recent price action and other inputs to determine optimal positioning for the next trading session - either taking a long position to capitalize on upside potential or staying neutral/short if downside is expected.

The model inputs encompass features that capture the current market state - latest closing prices, trends, volatility, volume etc. The target variable the classifier is trained on is derived

from historical daily returns. Essentially, if returns were positive on a given past day, that day receives a target label of 1. If returns were negative, the target is 0.

So the model learns the patterns in features like volatility and momentum that preceded positive or negative return days. This picks up on the dynamics that typically signal bullish or bearish outcomes.

The trained model can then take today's most recent real-time data and check if the current state aligns more closely with historical instances that brought upside or downside subsequently. Based on probabilistic similarity scores, it assigns a prediction of 1 for expected positive returns tomorrow or 0 for negative.

Finally, this 1 or 0 signal maps to trade recommendations. 1 means predictions of positive returns, so adopt long positions to capture that upside. 0 means negative returns expected, so stay neutral or short.

In essence, the classifier synthesizes current conditions to determine bullish or bearish leanings and transforms that into actionable, tailored trading guidance - enter longs or stay sidelines/short. This maximizes gains during rising markets while minimizing losses when indexes fall.

The machine learning pattern recognition of market dynamics precedes prudent positioning.

Multiple Regression and Random Forest Regression Model:

The multiple linear regression and random forest model is trained to forecast the opening, high, low, closing prices, expected percent price range, and projected return for the next trading day. The features fed into the model encompass the latest actual price bars along with technical indicators like volatility and intraday movement.

By predicting the high and low range bounds for tomorrow, the model provides guidance on an expected tradable price envelope. Specifically, the predicted low price forms the theoretical buy limit for trade entry to capture the upswing. By buying as close to the lows as possible, the strategy seeks to maximize exposure to the ensuing rally.

The model's projection of the probable closing price provides the trade exit target. By exiting near the predicted daily settlement, the model enables selling into strength at stretched, high prices near tops to lock in swings. Selling just before the market closes allows full exposure to upside while avoiding overnight gaps.

So in summary, the multiple regression and random forest model forecasts guide entries around bottoms and exits around peaks to systematically buy low and sell high - riding both the down and up waves in each session to compound gains. The model essentially predicts

support/demand and resistance/supply levels to time bounce points. Only by accurately forecasting the price trajectory can this low-risk range trading approach function consistently. The algorithmic future projections power the entire signal generation process that dictates getting in-and-out of positions.

Here the model accuracy directly impacts strategy win rate, risk management, and ultimate profitability. The price forecasts form the crux and allow back testing various execution rules around the machine-driven market timing.

Probability of Capturing Target Price Swing:

A key metric calculated in the strategy is the historical probability of capturing the desired target price swing from daily low to close. This winning rate estimate provides insight into the frequency and reliability of the intended trading outcomes over time.

The calculation methodology is as follows:

1) Specify target Delta - for example 1% intraday return

2) Scan entire historical dataset

3) Identify subset of days that meet target criteria:
    - Days with positive daily return
    - Intraday swing(delta) from low to close exceeds or equal to 1%

4) Count number of days meeting criteria as wins (W)

5) Count total number of days in dataset as total (T)

6) Divide number of winning days by total days

7) Probability Estimate = Number of Wins / Total Days = W / T

This gives the historical win rate for the specified target return threshold based on actual realized prices. By dividing the number of winning instances by the entire sample size, we derive the empirical probability estimate.

The higher the probability measure, the more frequently the target return has been achieved in the historical data. This gives confidence in the reliability of hitting daily goals. If the probability is too low, it implies the targets are rarely attained.

The probability can be simulated across various target return thresholds to map out historical frequencies. This methodology quantifies opportunity and forms reasonable win expectations.

PREDICTIONS:

Date: 2023-12-11

Signal: Long Position

DELTA: 1.5 %

Multiple Linear Regression Prediction:

Probability of winning with delta 1.5% is 18.38%.

```
Prediction of Multiple Linear Regression Model:

The probability of making 0.015 on 2023-12-11 00:00:00 is 18.38% with these predictions:

          Open        High         Low  Close  Daily_Return  Move_per
0  114.489258  115.614351  114.239998  115.0      0.007666  0.006658
```

Random Forest Regressor prediction:

```
Prediction of Random Forest Model:

The probability of making 0.015 on 2023-12-11 00:00:00 is 18.38% with these predictions:

        Open      High       Low       Close  Daily_Return  Move_per
0  114.5741  115.5539  113.9874  114.923999     -0.002728  0.006648
```

Actual Values:

Date	Open	High	Low	Close	Adj Close	Volume	Daily_Return	Move	Move_per
2023-12-11	114.949997	115.400002	114.440002	115.07	115.07	713800	0.000609	0.629997	0.005505

Date: 2023-12-12

Signal: Long Position

DELTA: 0.5 %

Multiple Linear Regression Prediction:

Probability of winning at delta 0.5 % is 44.18%

```
Prediction of Multiple Linear Regression Model:

The probability of making 0.005 on 2023-12-12 00:00:00 is 44.18% with these predictions:

          Open        High         Low   Close  Daily_Return  Move_per
0  114.920847  115.862023  114.440002  115.07      0.004535  0.004085
```

Random Forest Regressor prediction:

```
Prediction of Random Forest Model:

The probability of making 0.005 on 2023-12-12 00:00:00 is 44.18% with these predictions:

        Open        High       Low     Close  Daily_Return  Move_per
0  115.0597  115.721701  114.3259  114.9989     -0.001688  0.005542
```

Actual Values:

Date	Open	High	Low	Close	Adj Close	Volume	Daily_Return	Move	Move_per
2023-12-12	113.849998	114.029999	112.68	113.370003	113.370003	667700	-0.014774	0.690002	0.006124

Date: 2023-12-13

Signal: Short or Neutral Position

DELTA: 0.25 %

Multiple Linear Regression Prediction:

Probability of winning at delta 0.25 % is 48.55%

```
Prediction of Multiple Linear Regression Model:

The probability of making 0.0025 on 2023-12-13 00:00:00 is 48.55% with these predictions:

       Open        High     Low       Close  Daily_Return  Move_per
0  113.71472  114.647241  112.68  113.370003     -0.002469  0.004322
```

Random Forest Regressor Prediction:

```
Prediction of Random Forest Model:

The probability of making 0.0025 on 2023-12-13 00:00:00 is 48.55% with these predictions:

       Open      High         Low       Close  Daily_Return  Move_per
0  113.7201  114.2382  112.695599  113.121401     -0.001662  0.006161
```

Actual Values:

Date	Open	High	Low	Close	Adj Close	Volume	Daily_Return	Move	Move_per
2023-12-13	113.559998	115.040001	112.970001	114.940002	114.940002	717400	0.013848	1.970001	0.017438

Date: 2023-12-14

Signal: Long Position

DELTA: 0.75 %

Probability of winning at delta 0.75 % is 37.02%

Multiple Linear Regression Prediction:

```
Prediction of Multiple Linear Regression Model:

The probability of making 0.0075 on 2023-12-14 00:00:00 is 37.02% with these predictions:

          Open        High         Low       Close  Daily_Return  Move_per
0  113.520597  115.307045  112.970001  114.940002      0.013735  0.020176
```

Random Forest Regressor Prediction:

```
Prediction of Random Forest Model:

The probability of making 0.0075 on 2023-12-14 00:00:00 is 37.02% with these predictions:

          Open      High         Low       Close  Daily_Return  Move_per
0  113.447701  115.2805  112.862899  114.738699      0.015891   0.01771
```

Actual Values:

Date	Open	High	Low	Close	Adj Close	Volume	Daily_Return	Move	Move_per
2023-12-14	116.25	118.57	116.25	118.419998	118.419998	860700	0.030277	2.169998	0.018667

Date: 2023-12-15

Signal: Long Position

DELTA: 1.5 %

Probability of winning at delta 1.2 % is 18.4%

Multiple Linear Regression Prediction:

```
Prediction of Multiple Linear Regression Model:

The probability of making 0.015 on 2023-12-15 00:00:00 is 18.4% with these predictions:

          Open       High     Low       Close  Daily_Return  Move_per
0  116.549786  118.28599  116.25  118.419998      0.020337  0.022912
```

Random Forest Regressor Prediction:

```
Prediction of Random Forest Model:

The probability of making 0.015 on 2023-12-15 00:00:00 is 18.4% with these predictions:

          Open      High         Low       Close  Daily_Return  Move_per
0  116.330002  119.3584  116.017601  118.167699      0.017379   0.01868
```

Actual Values:

Date	Open	High	Low	Close	Adj Close	Volume	Daily_Return	Move	Move_per
2023-12-15	117.709999	118.110001	116.830002	117.709999	117.709999	611400	-0.005996	0.879997	0.007532

Date: 2023-12-18

Signal: Short or Neutral Position

DELTA: 0.9 %

Probability of winning at delta 0.9 % is 32.71%

Multiple Linear Regression Prediction:

```
Prediction of Multiple Linear Regression Model:

The probability of making 0.009000000000000001 on 2023-12-18 00:00:00 is 32.71% with these predictions:

          Open        High         Low       Close  Daily_Return  Move_per
0  117.595906  118.690364  116.830002  117.709999      0.002801  0.006237
```

Random Forest Regressor Prediction:

```
Prediction of Random Forest Model:

The probability of making 0.009000000000000001 on 2023-12-18 00:00:00 is 32.71% with these predictions:

        Open        High       Low       Close  Daily_Return  Move_per
0  117.4492  118.089501  116.9874  117.658099      0.001229  0.007616
```

Actual Values:

	Open	High	Low	Close	Adj Close	Volume	Daily_Return	Move	Move_per
Date									
2023-12-18	119.5	120.519997	118.699997	118.830002	118.830002	526600	0.009515	0.130005	0.001095

Date: 2023-12-19

Signal: Long Position

DELTA: 0.6 %

Probability of winning at delta 0.6 % is 41.09%

Multiple Linear Regression Prediction:

```
Prediction of Multiple Linear Regression Model:

The probability of making 0.006 on 2023-12-19 00:00:00 is 41.09% with these predictions:

          Open        High         Low       Close  Daily_Return  Move_per
0  119.713729  119.862198  118.699997  118.830002     -0.005057 -0.000717
```

Random Forest Regressor Prediction:

```
Prediction of Random Forest Model:

The probability of making 0.006 on 2023-12-19 00:00:00 is 41.09% with these predictions:

       Open        High         Low       Close  Daily_Return  Move_per
0  119.026  119.992801  118.595401  119.271101     -0.013328  0.001073
```

Actual Values:

Date	Open	High	Low	Close	Adj Close	Volume	Daily_Return	Move	Move_per
2023-12-19	118.010002	119.230003	117.699997	119.220001	119.220001	554200	0.003282	1.520004	0.012914

Conclusion:Trading Strategy

The core strategy is to purchase stocks at the lowest price of the trading day, and hold the position until the market closes to capture the intraday price appreciation.

Specifically, a buy limit order would be placed each morning at market open. The limit price for this order would be set at the predicted low price of the day, obtained from machine learning models. Once the actual stock price drops to this limit level during the day, the order would be executed at the lows. The trade would then be held until the closing bell to benefit from any price gains into the daily highs and close.

The intention is to leverage the machine learning predictions to accurately buy into the day's lows, facilitating the trade's upside exposure to the ensuing intraday rally up to the eventual market close. By pinpointing the lows through data modeling, the strategy aims to systematically buy at the bottom of the daily range and sell at the top - pocketing that full price swing from trough to peak.

As seen in the provided predictions, the machine learning models are forecasting the daily low prices with fairly accurate precision overall. The predicted values are relatively close to the actual lows that end up being reached during the trading day.

This indicates that the models and data pipelines driving the predictions are generally sound. When reviewing previous daily price charts, the target profit delta between the day's lows and close does seem achievable in many cases. So the core strategy itself has merit.

Here are the important values for the days which I tried to make a trade:

Date	Low	Close	Ml_pred_low	Rf_pred_low	Price_ml_trigger
2023-12-11	114.440002	115.070000	114.23	113.98	False
2023-12-12	112.680000	113.370003	144.44	114.32	True
2023-12-13	112.970001	114.940002	112.68	112.69	False
2023-12-14	116.250000	118.419998	112.97	112.86	False
2023-12-15	116.830002	117.709999	116.25	116.01	False
2023-12-18	118.699997	118.830002	116.83	116.98	False
2023-12-19	117.699997	119.220001	118.69	118.59	True

The "Low" and "Close" columns display the actual price values for that trading day - specifically, the lowest point reached and the final closing price. These are the real historical prices.

"Ml_pred_low" shows the low price predictions from the multiple linear regression model for each day. Similarly, "Rf_pred_low" contains the predictions made by the random forest regression model. These are the algorithmic forecasts for where the day's lows would land.

The "Price_ml_trigger" column has binary indicators derived from comparing the predicted lows versus the actual lows: A "True" means that a prediction was higher than truth, while a "False" signals it was lower. This essentially flags where the predictive models overestimated and would have caused the trading strategy to fail, since the buy limits would not have triggered with the predictions being above the real lows.

From the information provided, it is evident that the anticipated low value is lower than the actual low value. As a result, our orders will not be executed in real time and will fail to capture the price movement between the low and close of that day.

Our projected minimum price exceeded the actual price on two specific occasions. On the first occurrence, which occurred on "2023-12-12," we would have incurred a loss. However, on "2023-12-19," we could have realized a profit of 53 cents due to the observed price movement.

Future Applications:

• Improvement of the trade entry is necessary for the successful implementation of this model/strategy.

• Require a reassessment and development of an improved trading plan system.

• I require the inclusion of an emotive indicator in my foundational feature.