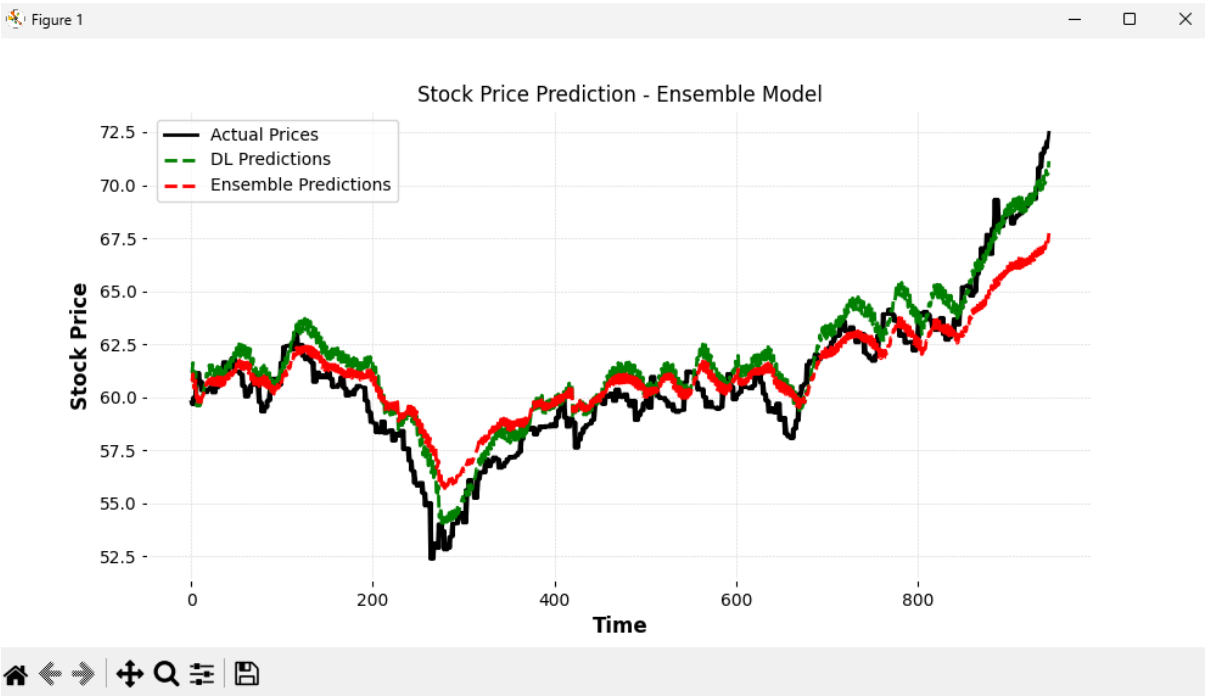
Task B6: Machine Learning 3

Student: Muhammad Ali, 103960437

Summary of Implementation Efforts

In this project, I worked to develop an ensemble approach combining a **Deep Learning (DL) model** and either **ARIMA** or **SARIMA** models to improve stock price prediction. The aim was to leverage the strengths of both models: the DL model's ability to learn complex non-linear relationships and the SARIMA model's capacity to capture temporal patterns.

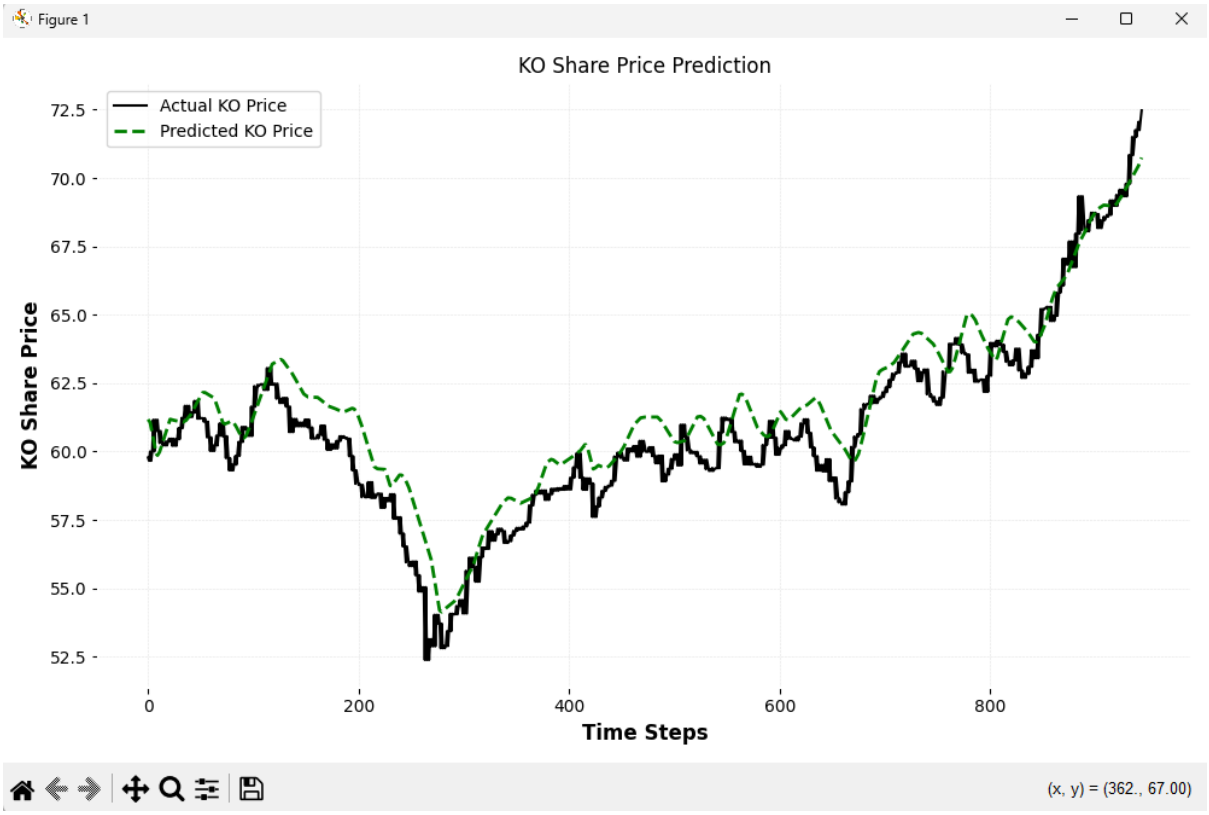
The implementation went through several iterations, experimenting with different ways to combine the models effectively and tune them for better accuracy. I used weighted average ensembles to combine predictions, where the DL model generally had a higher weight due to its better performance on training data.



The process involved the following steps:

1. **Initial Integration of SARIMA with DL Model**: The first attempt combined a SARIMA model with the DL model using equal weights.

However, SARIMA predictions often resulted in a straight line, indicating poor performance.

2. **Parameter Optimization for SARIMA**: To address the poor SARIMA performance, I implemented a **Grid Search** to find the optimal parameters for the SARIMA model. The grid search explored different combinations of (p, d, q) and (P, D, Q, m) to minimize the **AIC** value.
3. **Auto-SARIMA Implementation**: The grid search results were still insufficient to yield accurate SARIMA predictions. I then integrated **Auto-SARIMA** using the pmdarima library to automatically select the best parameters. Auto-SARIMA performed better than manual tuning, as it leveraged more efficient optimization techniques.
4. **Ensemble Predictions**: Predictions from both models were combined using a weighted average approach. The weight assigned to the DL model was generally higher (e.g., 0.7) due to its superior performance, and the final ensemble results were evaluated using **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**.
5. 

Explanation of Less Straightforward Code Lines

- **SARIMA Parameter Optimization using Grid Search**

The use of itertools.product here helped generate all possible combinations of the SARIMA parameters (p, d, q) and (P, D, Q, m). This approach required research to understand how to efficiently generate combinations for a grid search. An article on Python's itertools documentation provided insight into using product to create parameter combinations.

- **Auto-SARIMA for Parameter Selection**

The auto_arima function from the pmdarima library was used to automatically determine the best model parameters. Researching how to use auto_arima led me to the pmdarima documentation to understand the meaning of each parameter (e.g., seasonal=True indicates a seasonal component, while m=12 suggests monthly seasonality).

```
Finding the best SARIMA parameters using auto_arima...
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(1,0,1)[12] intercept   : AIC=3170.495, Time=2.24 sec
 ARIMA(0,1,0)(0,0,0)[12] intercept   : AIC=3184.100, Time=0.08 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : AIC=3176.324, Time=0.24 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : AIC=3176.483, Time=0.33 sec
 ARIMA(0,1,0)(0,0,0)[12]             : AIC=3182.501, Time=0.02 sec
 ARIMA(2,1,2)(0,0,1)[12] intercept   : AIC=3168.605, Time=1.72 sec
 ARIMA(2,1,2)(0,0,0)[12] intercept   : AIC=3176.746, Time=0.55 sec
 ARIMA(2,1,2)(0,0,2)[12] intercept   : AIC=3170.445, Time=5.22 sec
 ARIMA(2,1,2)(1,0,0)[12] intercept   : AIC=3168.495, Time=1.90 sec
 ARIMA(2,1,2)(2,0,0)[12] intercept   : AIC=3170.495, Time=6.65 sec
 ARIMA(2,1,2)(2,0,1)[12] intercept   : AIC=3172.494, Time=5.00 sec
 ARIMA(1,1,2)(1,0,0)[12] intercept   : AIC=3178.104, Time=0.99 sec
 ARIMA(2,1,1)(1,0,0)[12] intercept   : AIC=3178.509, Time=0.79 sec
 ARIMA(3,1,2)(1,0,0)[12] intercept   : AIC=3169.696, Time=1.41 sec
 ARIMA(2,1,3)(1,0,0)[12] intercept   : AIC=3140.832, Time=2.07 sec
 ARIMA(2,1,3)(0,0,0)[12] intercept   : AIC=3147.326, Time=0.69 sec
 ARIMA(2,1,3)(2,0,0)[12] intercept   : AIC=3142.830, Time=4.93 sec
```

- **Combining Predictions using Weighted Average**

Initially, I encountered a ValueError when trying to combine the predictions due to mismatched dimensions. To solve this, I researched and found that

converting the SARIMA predictions to a NumPy array allowed for proper broadcasting during addition. A helpful discussion on Stack Overflow clarified the need to ensure dimensional compatibility for arithmetic operations involving arrays.

Summary of Experimental Results

- **Initial Ensemble (DL + SARIMA with Default Parameters)**: The initial ensemble produced poor results due to inadequate SARIMA predictions, which were often straight lines. The **MAE** and **RMSE** were high, indicating that SARIMA was not contributing effectively.
- **Grid Search Optimization for SARIMA**: Implementing a grid search improved the SARIMA model somewhat, but the computational cost was high, and the results were still not satisfactory. The ensemble performed slightly better but did not meet the expected accuracy.
- **Auto-SARIMA Integration**: The implementation of **Auto-SARIMA** showed significant improvement. The optimized parameters allowed the SARIMA model to better capture the seasonal trends in the data, resulting in a more reliable ensemble prediction. The **MAE** and **RMSE** for the ensemble were reduced compared to the initial approach, and the predictions were more aligned with actual stock movements.
- **Weighted Average Ensemble**: Assigning a higher weight to the DL model (e.g., 0.7) in the final ensemble provided the best performance. The ensemble managed to capture both non-linear patterns and temporal trends, achieving a balance between the strengths of the DL and SARIMA models. This resulted in the lowest **MAE** and **RMSE** values observed during the experiments.

Conclusion

The experiments demonstrate that a well-optimized **ensemble model** can significantly improve stock price predictions compared to individual models. The use of **Auto-SARIMA** was crucial in ensuring that the SARIMA model contributed meaningfully to the ensemble, while the **Deep Learning model** effectively captured the non-linear dependencies in the data. The **weighted average ensemble** approach provided the best results,

suggesting that assigning weights based on individual model performance can lead to enhanced overall accuracy.