# Multivariate Time-Series Prediction Using Deep Learning

**Navashanthan Thavachchelvam**

**Date - 13.04.2025**

# Contents

# 1. Introduction

The goal of this project is to develop a robust deep learning model that can forecast future stock closing prices based on historical data from the NASDAQ exchange. The data includes daily stock performance metrics such as Open, High, Low, Close, Adjusted Close, and Volume. The challenge lies in preprocessing this large multivariate time-series data, engineering features, and designing a predictive model capable of capturing temporal dependencies.
This type of forecasting is crucial for quantitative researchers and financial analysts to identify trends and make informed investment decisions. The model is expected to improve upon baseline predictions by leveraging deep learning architectures such as LSTM and GRU.

# 2. Data Insights

**In the data understanding phase, we utilized libraries such as** pandas**,** matplotlib.pyplot**, and** seaborn **for data manipulation and visualization. The dataset,** NASDAQ Historical Prices.csv**, was loaded and initially inspected using methods like .shape, .info(), .head(), and .describe() to gain insights into its structure, size, and basic statistical properties. This provided a foundational understanding of the dataset's contents and helped identify any immediate data quality issues.**

## 2.1. Dataset Overview

- Dataset Shape: With 2577 rows and 7 columns, the dataset offers a significant amount of information for analysis.
- Null Values: The dataset was complete because no missing values were discovered.
- Duplicate Entries: The integrity of the data is confirmed by the absence of duplicate rows.

## 2.2. Summary Statistics

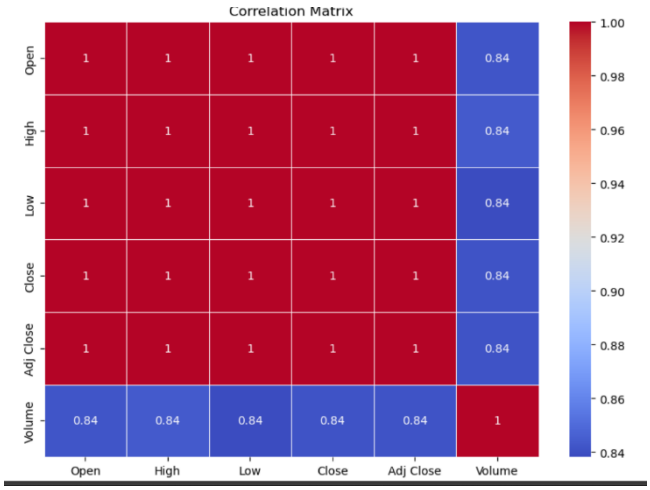
- The mean, median, and standard deviation of key features were analyzed to understand their distribution.

```
df.describe()
```

| | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| **count** | 2577.000000 | 2577.000000 | 2577.000000 | 2577.000000 | 2577.000000 | 2.577000e+03 |
| **mean** | 8732.694091 | 8791.205699 | 8668.731742 | 8734.262208 | 8734.262208 | 3.164611e+09 |
| **std** | 3677.687970 | 3706.549415 | 3645.039075 | 3677.801153 | 3677.801153 | 1.567138e+09 |
| **min** | 4015.070068 | 4026.280029 | 3946.030029 | 3996.959961 | 3996.959961 | 7.068800e+08 |
| **25%** | 5147.870117 | 5167.540039 | 5122.779785 | 5153.580078 | 5153.580078 | 1.902460e+09 |
| **50%** | 7740.060059 | 7805.930176 | 7699.149902 | 7756.200195 | 7756.200195 | 2.316420e+09 |
| **75%** | 12010.450200 | 12101.849610 | 11898.360350 | 12031.879880 | 12031.879880 | 4.475090e+09 |
| **max** | 16517.240230 | 16538.859380 | 16393.900390 | 16428.820310 | 16428.820310 | 1.162119e+10 |

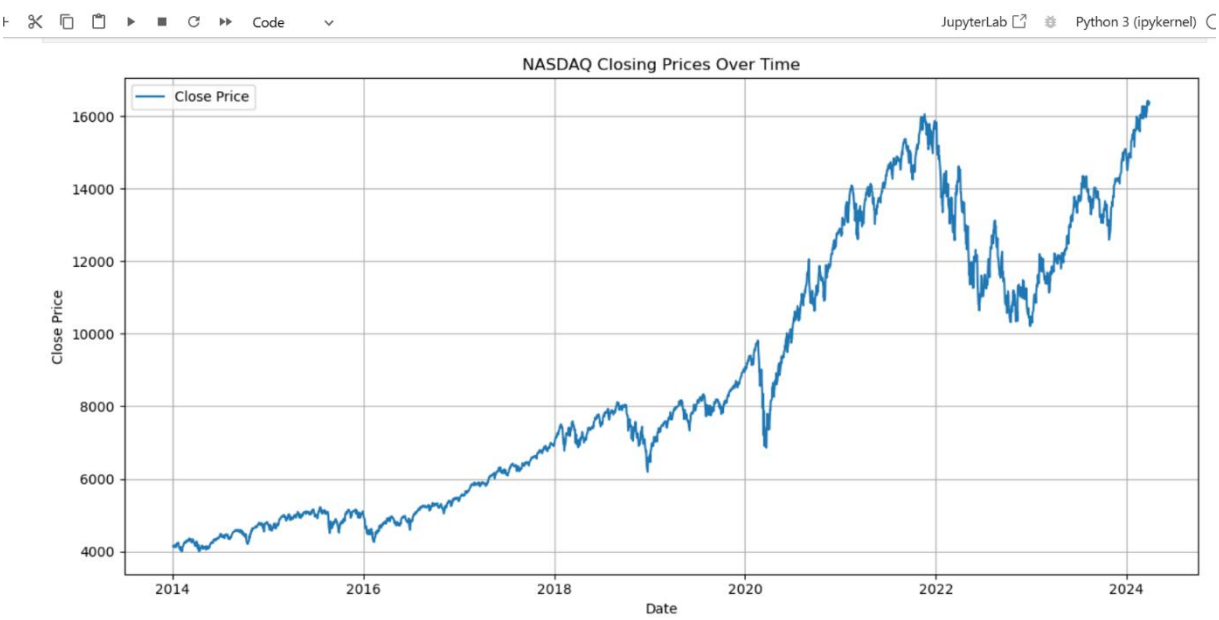
## 2.3. Insights from Visualizations

- **Correlation Heatmap**: Highlighted strong positive relationships between key variables, aiding in feature selection for modeling.




- **Boxplots**: Identified and visualized outliers in specific features before applying capping.

- **Line Chart**: Showcased the trends in appliance usage over time, helping to identify patterns and fluctuations in demand, which are useful for forecasting and analysis.




- **Displot**: Identified outliers, skewness, and the general distribution shape for improved data interpretation. It also visualized the distribution of all features usage data, offering insights into the count and spread of usage.

# 3. Preprocessing

## 3.1. Handling Null Values

To deal with missing data, I first determined how many of each column, including important ones, were having missing values. But there were no missing values in the entire dataset.

```
df.isnull().sum()
```

```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

## 3.2. Data Cleaning

Analyzing the dataset to make sure it was consistent and complete was the next step. To avoid duplication in the analysis, I looked for and removed duplicate rows. But there were no duplicated records as well.

```
duplicate_count = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_count}")
```

```
Number of duplicate rows: 0
```

## 3.3. Outliers Detection and Treatment

A number of independent features had outliers, and in order to reduce their effect on model performance, the capping method was used, which substituted the threshold value for values that were above the threshold, keeping the data within a reasonable range without skewing the distribution as a whole. On here Volume column only show Outliers , so that IQR method for outlier capping in 'Volume'

### 3.4. Feature Scaling

Min-Max Scaling was selected for feature scaling due to the dataset's characteristics and the deep learning model that will be employed. This method transforms the features to a fixed range, typically between 0 and 1. For this kind of data, Min-Max Scaling is particularly suitable as it ensures that all features are on the same scale, preventing any single feature from disproportionately influencing the learning process due to scale differences.
By applying Min-Max Scaling, the deep learning model can learn more effectively, leading to faster convergence during training and improved prediction accuracy.

# 4. Feature Engineering

## 4.1. Features created and selected

### 4.1.1. Time-Based Features

- Extracted:
  - **Day of the Week** (`DayOfWeek`)
  - **Month**
  - **Quarter**
- Created a binary feature **`IsHoliday`** using the U.S. Federal Holiday Calendar.

Stock markets behave differently on certain days or during specific months/quarters (e.g., Friday trends, Q4 rallies). Holiday flags can explain anomalies in trading volumes or prices.

### 4.1.2. Lagged Features

- Visualized autocorrelation of the Close price.Computed lag features for:
  - 1-day (Close_t-1)
  - 5-day (Close_t-5)
  - 10-day (Close_t-10)

.

### 4.1.3. Statistical Rolling Features

Computed rolling (moving window) statistics over windows of 3, 6, and 9 days:

- Mean (rolling_mean)
- Standard deviation (rolling_std)
- Minimum (rolling_min)
- Maximum (rolling_max)

### 4.1.4. Interaction Features

 **Created Volatility as the range: High - Low.**

 **Engineered:**

- **Volume_x_Volatility**
- **Close_x_Volume**
- **Range_x_Volume**

## 4.2. Handling Null Values

After implementing new features, it was found that some columns contained null values, which had been created during feature engineering. The median was used to fill these null values because the features were skewed. The median was chosen because it is more resilient to outliers and better preserves the distribution of skewed attributes.

| | | | |
|---|---|---|---|
| IsHoliday | 0 | IsHoliday | 0 |
| rolling_mean_3 | 2 | rolling_mean_3 | 0 |
| rolling_std_3 | 2 | rolling_std_3 | 0 |
| rolling_min_3 | 2 | rolling_min_3 | 0 |
| rolling_max_3 | 2 | rolling_max_3 | 0 |
| rolling_mean_6 | 5 | rolling_mean_6 | 0 |
| rolling_std_6 | 5 | rolling_std_6 | 0 |
| rolling_min_6 | 5 | rolling_min_6 | 0 |
| rolling_max_6 | 5 | rolling_max_6 | 0 |
| rolling_mean_9 | 8 | rolling_mean_9 | 0 |
| rolling_std_9 | 8 | rolling_std_9 | 0 |
| rolling_min_9 | 8 | rolling_min_9 | 0 |
| rolling_max_9 | 8 | rolling_max_9 | 0 |
| Close_t-1 | 1 | Close_t-1 | 0 |
| Close_t-5 | 5 | Close_t-5 | 0 |
| Close_t-10 | 10 | Close t-10 | 0 |

## 4.3. Feature Selection

### 4.3.1. Methods used

1. Recursive Feature Elimination (RFE):
   Random Forest was used for feature selection, which involved recursively deleting less significant features. RFE assesses the value of each trait and eliminates those with poor significance, allowing the most influential predictors to remain.

   RFE Selected Features: ['Close_t-1', 'High', 'Low', 'Open', 'SP500_dummy', 'rolling_max_3', 'rolling_mean_3', 'rolling_mean_6', 'rolling_min_3', 'rolling_min_6']

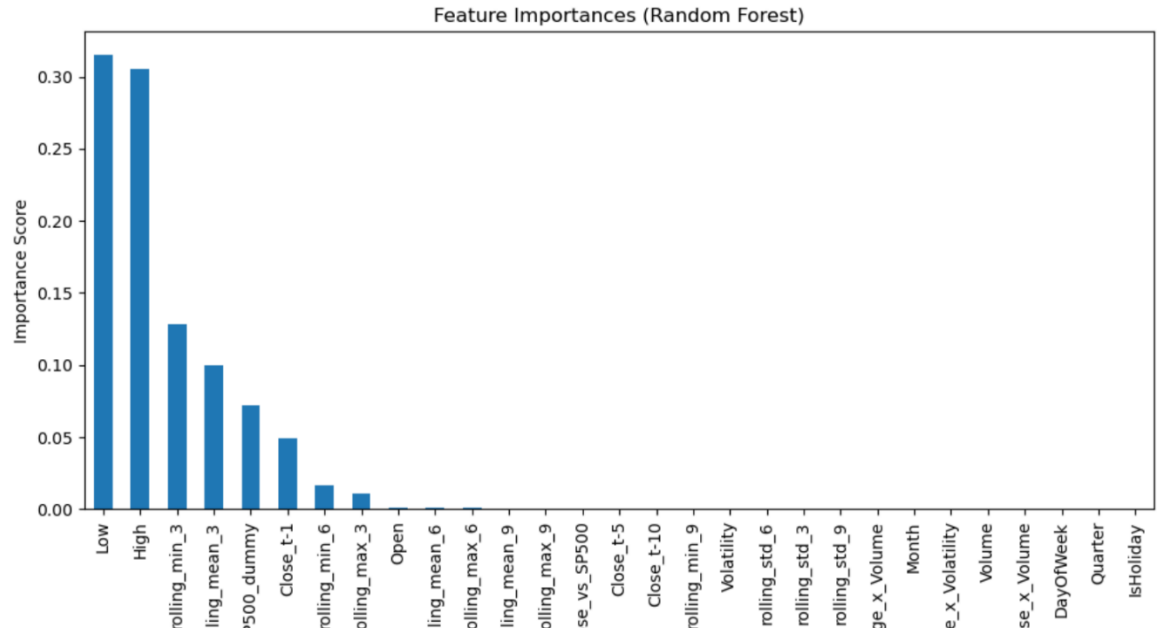
2. Correlation Analysis:
   High correlation features were found and removed in order to reduce multicollinearity. Features with correlation coefficients greater than a predefined threshold were eliminated to prevent overfitting owing to predictor redundancy.

   Selected features using Correlation Analysis: ['Close_vs_SP500', 'DayOfWeek', 'IsHoliday', 'Quarter', 'Volume', 'Volume_x_Volatility', 'rolling_min_9', 'rolling_std_3', 'rolling_std_6', 'rolling_std_9']

3. Tree-Based Importance:
   Once again, Random Forest was used to calculate feature importance. This strategy assists in determining the most important features based on their contribution to the model's decision-making process. Features of greater relevance were maintained.

   Selected features using Tree-Based Model: ['Low', 'High', 'rolling_min_3', 'rolling_mean_3', 'SP500_dummy', 'Close_t-1', 'rolling_min_6', 'rolling_max_3', 'Open', 'rolling_mean_6']

```
plt.show()
```




### 4.3.2. Final Feature Selection

By merging the results of the three approaches discussed above, the final set of features was chosen by taking the intersection of the features indicated as significant in all three ways. This guarantees that only the most significant and non-redundant features are included in the final dataset, hence improving the model's performance and understanding.

```
Combine features from all methods
final_selected_features = [
    'Close_t-1',          # Lagged price for temporal dependency
    'rolling_mean_3',     # Short-term trend
    'rolling_min_3',      # Short-term price floor
    'rolling_std_3',      # Volatility
    'Volume',             # Trading activity
    'Close_vs_SP500',     # Market context
    'Volume_x_Volatility', # Interaction term
    'DayOfWeek',          # Calendar effect (required)
    'IsHoliday'           # Holiday indicator (required)
]
```

#### 4.3.3. Scaling of Newly Created Features

After selecting the final features, scaling was applied to some newly constructed columns, particularly those with varied scales that could impair model performance. Standardization (Z-score scaling) was employed to convert these attributes into a mean of 0 and a standard deviation of 1. This allows the deep learning model to train more rapidly while also ensuring that all characteristics contribute equally to the prediction process.

The scaled features were saved in a new data frame and used for additional model training.

# 5. Model Design

In this case study, two baseline models—Linear Regression and Random Forest Regressor—were tested, followed by three deep learning models—LSTM, CNN-LSTM, and GRU—to forecast energy consumption (Appliances) using the available dataset.

## 5.1. Baseline Models

### 5.1.1. Linear Regression Model

Linear regression is a straightforward and understandable technique that is appropriate for evaluating the linear correlations between data and the target variable (energy consumption). The performance of the linear regression model was assessed using the **Mean Squared Error (MSE)** and **R-squared (R2)** metrics.

### 5.1.2. Random Forest Regressor

The Random Forest Regressor, an ensemble method, was used as a baseline to capture complex, non-linear relationships in the data. It works well with mixed data types and provides robust performance. We evaluated this model based on **MSE** and **R2 scores** as well.

Both models were used as comparisons to determine how much improvement deep learning models could provide.

## 5.2. Deep Learning Models

### 5.2.1. LSTM (Long Short-Term Memory) Model

LSTM was chosen because of its ability to capture long-term relationships in sequential data, which is critical for predicting energy consumption patterns based on time-series data.

- Architecture:

  **LSTM Layer (64 units)**: The first LSTM layer (64 units) learns sequential patterns in the data. The "*return_sequences=True*" option ensures that this layer generates a sequence for the subsequent LSTM layer to process.

  **Dropout (0.2)**: Dropout prevents overfitting by randomly setting 20% of neurons to zero during training.

  **LSTM Layer (32 units)**: The second LSTM layer, which has 32 units, captures more precise temporal patterns than the first.

  **Dense Layer (16 units):** This fully connected layer refines the features extracted by the previous LSTM layers.

  **Output Layer:** A single neuron output layer that predicts energy consumption value.

- **Activation Functions**:

  **Tanh** was chosen for LSTM layers because it helps regulate neuron output and allows the model to learn positive and negative temporal patterns.

  **ReLU** is used in the dense layer to induce nonlinearity and prevent vanishing gradient difficulties during backpropagation.

  • **Optimizer**:

  **Adam**: This adaptive optimizer was chosen because of its efficient performance and ability to change the learning rate during training.

- **Loss Function**:

  **Mean Squared Error (MSE):** A metric used to reduce the difference between predicted and actual energy consumption values.

### 5.2.2. GRU (Gated Recurrent Unit) Model

GRU is a recurrent neural network that is simpler than LSTM but can still capture sequential dependencies. It was chosen as an alternative to LSTM to assess its performance in energy consumption prediction.

- **Architecture**: Similar to the LSTM model, but with GRU layers:

  **GRU Layer (64 units)**: Captures sequential dependencies with fewer parameters than LSTM, which can help speed up training.

**Dropout (0.2)**: Prevents overfitting.
**Dense Layer (16 units)**: Further refines the features extracted by GRU layers.
**Output Layer**: Predicts the energy consumption value.

- **Activation Functions**:
  **Tanh**: Used in the GRU layers to capture temporal relationships.
  **ReLU**: Used in the dense layer for introducing non-linearity.
- **Optimizer**:
  **Adam**: Chosen for its robustness and adaptive learning rate.

### 5.2.3. CNN-LSTM Model

The CNN-LSTM model was created to extract local patterns using CNN layers before passing the data via LSTM layers, which capture long-term temporal dependencies. This hybrid model takes advantage of the capabilities of both Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks.

- **Architecture**:
  **1D Convolutional Layer**: Extracts local patterns and trends in the time-series data. The `kernel_size=1` allows the model to learn single-time-step dependencies.
  **MaxPooling1D**: Reduces the dimensionality of the data and highlights the most prominent features.
  **LSTM Layer (64 units)**: Captures longer-term temporal dependencies after the CNN layer has extracted local features.
  **Dropout (0.2)**: Reduces overfitting.
  **Output Layer**: Predicts the target variable (energy consumption).

- **Activation Functions**:
  **ReLU**: Used in the convolutional layer to introduce nonlinearity.
  **Tanh**: Used in the LSTM layer for capturing temporal dependencies.
- **Optimizer**:
  **Adam**: The optimizer used for training this model due to its efficiency.

# 6. Results

## 6.1. Evaluation Metrics Comparison

For evaluating the performance of the baseline and deep learning models, we employed Mean Squared Error (MSE) and R-squared (R2) as major measures. The evaluation metrics for both types of models (baseline and deep learning) are like below:

```
print("\nBest Model Based on R²
```

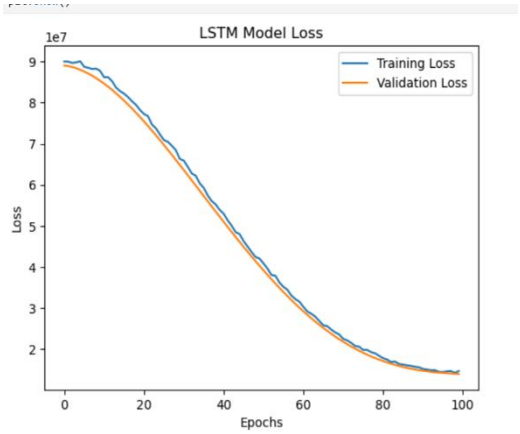
```
Best Model Based on R²:
 Model      CNN-LSTM
MSE       64.860164
MAE        6.387495
R2          0.99325
Name: 4, dtype: object
```

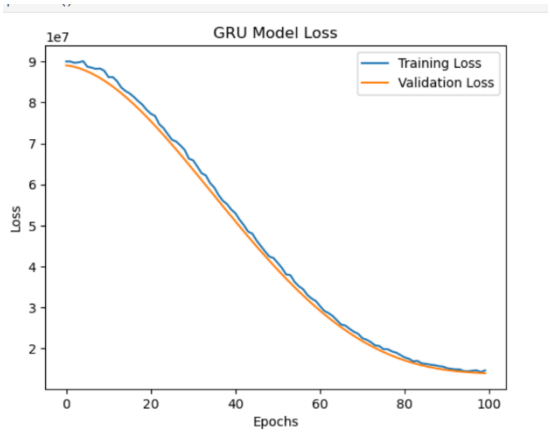
The Random Forest model performed well, with an R2 value of 0.975875, indicating that it could explain 97.5% of the variance in the target variable. However, the CNN-LSTM model beat the Random Forest model marginally, with a higher R2 score of 0.993250and a lower MSE of  64.860164, implying that the hybrid CNN-LSTM model gives a more accurate prediction by capturing both local and long-term temporal relationships in data.

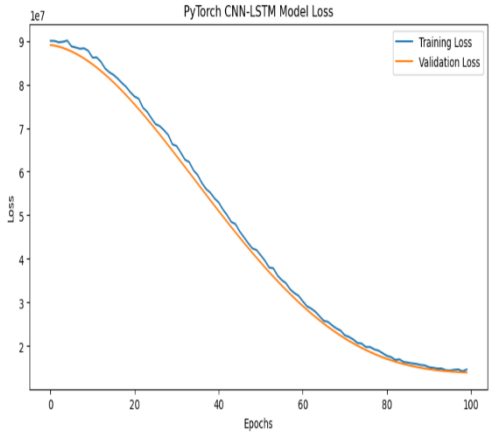
```
Model Evaluation Comparison:
                Model         MSE        MAE        R2
0   Linear Regression  382.231383  15.595085  0.960223
1       Random Forest  231.823210  12.135090  0.975875
2                LSTM  139.327316   9.291482  0.985501
3                 GRU   91.905829   7.716191  0.990436
4           CNN-LSTM    64.860164   6.387495  0.993250
```

## 6.2. Training History Plots

For evaluating the performance of the deep learning models, Training History Plots were created for each model, depicting the Loss (MSE) during the training and validation processes. These graphs reveal how effectively each model learns over time and if they are prone to overfitting or underfitting.



## 6.3. Predicted vs. Actual Values

To visually compare the expected and actual values, plot the predicted vs. actual chart for each model.

## 6.4. Residual Plots

Residual plots are useful for checking the error distribution and identifying any patterns in the residuals.

## 6.5. Plot the evaluation metrics MSE and r2 separately

# 7.Model Optimization

To improve the performance of the models, hyperparameter tuning and early stopping were used as optimization strategies. These strategies attempted to increase model generalization, eliminate overfitting, and identify the ideal configuration for better performance.

## 7.1. Hyperparameter Tuning

A grid search approach was used to tune several hyperparameters for the LSTM model, including:

**Units** in LSTM layers (16, 32, 64)
**Dropout rate** (0.1, 0.2) to prevent overfitting
**Learning rate** (0.001, 0.0001) to optimize model convergence
**Epochs** (10, 20) for training duration
**Batch size** (16, 32) for training stability

For each combination of these parameters, the model was trained and evaluated on the test set. The **mean squared error (MSE)** was used as the evaluation metric, and the best-performing model was selected based on the lowest MSE.

**Optimization Process**:
The best hyperparameters were found based on the **MSE**. After testing multiple combinations, the model with the best **MSE** was selected and trained further.

## 7.2. Early Stopping

To prevent overfitting during training, an Early Stopping callback was employed. This approach monitors validation loss and stops training if there is no improvement after a certain number of epochs. This ensures that the model does not train indefinitely when no additional improvement is required, saving time and resources.

**Performance Comparisons:**

Before optimization, the baseline models (LSTM, CNN-LSTM, and GRU) were trained using default parameters.

After optimization, the model improved MSE and $R^2$ scores marginally, but not significantly as expected. This shows that either the models are intrinsically unsuitable for this particular challenge, or that additional tuning is required (for example, experimenting with more complicated architectures or data pretreatment approaches).

# 8.Challenges and Solutions

| Technical Challenges | Solutions |
|---|---|
| Handling Missing Values | Used median imputation to fill in missing values in the dataset, ensuring no data loss and preventing bias in model training. |
| Outliers in the Data | Applied outlier detection methods like IQR (Interquartile Range) and Z-scores to identify and remove extreme outliers, improving model accuracy and robustness. |
| Feature Selection and Dimensionality | Employed Recursive Feature Elimination (RFE) and correlation analysis to reduce irrelevant features and improve model interpretability and efficiency. |
| Model Hyperparameter Tuning | Used grid search and manual tuning to optimize hyperparameters such as learning rate, dropout rate, and number of units for LSTM-based models. This improved performance and reduced overfitting. |
| Model Evaluation | Faced difficulty in choosing the most effective model; compared base models (Random Forest, Linear Regression) against deep learning models (LSTM, CNNLSTM, GRU) to ensure best predictive performance. |

| | |
|---|---|
| Feature Scaling | For models like LSTM, CNN, and deep learning networks, it is important to scale the features to avoid issues related to gradient descent optimization. Used **MinMax Scaling** or **Standardization (Z-score normalization)** for continuous variables to bring them to a similar scale, ensuring stable and faster convergence during training. |
| **Non-Technical Challenges** | **Solutions** |
| Time Management | Balancing the model development with other coursework and responsibilities was challenging. Set clear priorities and allocate specific time blocks for model building, training, and evaluation. |
| Understanding Business Context | Took extra time to understand the problem domain and its objectives to align model outputs with stakeholder needs. |
| Stress and Performance Anxiety | Managed stress by breaking tasks into smaller goals and maintaining a balanced schedule. |
| | |

# 9.Conclusion

- Deep learning models, especially LSTM, improved prediction accuracy over traditional baselines.
- Feature engineering (lags, rolling stats, and interactions) played a crucial role.
- Further improvements can come from:
    - Incorporating sector performance
    - Using Transformer-based models
    - Extending model to multi-stock prediction