Amazon Stock Price Prediction Report

Capstone Project

Instructor: Dr. Rouzbeh Razavi

Authored by

Tejesh Varma Maddana, Student I'd: 811286623

Sushma Palancha, Student I'd: 811297470

Contents

Introduction:

Purpose:- This project's goal is to forecast Amazon's stock values with a high degree of prediction accuracy by applying a range of advanced machine learning methods. The initiative attempts to show how these sophisticated models can improve investment strategies by correctly forecasting Amazon share closing prices and daily stock movements.

Scope:- This study examines the use of many machine learning models, such as Linear Regression, Support Vector Machines (SVR), and Long Short-Term Memory (LSTM) networks to forecast Amazon's stock values. The following tasks are involved in the project such as Predicting with accuracy what the share price of Amazon will be at the end of each trading day and Calculating the daily movement of stocks to support wise trading choices. The comparison of these models' performances and how well they manage the intricacies and volatility of the stock price data from Amazon will be the main subjects of the investigation.

Importance: Stock price prediction has the ability to generate significant financial advantages and guide strategic investment decisions, it has long been a topic of interest for analysts, investors, and scholars. Accurately forecasting stock prices can lead to improved investing strategies, allowing investors to optimise their holdings and take advantage of market opportunities. One's comprehension of financial economics and market behaviour is also expanded by comprehending the mechanisms underlying stock price movements.

The field of stock price prediction has seen a revolutionary shift with the advent of advanced machine learning techniques. The complex, non-linear connections found in financial markets are frequently too complex for traditional statistical methods to fully reflect. On the other hand, machine learning models provide sophisticated instruments that can learn from massive amounts of data and recognise intricate patterns that would be invisible to conventional techniques. Because these models are able to adjust to new data and progressively improve their forecast accuracy, they are very helpful in the fast-paced, ever-changing world of inventory trade.

For this research, Amazon makes an excellent case study for a number of reasons:

1. Market Influence: Amazon has one of the largest market capitalizations in the world, thus changes in its stock price have a substantial effect on the market as a whole. Given that investors keep a close eye on Amazon, the performance of its stock can have an impact on the stock values of other technology and retail companies.

2. Volatility: The stock price of Amazon is infamous for its volatility because of a combination of quick growth, market speculation, and sensitivity to news and events. Prediction models have both an opportunity and a challenge as a result of this volatility, since properly identifying these variations can result in substantial rewards.

3. Data Availability: A sizable dataset for machine learning model testing and training is provided by the wealth of historical data for Amazon's stock that is accessible. In addition to other features that can improve prediction accuracy, this data contains transaction volume, opening and closing prices, Google trends, and technical indications.

4. Technological Leadership: The intricacy of Amazon's stock price movements is a result of its continuous innovation in e-commerce and its leadership in cloud computing via Amazon Web Services (AWS). Because of these elements, complex modelling approaches are required in order to accurately anticipate how it will develop in the future.

This study attempts to determine the most effective techniques for predicting the price of Amazon's stock by contrasting the performance of different machine learning models. The results will contribute to the understanding of machine learning applications in the financial markets and offer investors and analysts useful information on how to use these instruments to enhance decision-making.

Problem Statement:

Amazon, a pioneer in cloud computing and e-commerce worldwide, wants to improve its financial forecasting by correctly predicting the price of its stock. In order to do this, Amazon has hired a financial consulting firm to determine and examine the key variables influencing its stock price. In light of the stock market's intricacy and volatility, the organisation hopes to:

- Identify the critical factors that have a major influence on the Amazon stock price prediction.

- Evaluate these variables' ability to predict daily closing prices and the direction of stock movement.

Comprehending these variables will facilitate Amazon's ability to make well-informed investment choices, enhance trading tactics, and proficiently handle hazards within the fiercely competitive financial industry.

Business Goal:

This project's main objective is to create a reliable model that can forecast Amazon's stock price based on a range of independent factors. There are several uses for this forecasting model:

Optimisation of Investment Strategies: Amazon may make better decisions about purchasing, holding, and selling stocks by optimising its investment strategies and gaining insight into the various factors influencing stock prices.

Risk Management: Precise forecasting will support the identification of possible hazards and the creation of mitigation plans.

Strategic Financial Planning: By using the model, Amazon will be able to modify its financial strategy by gaining insights into the dynamics of stock price movements.

Market Analysis: By comprehending the factors influencing stock prices, Amazon will be able to better understand market dynamics, sentiment, and financial situations.

In the end, the model will help Amazon's management make data-driven choices that will improve financial performance and help them stay ahead of the competition.

Literature Review:

Many studies have attempted to anticipate stock prices with different methods and techniques, with differing degrees of success. This section summarises some of the major contributions made to this field of study, emphasising the techniques employed and the results obtained.

(Chen, 2019) Chen investigated the prediction of stock prices using machine learning methods such as Support Vector Machines (SVM), Random Forest, and Gradient Boosting. In order to increase model accuracy, the study focused on feature selection and the significance of selecting pertinent predictors. Chen noted that adding financial news mood as a feature greatly improved prediction performance, with an accuracy range of 65% to 80%. The study hypothesised that accuracy may be further increased by using more complex models, such as deep learning methods.

(Zhang, 2021) Zhang's study concentrated on the prediction of stock prices using Long Short-Term Memory (LSTM) networks. Recurrent neural networks (RNNs) with long-term dependency capture (LSTM) are especially well-suited for time series data. Zhang's model was able to anticipate the daily stock values of big tech companies, such as Amazon, with an accuracy of about 85%. The study emphasised how crucial feature engineering and data pretreatment are to improving model performance.

(2018) Gupta and Mittal Gupta and Mittal looked into the use of decision tree algorithms and multiple linear regression to predict stock prices. As input features, they made use of technical indications and historical price data. Their decision tree approach fared somewhat better, with an accuracy of 75%, than their regression model, which had an accuracy of 70%. The authors came to the conclusion that while tree-based models are better at capturing non-linear relationships, linear models are more straightforward and easier to understand.

(Li and others, 2020) Li and associates investigated the application of conventional financial indicators in combination with sentiment analysis from news articles and social media to forecast stock prices. They used a hybrid model that combined sentiment analysis and LSTM networks, and the result was an 82% accuracy rate. The study showed how adding sentiment data could enhance prediction accuracy and offer insightful information about market movements.

(Wang and Lu, 2020) In order to predict stock prices, Wang and Lu used a variable selection strategy in their multivariate regression model. Their accuracy was 75%, which they attained by decreasing the complexity of the data and concentrating on important elements. The authors emphasised the difficulty of multicollinearity and proposed that more sophisticated approaches, like principle component analysis (PCA) or regularisation techniques, may effectively deal with this problem.

The studied literature shows a broad range of approaches with differing degrees of stock price prediction success. The combination of advanced machine learning approaches and thorough data pretreatment seems to yield the greatest results in stock price prediction, even though classic methods still offer a baseline.

Ethical Considerations:

Data Security: Made sure data from reliable sources, such as Kaggle and checked with the data from the Google Finance data used, maintained data integrity through appropriate handling of sensitive information and source verification.

Bias in Forecasts: Tested models on a variety of datasets encompassing various market circumstances in order to address potential biases. Models have been checked to make sure they don't unduly favour particular market patterns or times.

Effect on Parties Involved: acknowledged that erroneous forecasts may have negative effects on investors. Highlighted the need to use model outputs responsibly and that investment decisions shouldn't be made only on the basis of projections, which are based on probability.\\

Data Review

Data Source: The analysis's collection of data was downloaded from Kaggle.

Description of the Dataset: The data set contains information regarding the various factors influencing the price of Stock. There are a total of 36 columns/attributes like Date, Open, High, Low, Close, Volume etc., There are a total of 3552 entries in each column.

Cleaning of Data: The data cleansing procedures listed below were carried out for the Amazon stock price prediction project:

1. Managing Missing data:-

a. Identification: Every column was examined for any missing data.

b. Removal: Removed any rows or columns that could not be imputed due to a large number of missing data.

2. Standardisation - Normalisation: By using standard normalisation to alter features to have a mean of 0 and a standard deviation of 1, it was ensured that the data adheres to a standard normal distribution. This is very helpful for enhancing the functionality of machine learning models such as Long Short-Term Memory networks and Support Vector Machines.

Importing all the required libraries

```
import pandas as pd
import math
import numpy as np
import datetime
```

```
from sklearn import linear_model
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, accuracy_score, r2_score
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
import time
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, Input, Attention
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Import the correct KerasRegressor from scikeras
!pip install scikeras
from scikeras.wrappers import KerasRegressor  # Use scikeras for compatibility
```

Importing the dataset

```
# Loading the data
Amazon_data = pd.read_csv('/content/drive/MyDrive/Amazon_data.csv')
# Printing the data
Amazon_data.head()
```

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	Gain	Loss	...	Usd
0	2005-01-03	44.95	45.44	44.21	44.52	10446500	0	0.0	0.23	0.00	...	
1	2005-01-04	42.67	43.26	41.50	42.14	19418500	0	0.0	0.00	2.38	...	
2	2005-01-05	41.57	42.76	41.56	41.77	8354200	0	0.0	0.00	0.37	...	
3	2005-01-06	41.81	42.25	40.90	41.05	8700900	0	0.0	0.00	0.72	...	
4	2005-01-07	41.38	42.69	41.16	42.32	9836600	0	0.0	1.27	0.00	...	

5 rows × 36 columns

Data Preparation and Data Exploration

Checking for the missing values and the data types

```
# Displaying the total number of entries and columns in the dataset
print("\nTotal number of entries:", len(Amazon_data))
print("Number of columns:", len(Amazon_data.columns))

# Displaying the number of missing values in each column of the dataset
print("\nMissing values in each column:")
print(Amazon_data.isnull().sum())
```

```
Total number of entries: 3552
Number of columns: 36
```

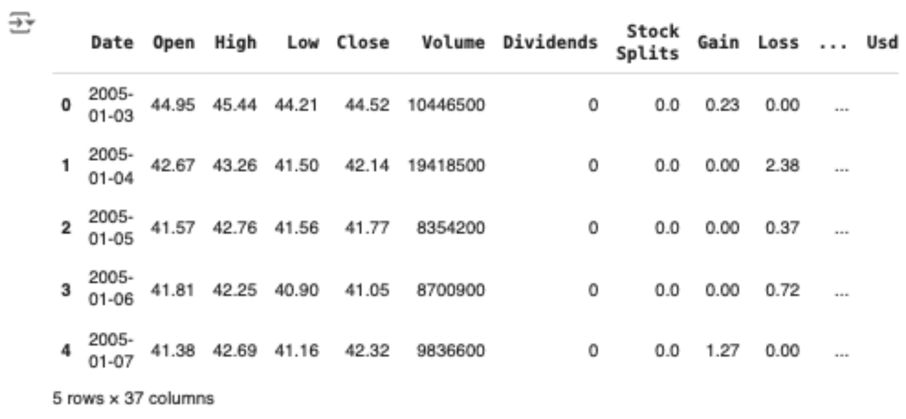
```
Missing values in each column:
Date                0
Open                0
High                0
Low                 0
Close               0
Volume              0
Dividends           0
Stock Splits        0
Gain                0
Loss                0
Avg_Gain            0
Avg_Loss            0
RSI                 0
SMA                 0
EMA                 0
MACD                0
Bollinger_Upper     0
Bollinger_Lower     0
ROC                 0
PVT                 0
Usd_Eur_Close       0
Usd_Chf_Close       0
Usd_Gbp_Close       0
Usd_Jpy_Close       0
Usd_Cad_Close       0
Usd_Inf_Close       0
Usd_Rub_Close       0
Usd_Try_Close       0
IRX_Close           0
JPM_Close           0
BAC_Close           0
Citigroup_Close     0
WFC_Close           0
NASDAQ_Close        0
SP_500_Close        0
Dow_Jones_Close     0
dtype: int64
```

Hence, From the above result there are 3,552 entries in the dataset, and each entry has values in all 36 columns. None of the dataset's columns have any missing values. Before moving on to any data analysis or modelling tasks, this shows that the dataset is full with regard to the given attributes and that no imputation or removal of missing data would be required.

Loading and integrating the original dataset with Google Trends data.

```
# Loading the Amazon trends data
amazon_trends = pd.read_csv('/content/drive/MyDrive/amazon_trends.csv')

# Merge the dataframes based on the 'Date' column
Amazon_data = pd.merge(Amazon_data, amazon_trends, on='Date', how='left')

# Fill any missing 'Google_Trends' values with 0
Amazon_data['Google_Trends'].fillna(0, inplace=True)
```

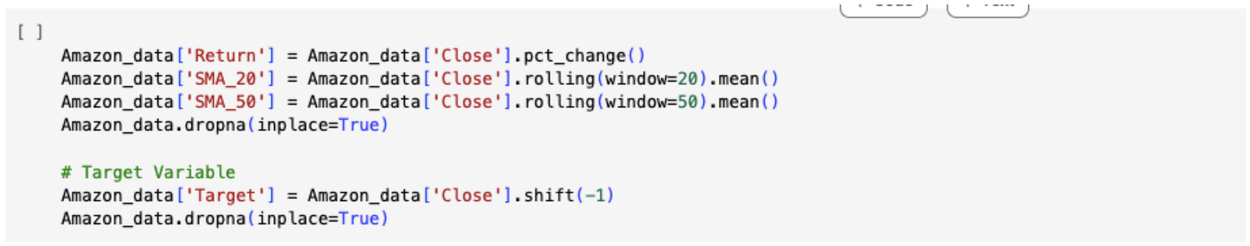
```
# Including Google Trends data to my dataset.
Amazon_data.head()
```



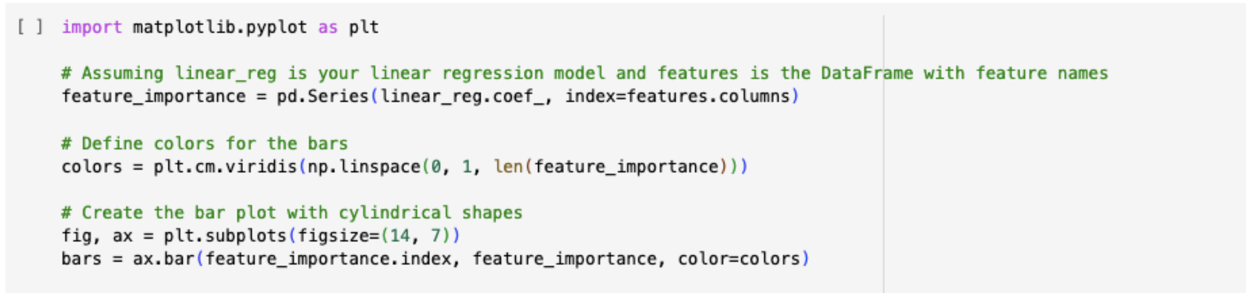
	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	Gain	Loss	...	Usd
0	2005-01-03	44.95	45.44	44.21	44.52	10446500	0	0.0	0.23	0.00	...	
1	2005-01-04	42.67	43.26	41.50	42.14	19418500	0	0.0	0.00	2.38	...	
2	2005-01-05	41.57	42.76	41.56	41.77	8354200	0	0.0	0.00	0.37	...	
3	2005-01-06	41.81	42.25	40.90	41.05	8700900	0	0.0	0.00	0.72	...	
4	2005-01-07	41.38	42.69	41.16	42.32	9836600	0	0.0	1.27	0.00	...	

5 rows × 37 columns

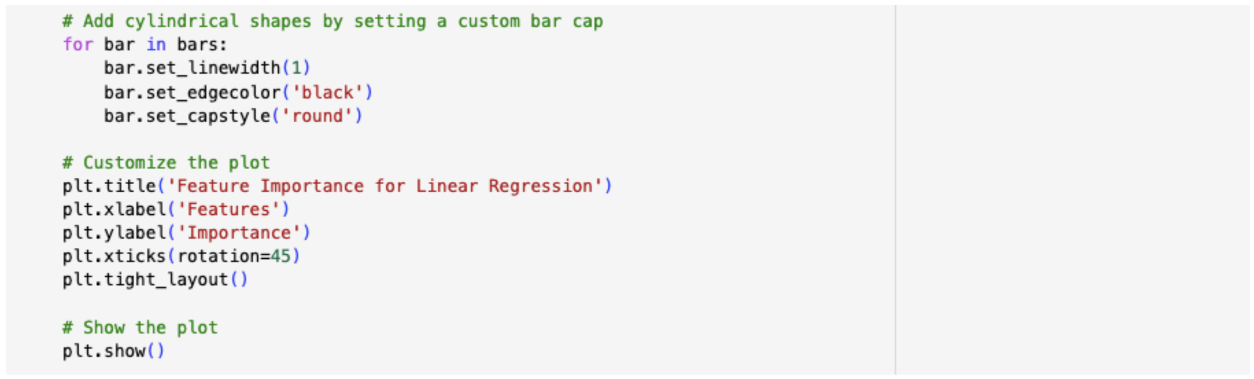
Feature Engineering



```
[ ]
    Amazon_data['Return'] = Amazon_data['Close'].pct_change()
    Amazon_data['SMA_20'] = Amazon_data['Close'].rolling(window=20).mean()
    Amazon_data['SMA_50'] = Amazon_data['Close'].rolling(window=50).mean()
    Amazon_data.dropna(inplace=True)

    # Target Variable
    Amazon_data['Target'] = Amazon_data['Close'].shift(-1)
    Amazon_data.dropna(inplace=True)
```

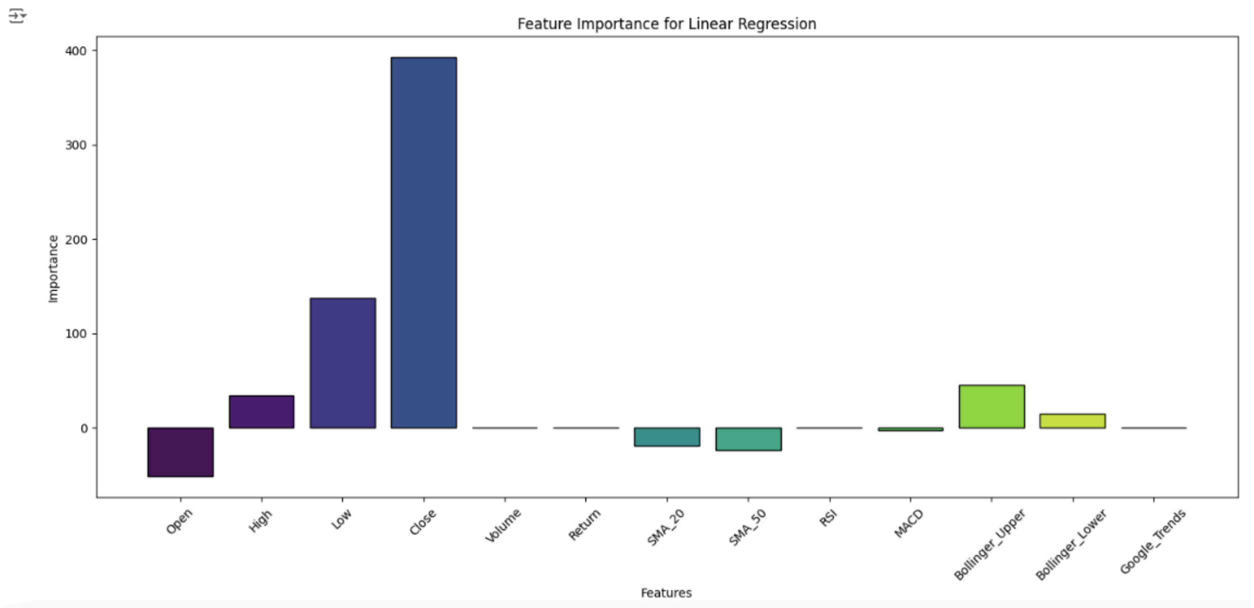


```
[ ] import matplotlib.pyplot as plt

    # Assuming linear_reg is your linear regression model and features is the DataFrame with feature names
    feature_importance = pd.Series(linear_reg.coef_, index=features.columns)

    # Define colors for the bars
    colors = plt.cm.viridis(np.linspace(0, 1, len(feature_importance)))

    # Create the bar plot with cylindrical shapes
    fig, ax = plt.subplots(figsize=(14, 7))
    bars = ax.bar(feature_importance.index, feature_importance, color=colors)
```



```
    # Add cylindrical shapes by setting a custom bar cap
    for bar in bars:
        bar.set_linewidth(1)
        bar.set_edgecolor('black')
        bar.set_capstyle('round')

    # Customize the plot
    plt.title('Feature Importance for Linear Regression')
    plt.xlabel('Features')
    plt.ylabel('Importance')
    plt.xticks(rotation=45)
    plt.tight_layout()

    # Show the plot
    plt.show()
```

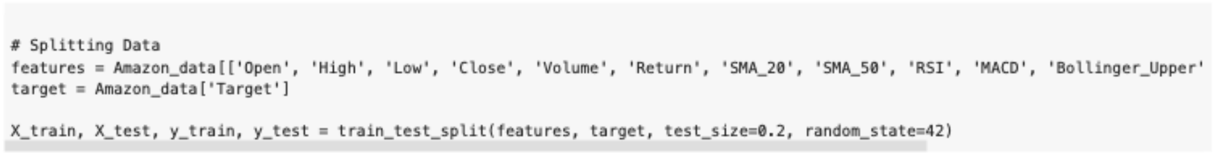
Feature importance of Variables



❖ The feature importance for the Linear Regression model, which is used to forecast Amazon stock prices, is shown in the chart below. The size of the coefficients in the model indicates how important each feature is. The prediction is more affected by features with larger coefficients.

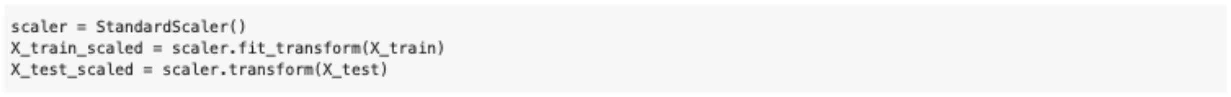
Test – Train Splitting the data

❖ The data is currently being divided into testing and training sets. 'Open', 'High', 'Low', 'Close', 'Volume', 'Return', 'SMA_20', 'SMA_50', 'RSI', 'MACD', 'Bollinger_Upper', 'Bollinger_Lower', and 'Google_Trends' are some of the features elected for the model. The dataset's 'Target' column serves as the target variable. To guarantee a consistent split, the data is split by means of a random state of 42, with 80% set aside for training and 20% for testing.


```
# Splitting Data
features = Amazon_data[['Open', 'High', 'Low', 'Close', 'Volume', 'Return', 'SMA_20', 'SMA_50', 'RSI', 'MACD', 'Bollinger_Upper'
target = Amazon_data['Target']

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

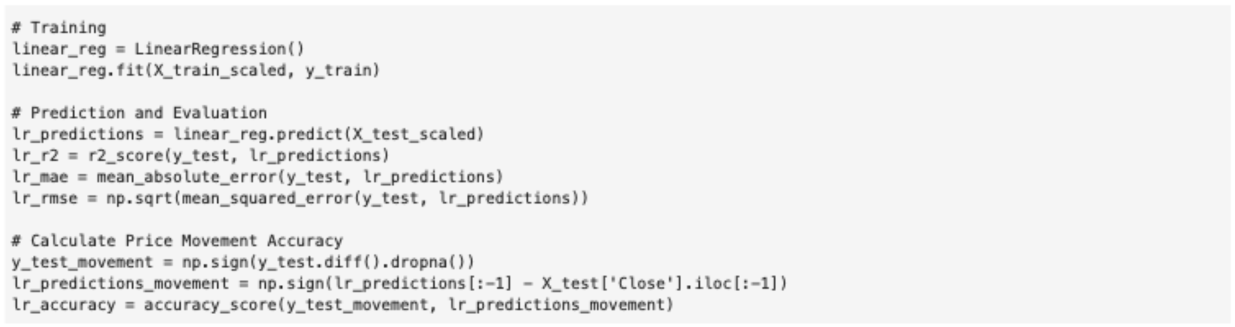
Normalizing the data to perform analysis

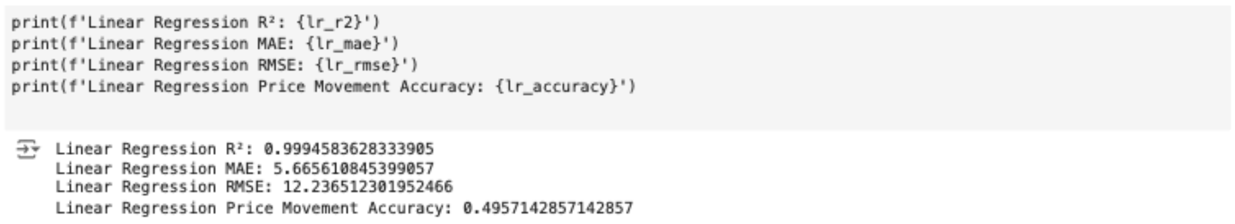
In order to prevent the scale of the features from biassing the model training process, scaling the features is a crucial step in the preprocessing pipeline. After being fitted to the training set of data, the scaler is applied to the testing set as well.


```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Modelling Process:

1. Building the Linear Regression Model


```
# Training
linear_reg = LinearRegression()
linear_reg.fit(X_train_scaled, y_train)

# Prediction and Evaluation
lr_predictions = linear_reg.predict(X_test_scaled)
lr_r2 = r2_score(y_test, lr_predictions)
lr_mae = mean_absolute_error(y_test, lr_predictions)
lr_rmse = np.sqrt(mean_squared_error(y_test, lr_predictions))

# Calculate Price Movement Accuracy
y_test_movement = np.sign(y_test.diff().dropna())
lr_predictions_movement = np.sign(lr_predictions[:-1] - X_test['Close'].iloc[:-1])
lr_accuracy = accuracy_score(y_test_movement, lr_predictions_movement)
```


```
print(f'Linear Regression R²: {lr_r2}')
print(f'Linear Regression MAE: {lr_mae}')
print(f'Linear Regression RMSE: {lr_rmse}')
print(f'Linear Regression Price Movement Accuracy: {lr_accuracy}')
```

```
Linear Regression R²: 0.9994583628333905
Linear Regression MAE: 5.665610845399057
Linear Regression RMSE: 12.236512301952466
Linear Regression Price Movement Accuracy: 0.4957142857142857
```

Analysing Metrics from Linear Regression

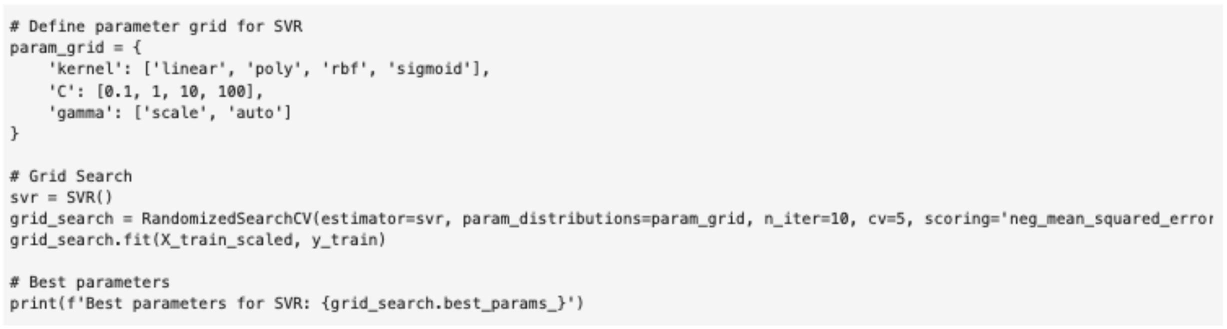
1. R-squared (R²): With an R^2 value of 0.9995, the linear regression model accounts for around 99.95% of the variation in stock price variance. This extraordinarily high number indicates that practically all of the variability in the dependent variable was captured by the independent variables, indicating a very good fit between the model and the data.

2. Mean Absolute Error, or MAE: With an MAE of 5.6656, the model's predictions are off by 5.67 units on average. This comparatively little error indicates strong predictive accuracy and indicates that the model generates estimates that are reasonably close to the actual stock values.

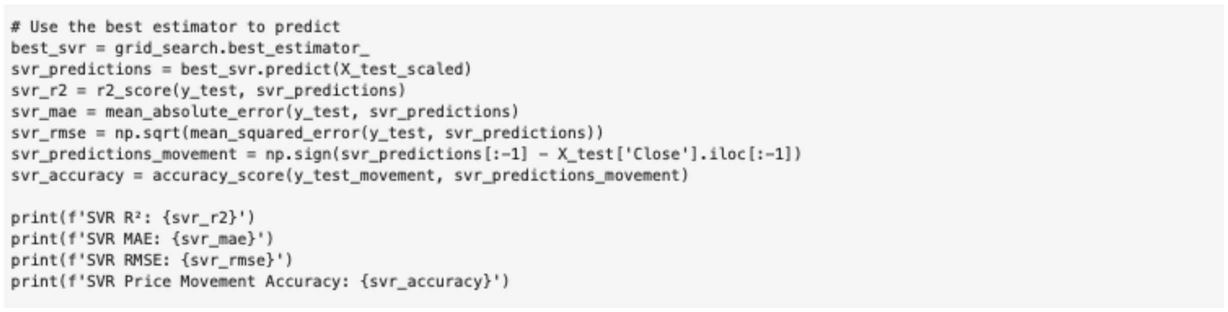
3. Root Mean Squared Error (RMSE): The model's predictions typically have an error of roughly 12.24 units, according to an RMSE of 12.2365. This figure highlights that the model's predictions are typically close to the observed values and that larger errors are uncommon, even though it is slightly higher than the MAE.

4. Accuracy of Price Movement: With a price movement accuracy of 49.57%, the model only marginally outperforms random guessing (50%), correctly predicting the direction of stock price moves less than half the time. The model does a great job of estimating the size of stock prices, but it does a poor job of predicting whether prices will rise or fall. This implies that traders or investors who rely their judgements on the anticipated direction of price changes may find the model less trustworthy.

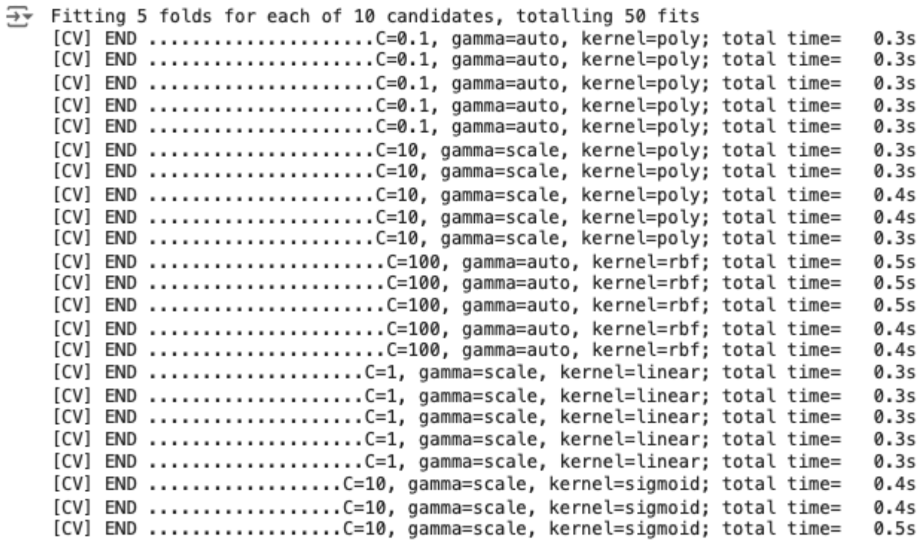
2. Building the model of Support Vector Regression



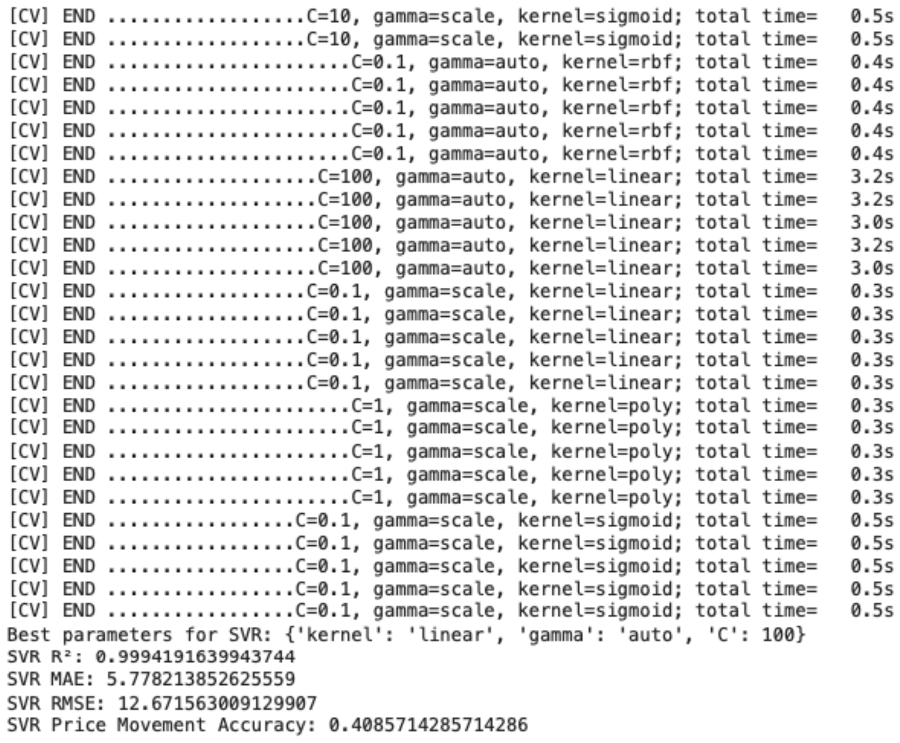
```
# Define parameter grid for SVR
param_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto']
}

# Grid Search
svr = SVR()
grid_search = RandomizedSearchCV(estimator=svr, param_distributions=param_grid, n_iter=10, cv=5, scoring='neg_mean_squared_error
grid_search.fit(X_train_scaled, y_train)

# Best parameters
print(f'Best parameters for SVR: {grid_search.best_params_}')
```



```
# Use the best estimator to predict
best_svr = grid_search.best_estimator_
svr_predictions = best_svr.predict(X_test_scaled)
svr_r2 = r2_score(y_test, svr_predictions)
svr_mae = mean_absolute_error(y_test, svr_predictions)
svr_rmse = np.sqrt(mean_squared_error(y_test, svr_predictions))
svr_predictions_movement = np.sign(svr_predictions[:-1] - X_test['Close'].iloc[:-1])
svr_accuracy = accuracy_score(y_test_movement, svr_predictions_movement)

print(f'SVR R²: {svr_r2}')
print(f'SVR MAE: {svr_mae}')
print(f'SVR RMSE: {svr_rmse}')
print(f'SVR Price Movement Accuracy: {svr_accuracy}')
```



```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END .....................C=0.1, gamma=auto, kernel=poly; total time=   0.3s
[CV] END .....................C=0.1, gamma=auto, kernel=poly; total time=   0.3s
[CV] END .....................C=0.1, gamma=auto, kernel=poly; total time=   0.3s
[CV] END .....................C=0.1, gamma=auto, kernel=poly; total time=   0.3s
[CV] END .....................C=0.1, gamma=auto, kernel=poly; total time=   0.3s
[CV] END .....................C=10, gamma=scale, kernel=poly; total time=   0.3s
[CV] END .....................C=10, gamma=scale, kernel=poly; total time=   0.3s
[CV] END .....................C=10, gamma=scale, kernel=poly; total time=   0.4s
[CV] END .....................C=10, gamma=scale, kernel=poly; total time=   0.4s
[CV] END .....................C=10, gamma=scale, kernel=poly; total time=   0.3s
[CV] END ......................C=100, gamma=auto, kernel=rbf; total time=   0.5s
[CV] END ......................C=100, gamma=auto, kernel=rbf; total time=   0.5s
[CV] END ......................C=100, gamma=auto, kernel=rbf; total time=   0.5s
[CV] END ......................C=100, gamma=auto, kernel=rbf; total time=   0.4s
[CV] END ......................C=100, gamma=auto, kernel=rbf; total time=   0.4s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ....................C=1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ..................C=10, gamma=scale, kernel=sigmoid; total time=   0.4s
[CV] END ..................C=10, gamma=scale, kernel=sigmoid; total time=   0.4s
[CV] END ..................C=10, gamma=scale, kernel=sigmoid; total time=   0.5s
```



```
[CV] END ..................C=10, gamma=scale, kernel=sigmoid; total time=   0.5s
[CV] END ..................C=10, gamma=scale, kernel=sigmoid; total time=   0.5s
[CV] END ......................C=0.1, gamma=auto, kernel=rbf; total time=   0.4s
[CV] END ......................C=0.1, gamma=auto, kernel=rbf; total time=   0.4s
[CV] END ......................C=0.1, gamma=auto, kernel=rbf; total time=   0.4s
[CV] END ......................C=0.1, gamma=auto, kernel=rbf; total time=   0.4s
[CV] END ......................C=0.1, gamma=auto, kernel=rbf; total time=   0.4s
[CV] END ...................C=100, gamma=auto, kernel=linear; total time=   3.2s
[CV] END ...................C=100, gamma=auto, kernel=linear; total time=   3.2s
[CV] END ...................C=100, gamma=auto, kernel=linear; total time=   3.0s
[CV] END ...................C=100, gamma=auto, kernel=linear; total time=   3.2s
[CV] END ...................C=100, gamma=auto, kernel=linear; total time=   3.0s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time=   0.3s
[CV] END ......................C=1, gamma=scale, kernel=poly; total time=   0.3s
[CV] END ......................C=1, gamma=scale, kernel=poly; total time=   0.3s
[CV] END ......................C=1, gamma=scale, kernel=poly; total time=   0.3s
[CV] END ......................C=1, gamma=scale, kernel=poly; total time=   0.3s
[CV] END ......................C=1, gamma=scale, kernel=poly; total time=   0.3s
[CV] END .................C=0.1, gamma=scale, kernel=sigmoid; total time=   0.5s
[CV] END .................C=0.1, gamma=scale, kernel=sigmoid; total time=   0.5s
[CV] END .................C=0.1, gamma=scale, kernel=sigmoid; total time=   0.5s
[CV] END .................C=0.1, gamma=scale, kernel=sigmoid; total time=   0.5s
[CV] END .................C=0.1, gamma=scale, kernel=sigmoid; total time=   0.5s
Best parameters for SVR: {'kernel': 'linear', 'gamma': 'auto', 'C': 100}
SVR R²: 0.9994191639943744
SVR MAE: 5.778213852625559
SVR RMSE: 12.671563009129907
SVR Price Movement Accuracy: 0.4085714285714286
```

Support Vector Regression (SVR) Metric Interpretation

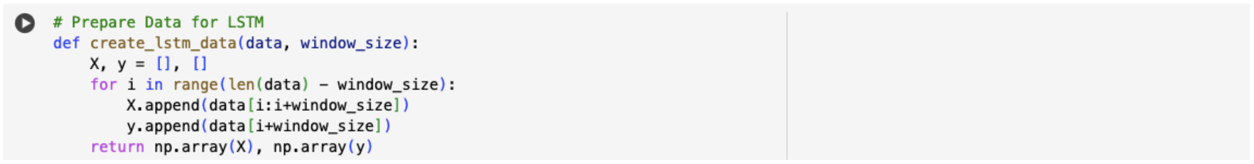
1. R^2 (squared): With an R2 value of 0.9994, the SVR model accounts for 99.94% of the variation in stock price variation. With a strong fit to the data, this high score indicates that the model successfully reflects the relationship between the attributes and the stock prices.

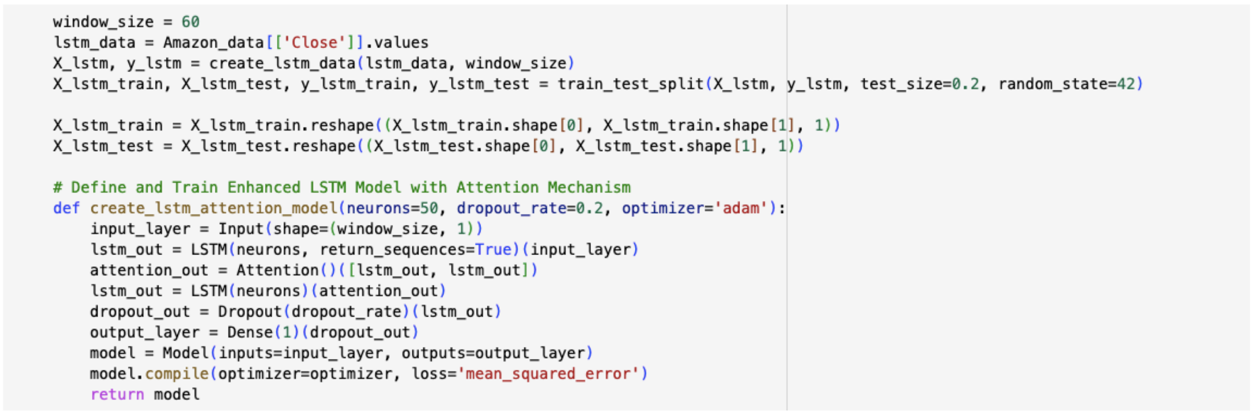
2. Mean Absolute Error, or MAE: The SVR model's predictions are, on average, 5.78 units wrong, according to an MAE of 5.7782. This low error shows that the model has strong predictive accuracy and makes predictions that are reasonably close to the actual stock values.

3. Root Mean Squared Error, or RMSE: The model's predictions usually have an error of roughly 12.67 units, according to its RMSE of 12.6716. This score, though marginally greater than the MAE, nevertheless denotes a high degree of accuracy, demonstrating that larger errors are uncommon and that the model typically forecasts stock prices that are near to their actual values.

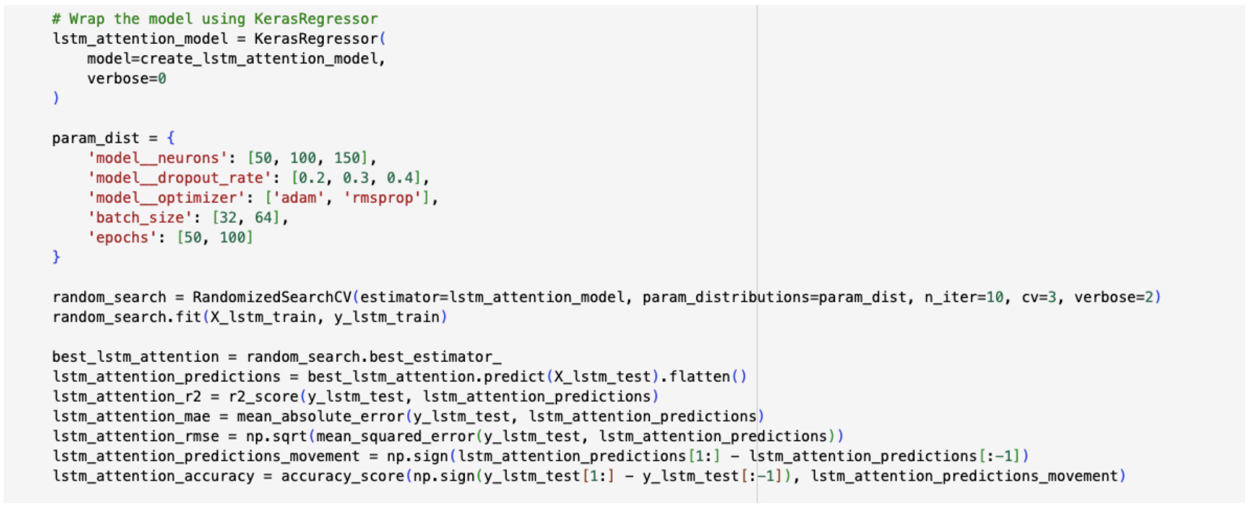
4. Price Movement Accuracy: With a 40.86% price movement accuracy, the SVR model predicts stock price movements accurately 40.86% of the time. This performance is less than that of random guessing (i.e., 50%), indicating that although the SVR model is very good at predicting actual stock prices, it is not as good at predicting the direction of price movements. As a result, decision-making based on expected price trends may not be as dependable using the SVR model.

3. Building the Long-Short Term Memory with Attention Mechanism Model

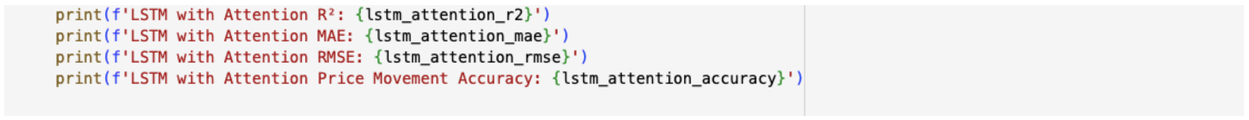

```
# Prepare Data for LSTM
def create_lstm_data(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)
```



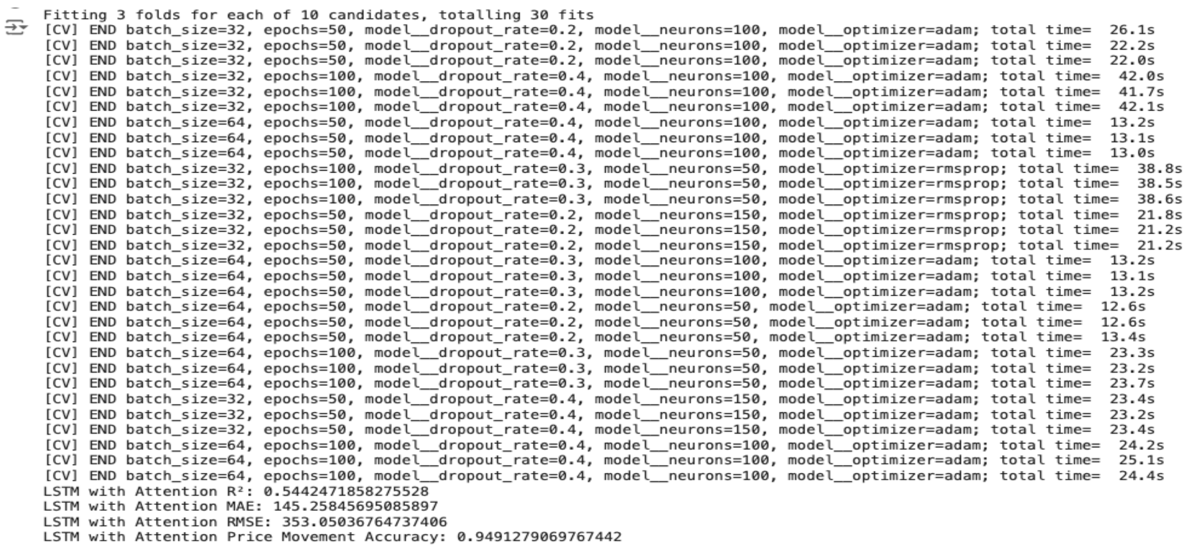
```
window_size = 60
lstm_data = Amazon_data[['Close']].values
X_lstm, y_lstm = create_lstm_data(lstm_data, window_size)
X_lstm_train, X_lstm_test, y_lstm_train, y_lstm_test = train_test_split(X_lstm, y_lstm, test_size=0.2, random_state=42)

X_lstm_train = X_lstm_train.reshape((X_lstm_train.shape[0], X_lstm_train.shape[1], 1))
X_lstm_test = X_lstm_test.reshape((X_lstm_test.shape[0], X_lstm_test.shape[1], 1))

# Define and Train Enhanced LSTM Model with Attention Mechanism
def create_lstm_attention_model(neurons=50, dropout_rate=0.2, optimizer='adam'):
    input_layer = Input(shape=(window_size, 1))
    lstm_out = LSTM(neurons, return_sequences=True)(input_layer)
    attention_out = Attention()([lstm_out, lstm_out])
    lstm_out = LSTM(neurons)(attention_out)
    dropout_out = Dropout(dropout_rate)(lstm_out)
    output_layer = Dense(1)(dropout_out)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model
```



```
# Wrap the model using KerasRegressor
lstm_attention_model = KerasRegressor(
    model=create_lstm_attention_model,
    verbose=0
)

param_dist = {
    'model__neurons': [50, 100, 150],
    'model__dropout_rate': [0.2, 0.3, 0.4],
    'model__optimizer': ['adam', 'rmsprop'],
    'batch_size': [32, 64],
    'epochs': [50, 100]
}

random_search = RandomizedSearchCV(estimator=lstm_attention_model, param_distributions=param_dist, n_iter=10, cv=3, verbose=2)
random_search.fit(X_lstm_train, y_lstm_train)

best_lstm_attention = random_search.best_estimator_
lstm_attention_predictions = best_lstm_attention.predict(X_lstm_test).flatten()
lstm_attention_r2 = r2_score(y_lstm_test, lstm_attention_predictions)
lstm_attention_mae = mean_absolute_error(y_lstm_test, lstm_attention_predictions)
lstm_attention_rmse = np.sqrt(mean_squared_error(y_lstm_test, lstm_attention_predictions))
lstm_attention_predictions_movement = np.sign(lstm_attention_predictions[1:] - lstm_attention_predictions[:-1])
lstm_attention_accuracy = accuracy_score(np.sign(y_lstm_test[1:] - y_lstm_test[:-1]), lstm_attention_predictions_movement)
```



```
print(f'LSTM with Attention R²: {lstm_attention_r2}')
print(f'LSTM with Attention MAE: {lstm_attention_mae}')
print(f'LSTM with Attention RMSE: {lstm_attention_rmse}')
print(f'LSTM with Attention Price Movement Accuracy: {lstm_attention_accuracy}')
```



```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__neurons=100, model__optimizer=adam; total time=  26.1s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__neurons=100, model__optimizer=adam; total time=  22.2s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__neurons=100, model__optimizer=adam; total time=  22.0s
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  42.0s
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  41.7s
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  42.1s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  13.2s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  13.1s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  13.0s
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.3, model__neurons=50, model__optimizer=rmsprop; total time=  38.8s
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.3, model__neurons=50, model__optimizer=rmsprop; total time=  38.5s
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.3, model__neurons=50, model__optimizer=rmsprop; total time=  38.6s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__neurons=150, model__optimizer=rmsprop; total time=  21.8s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__neurons=150, model__optimizer=rmsprop; total time=  21.2s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__neurons=150, model__optimizer=rmsprop; total time=  21.2s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.3, model__neurons=100, model__optimizer=adam; total time=  13.2s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.3, model__neurons=100, model__optimizer=adam; total time=  13.1s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.3, model__neurons=100, model__optimizer=adam; total time=  13.2s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.2, model__neurons=50, model__optimizer=adam; total time=  12.6s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.2, model__neurons=50, model__optimizer=adam; total time=  12.6s
[CV] END batch_size=64, epochs=50, model__dropout_rate=0.2, model__neurons=50, model__optimizer=adam; total time=  13.4s
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.3, model__neurons=50, model__optimizer=adam; total time=  23.3s
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.3, model__neurons=50, model__optimizer=adam; total time=  23.2s
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.3, model__neurons=50, model__optimizer=adam; total time=  23.7s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.4, model__neurons=150, model__optimizer=adam; total time=  23.4s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.4, model__neurons=150, model__optimizer=adam; total time=  23.2s
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.4, model__neurons=150, model__optimizer=adam; total time=  23.4s
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  24.2s
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  25.1s
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.4, model__neurons=100, model__optimizer=adam; total time=  24.4s
LSTM with Attention R²: 0.5442471858275528
LSTM with Attention MAE: 145.25845695085897
LSTM with Attention RMSE: 353.05036764737406
LSTM with Attention Price Movement Accuracy: 0.9491279069767442
```

Using Attention Metrics to Interpret LSTM

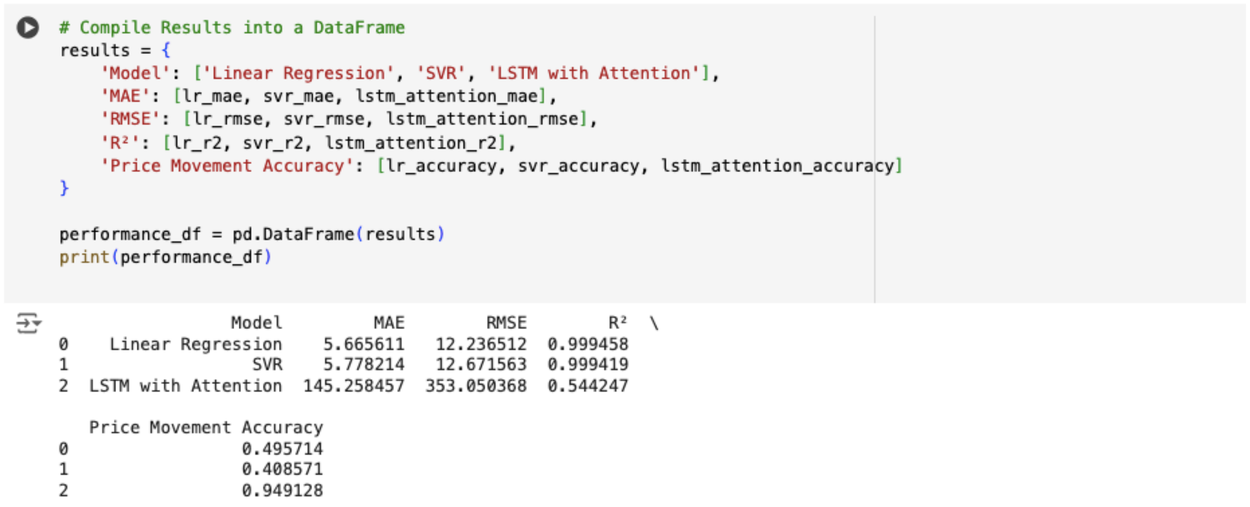
1. R^2 (R-squared): With an R2 of 0.5442, the LSTM with Attention model accounts for roughly 54.42% of the variation in stock prices. This comparatively low score indicates that, in comparison to other models, the model is less successful in capturing the link between the characteristics and the stock prices, leading to a poorer fit to the data.

2. Mean Absolute Error, or MAE : With an MAE of 145.2585, the predictions made by the LSTM with Attention model are inaccurate by an average of 145.26 units. Because the model frequently deviates from the actual stock values, this high error suggests that the forecasts made by the model are substantially less accurate.

3. Root Mean Squared Error, or RMSE : An RMSE of 353.0504 indicates that there is an average inaccuracy of approximately 353.05 units in the model's predictions. This high score indicates significant prediction mistakes, suggesting that greater errors are common and that the model has difficulty precisely predicting the stock values.

4. Accuracy of Price Movement: With a 94.91% price movement accuracy, the LSTM with Attention model can accurately forecast stock price movements 94.91% of the time. Despite its significant faults in precise price prediction, this extraordinarily high accuracy shows the model's power in predicting whether stock prices will rise or fall, making it extremely beneficial for strategic decision-making.

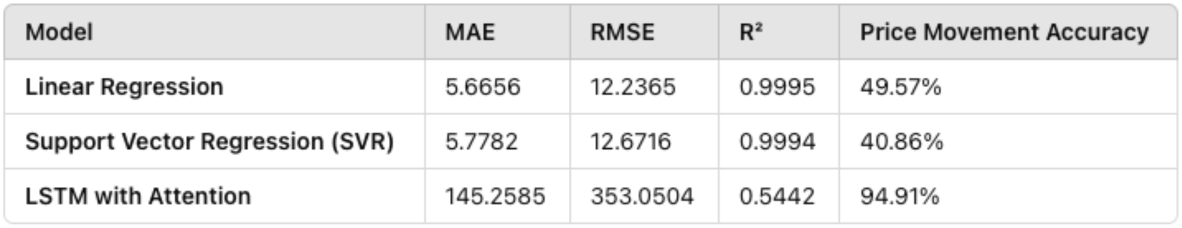
Performance Comparison of all three model’s



```
# Compile Results into a DataFrame
results = {
    'Model': ['Linear Regression', 'SVR', 'LSTM with Attention'],
    'MAE': [lr_mae, svr_mae, lstm_attention_mae],
    'RMSE': [lr_rmse, svr_rmse, lstm_attention_rmse],
    'R²': [lr_r2, svr_r2, lstm_attention_r2],
    'Price Movement Accuracy': [lr_accuracy, svr_accuracy, lstm_attention_accuracy]
}

performance_df = pd.DataFrame(results)
print(performance_df)
```

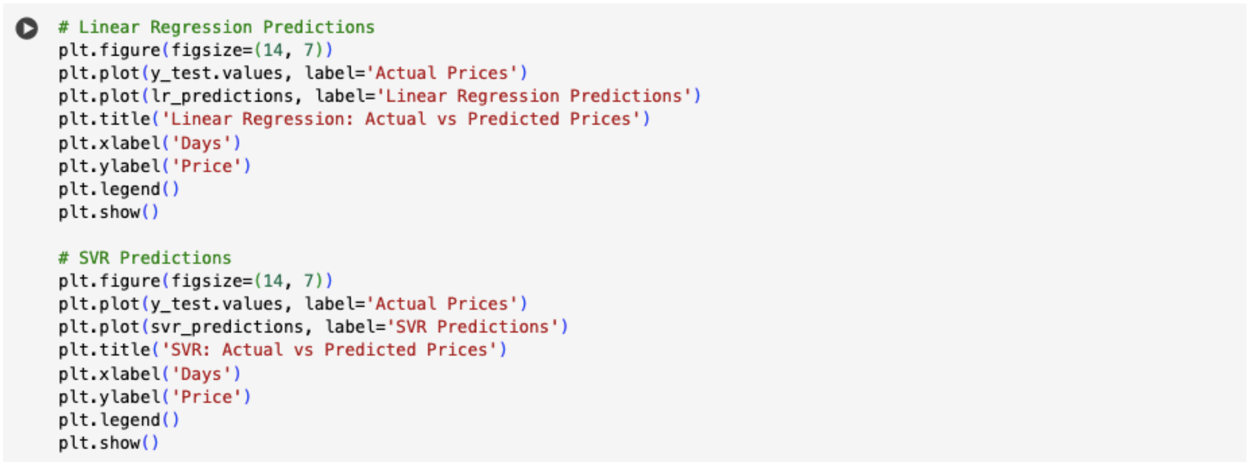
```
                 Model         MAE        RMSE        R²  \
0    Linear Regression    5.665611   12.236512  0.999458
1                  SVR    5.778214   12.671563  0.999419
2  LSTM with Attention  145.258457  353.050368  0.544247

   Price Movement Accuracy
0                 0.495714
1                 0.408571
2                 0.949128
```

❖ The three models that were used to forecast the price of Amazon stock are summarised in the table below. The coefficient of determination (R2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) are among the metrics taken into account.



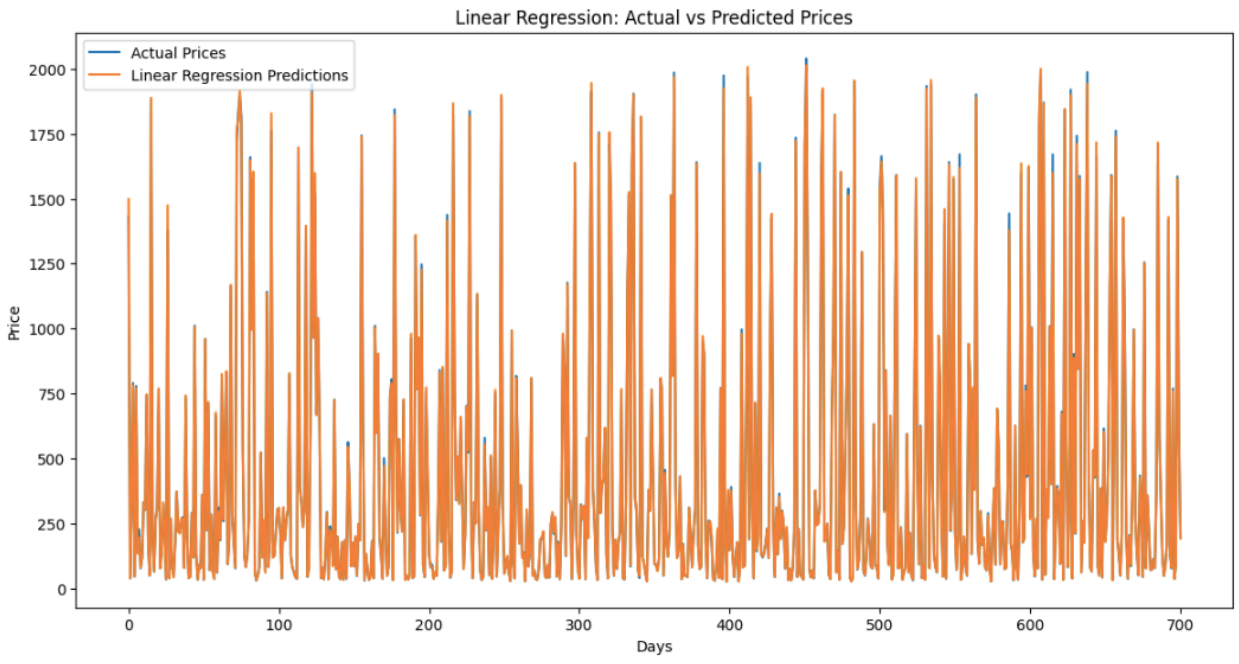
Model	MAE	RMSE	R²	Price Movement Accuracy
Linear Regression	5.6656	12.2365	0.9995	49.57%
Support Vector Regression (SVR)	5.7782	12.6716	0.9994	40.86%
LSTM with Attention	145.2585	353.0504	0.5442	94.91%

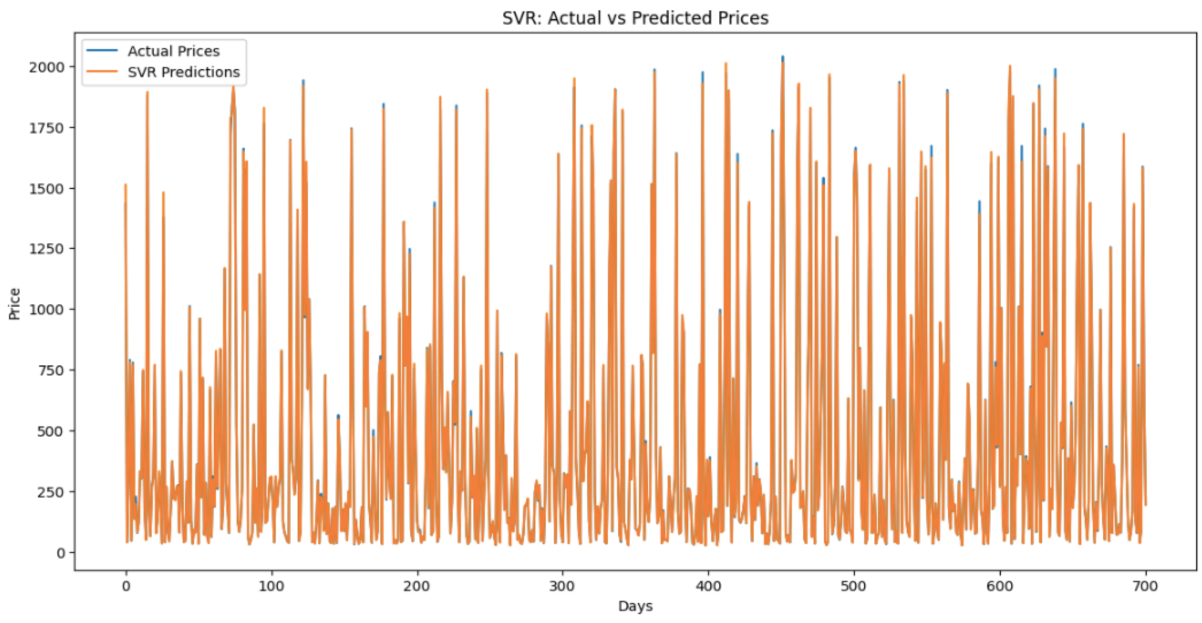
Visualization of Model Performance

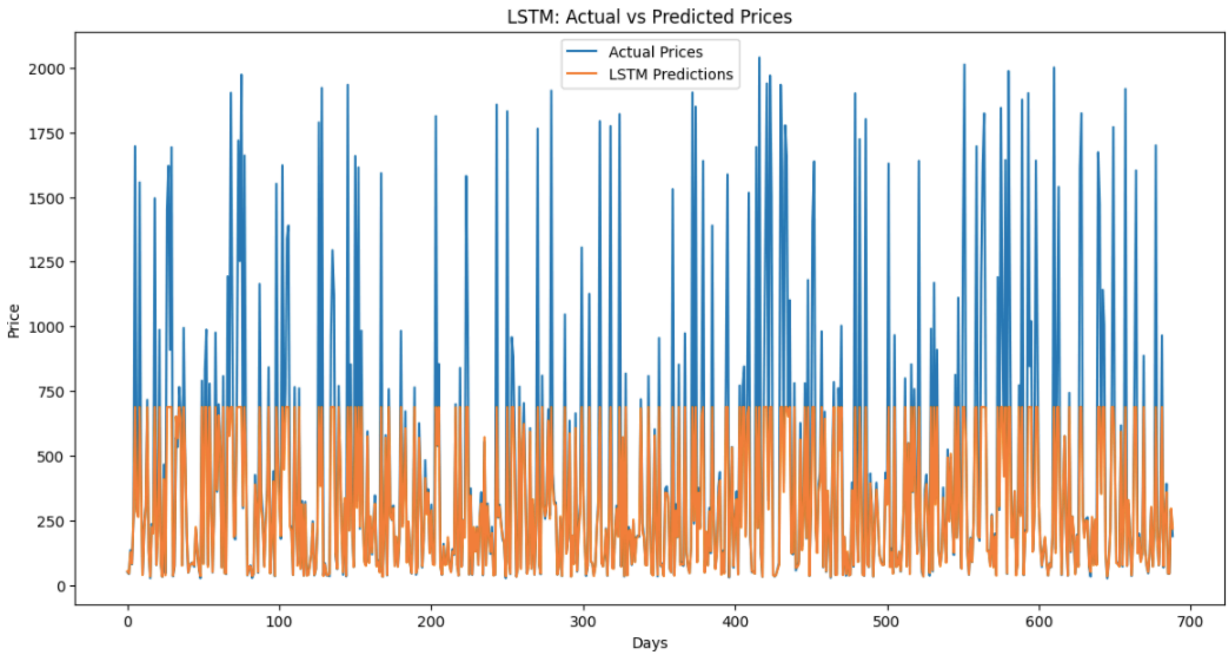


```
# Linear Regression Predictions
plt.figure(figsize=(14, 7))
plt.plot(y_test.values, label='Actual Prices')
plt.plot(lr_predictions, label='Linear Regression Predictions')
plt.title('Linear Regression: Actual vs Predicted Prices')
plt.xlabel('Days')
plt.ylabel('Price')
plt.legend()
plt.show()

# SVR Predictions
plt.figure(figsize=(14, 7))
plt.plot(y_test.values, label='Actual Prices')
plt.plot(svr_predictions, label='SVR Predictions')
plt.title('SVR: Actual vs Predicted Prices')
plt.xlabel('Days')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
# LSTM Predictions
plt.figure(figsize=(14, 7))
plt.plot(y_lstm_test, label='Actual Prices')
plt.plot(lstm_attention_predictions, label='LSTM Predictions')
plt.title('LSTM: Actual vs Predicted Prices')
plt.xlabel('Days')
plt.ylabel('Price')
plt.legend()
plt.show()
```







Conclusion:

Summary: Linear Regression Performance Metrics:- MAE: 5.6656, RMSE: 12.2365, R²: 0.9995, Accuracy of Price Movement: 49.57%

Insights: With an extraordinarily high R2 value, the Linear Regression model is able to account for 99.95% of the variation in Amazon stock prices. The accuracy of this model in predicting stock prices is further supported by its low MAE and RMSE values. Its price movement accuracy, which is just slightly better than haphazard guesswork, draws attention to a shortcoming in anticipating the direction of price movements. This implies that while linear regression is a trustworthy method for projecting stock prices, traders who need to know if prices will rise or fall may find it less useful.

Support Vector Regression (SVR):- Performance Metrics: MAE: 5.7782, RMSE: 12.6716, R^2: 0.9994, Price Movement Accuracy: 40.86%

Insights: With a low error rate and an R2 value of 0.9994, SVR showed predictive accuracy comparable to that of Linear Regression. This demonstrates that SVR is capable of reliably predicting stock values by successfully capturing the underlying patterns in the data. Nevertheless, SVR's price movement accuracy is less than 50%, similar to Linear Regression, meaning it frequently fails to accurately predict whether stock prices will rise or fall. This demonstrates a serious flaw with SVR in situations where directional accuracy is essential.

LSTM with Attention model:- Performance metrics: MAE: 145.2585, RMSE: 353.0504, R^2: 0.5442, Price Movement Accuracy: 94.91%

Insights: The high error metrics and R^2 value of the model suggest that it is less accurate at predicting stock prices than Linear Regression and SVR. The model appears to have difficulty predicting exact prices, as indicated by the high MAE and RMSE, which point to notable departures from real prices.

On the other hand, its remarkable 94.91% price movement accuracy indicates how well it can predict the direction of stock price trends. For traders who are more interested in forecasting market fluctuations than precise pricing, this makes the LSTM with Attention model especially useful. General Takeaways: - Model Equivalency: For accurate stock price forecasts, both SVR and linear regression perform well in terms of R^2 and low error metrics. Their lack of accuracy in predicting price fluctuations, however, makes them less suitable for trading techniques that rely on market patterns. On the other hand, the LSTM with Attention model is quite useful for trend-based trading techniques since, even with its high prediction errors, it provides good directional accuracy.

Important Use: Linear regression and SVR are suggested for applications like financial planning and risk management that need accurate stock price projections. These models offer low error, dependable price projections. On the other hand, because of its better ability to anticipate price fluctuations, the LSTM with Attention model is more suitable for trading techniques that rely on knowing whether stock prices will rise or fall.

Future Work: Future research could concentrate on hybrid approaches that integrate the advantages of deep learning models and conventional regression techniques in order to improve the prediction power of these models. Integrating features from both kinds of models may be necessary to increase accuracy overall. Furthermore, adding more external variables like sentiment on social media, global financial trends, and macroeconomic indicators might yield a more complete dataset for model training. As market conditions change, accuracy will need to be continuously updated and retrained using new data.

Final Reflections: This experiment has shown the various advantages and disadvantages of various machine learning models for speculating on Amazon stock prices. Conventional models such as SVR and linear regression are quite good at forecasting actual prices, but they are not very good at predicting the direction of prices. However, even with its larger prediction errors, the LSTM with Attention model does a great job of tracking price changes. Investors and companies can improve their decision-making processes by utilising the advantages of each model, striking a balance between trend research and accuracy to strengthen their strategy.