In [1]:
```
import yfinance as yf
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Lets import data

In [2]:
```
start='2012-01-01'
end='2023-12-30'
stock='GOOG'
data=yf.download(stock,start,end)
```

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

In [3]:
```
data.reset_index(inplace=True)
```

In [4]:
```
print(data)
```

```
           Date        Open        High         Low       Close   Adj Close  \
0    2012-01-03   16.262545   16.641375   16.248346   16.573130   16.554291
1    2012-01-04   16.563665   16.693678   16.453827   16.644611   16.625692
2    2012-01-05   16.491436   16.537264   16.344486   16.413727   16.395069
3    2012-01-06   16.417213   16.438385   16.184088   16.189817   16.171415
4    2012-01-09   16.102144   16.114599   15.472754   15.503389   15.485767
...         ...         ...         ...         ...         ...         ...
3013 2023-12-22  142.130005  143.250000  142.054993  142.720001  142.557770
3014 2023-12-26  142.979996  143.945007  142.500000  142.820007  142.657669
3015 2023-12-27  142.830002  143.320007  141.050995  141.440002  141.279236
3016 2023-12-28  141.850006  142.270004  140.828003  141.279999  141.119415
3017 2023-12-29  140.679993  141.434998  139.899994  140.929993  140.769806

         Volume
0     147611217
1     114989399
2     131808205
3     108119746
4     233776981
...         ...
3013   18494700
3014   11170100
3015   17288400
3016   12192500
3017   14872700

[3018 rows x 7 columns]
```

In [5]:
```
# Feature engineering – assuming 'Close' is the target variable and other columns are features & lets drop not
```

In [6]:
```
data.drop(columns=['Date', 'Adj Close'], inplace=True)
```

In [7]:
```
data
```

Out[7]:

	Open	High	Low	Close	Volume
0	16.262545	16.641375	16.248346	16.573130	147611217
1	16.563665	16.693678	16.453827	16.644611	114989399
2	16.491436	16.537264	16.344486	16.413727	131808205
3	16.417213	16.438385	16.184088	16.189817	108119746
4	16.102144	16.114599	15.472754	15.503389	233776981
...	...	...	...	...	...
3013	142.130005	143.250000	142.054993	142.720001	18494700
3014	142.979996	143.945007	142.500000	142.820007	11170100
3015	142.830002	143.320007	141.050995	141.440002	17288400
3016	141.850006	142.270004	140.828003	141.279999	12192500
3017	140.679993	141.434998	139.899994	140.929993	14872700

3018 rows × 5 columns

In [8]:
```
y = data['Close']
X = data.drop(columns=['Close'])
```

In [9]:
```
# Convert data to NumPy arrays for scaling
X_np = X.values
y_np = y.values
```

Data preprocessing and Model Development

Data preprocessing and Model Development

In [10]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_np)
```

Above code assure that The StandardScaler() from Scikit-learn is instantiated and applied to the features (X_np). This process ensures that each feature's values are scaled to a comparable range.

Training the predictive model:

In [11]:
```
#Random Forest Regression
```

In [12]:
```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
```

In [13]:
```
rf.fit(X_scaled, y_np)
```

Out[13]:
```
▾        RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In the snippet above, we instantiate a RandomForestRegressor() from Scikit-learn with 100 decision trees (n_estimators=100) and a fixed random state for reproducibility (random_state=42). We then train the model using our preprocessed and scaled features (X_scaled) along with the target variable (y_np).

Prediction

In [14]:
```
# New data point for prediction
new_data_point = pd.DataFrame({
    'Open': [145.00],
    'High': [146.00],
    'Low': [144.00],
    'Volume': [15000000]
})
```

In [15]:
```
# Scale the new data point
new_data_point_scaled = scaler.transform(new_data_point)
```

```
C:\Users\USER\anaconda3\Lib\site-packages\sklearn\base.py:457: UserWarning: X has feature names, but StandardSc
aler was fitted without feature names
  warnings.warn(
```

In [16]:
```
predicted_close_price = rf.predict(new_data_point_scaled)
```

In [17]:
```
print(predicted_close_price)
```

```
[144.95023361]
```

Now lets evaluate this first:

Model Evaluation

But before evaluation lets do same Stock Market prediction by making pipeline.

In [18]:
```
import yfinance as yf
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
```

In [19]:
```
# Define the stock and date range
start = '2012-01-01'
end = '2023-12-30'
stock = 'GOOG'

# Download stock data
data = yf.download(stock, start, end)
data.reset_index(inplace=True)
```

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

In [20]:
```
# Drop unnecessary columns
data.drop(columns=['Date', 'Adj Close'], inplace=True)
```

In [21]:
```
# Define the target and features
y = data['Close']
X = data.drop(columns=['Close'])
```

In [22]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [23]:
```python
# Create a pipeline with StandardScaler and RandomForestRegressor
pipeline = make_pipeline(StandardScaler(), RandomForestRegressor(n_estimators=100, random_state=42))
```

In [24]:
```python
# Train the pipeline on the training data
pipeline.fit(X_train, y_train)
```

Out[24]:
```
▸ Pipeline
  ▸ StandardScaler
  ▸ RandomForestRegressor
```

In [25]:
```python
# Make predictions on the testing set
y_pred = pipeline.predict(X_test)
```

In [26]:
```python
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [27]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [28]:
```python
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared Score: {r2}")
```

```
Mean Absolute Error (MAE): 0.3770729798196957
Mean Squared Error (MSE): 0.4621673510428435
R-squared Score: 0.9996887360346782
```

Interpretation:

Low MAE and MSE: These low values indicate that the model's predictions are very close to the actual stock prices. Small errors imply high accuracy.

High R^2 Score: This score suggests that the model can explain almost all the variance in the stock prices based on the features (Open, High, Low, Volume). This implies that the model is very effective at capturing the relationships between these features and the stock price.

Mean Absolute Error (MAE): 0.377

The MAE measures the average absolute difference between the predicted values and the actual values. An MAE of 0.377 means that, on average, the predictions are off by about 0.377 units. For stock prices, this is a very low error, suggesting high accuracy. Mean Squared Error (MSE): 0.462

The MSE measures the average squared difference between the predicted values and the actual values. It gives more weight to larger errors. An MSE of 0.462 is also very low, indicating that large errors are rare and the model's predictions are generally very close to the actual values. R-squared Score (R^2): 0.9997

The R-squared score indicates how well the independent variables explain the variance in the dependent variable. An R^2 score of 0.9997 is almost perfect (1.0 would be perfect).

In [29]:
```python
# Now if want to train the entire dataset
```

In [30]:
```python
# Train the pipeline on the entire dataset
pipeline.fit(X, y)

# New data point for prediction
new_data_point = pd.DataFrame({
    'Open': [145.00],
    'High': [146.00],
    'Low': [144.00],
    'Volume': [15000000]
})

# Predict using the pipeline
predicted_close_price = pipeline.predict(new_data_point)

print(predicted_close_price[0])
```

```
144.95023361206054
```