CS419 Project Report

Stock Price Predictor

GitHub link: https://github.com/priyanshigupta31/CS419_Project

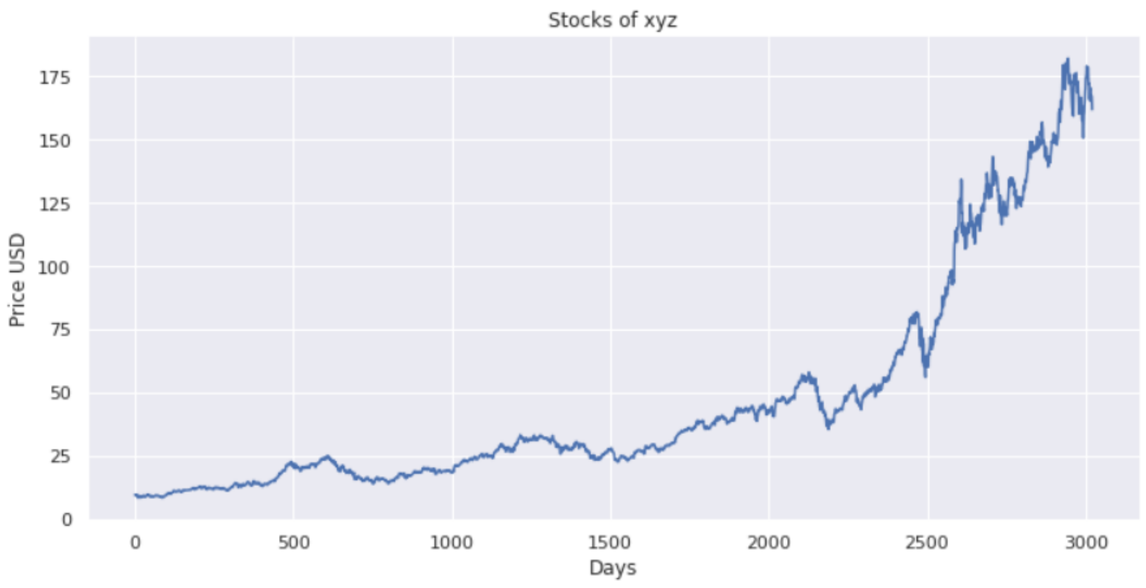
Team Members:
Yash Vagadia
Tejas Amritkar
Vinayak Goyal
Priyanshi Gupta

Problem Statement

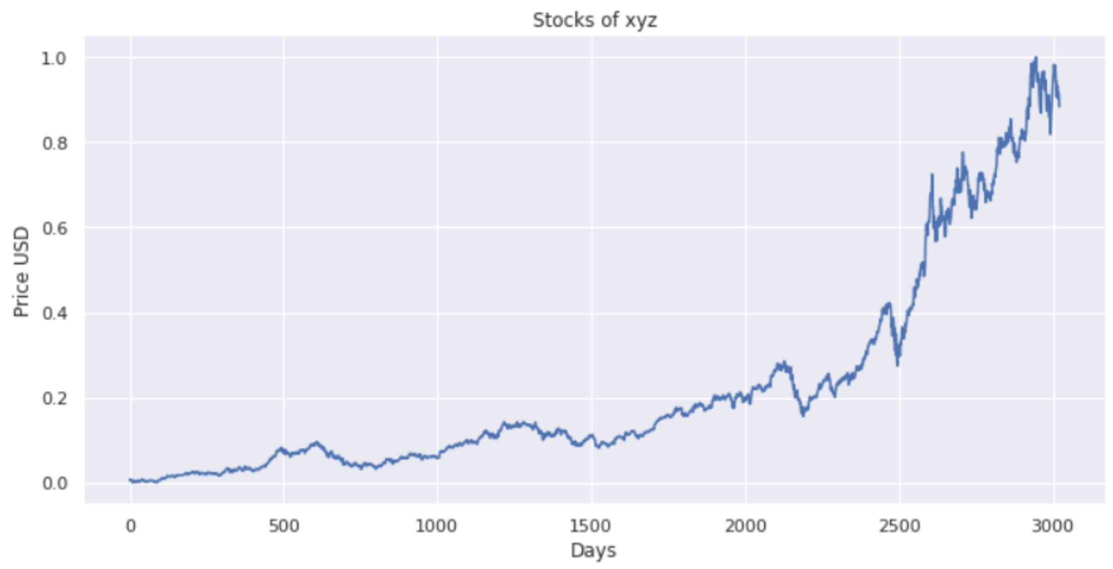
The aim of this project is to accurately predict the future closing value of a given stock across a given period of time in the future. For this project, we have used Long Short Term Memory Networks (LSTMs) to predict the closing price using a dataset of past prices.
The goal is to first implement a basic model using linear regression, then implement LSTM and compare the results.

Data Exploration

The data used in this project is of the Apple Inc. from April 23, 2010 to April 22, 2022, indexed in time order. The goal was to predict the closing price for any given date after training.



The data has been normalised the MinMaxScaler helper function from Scikit-Learn.



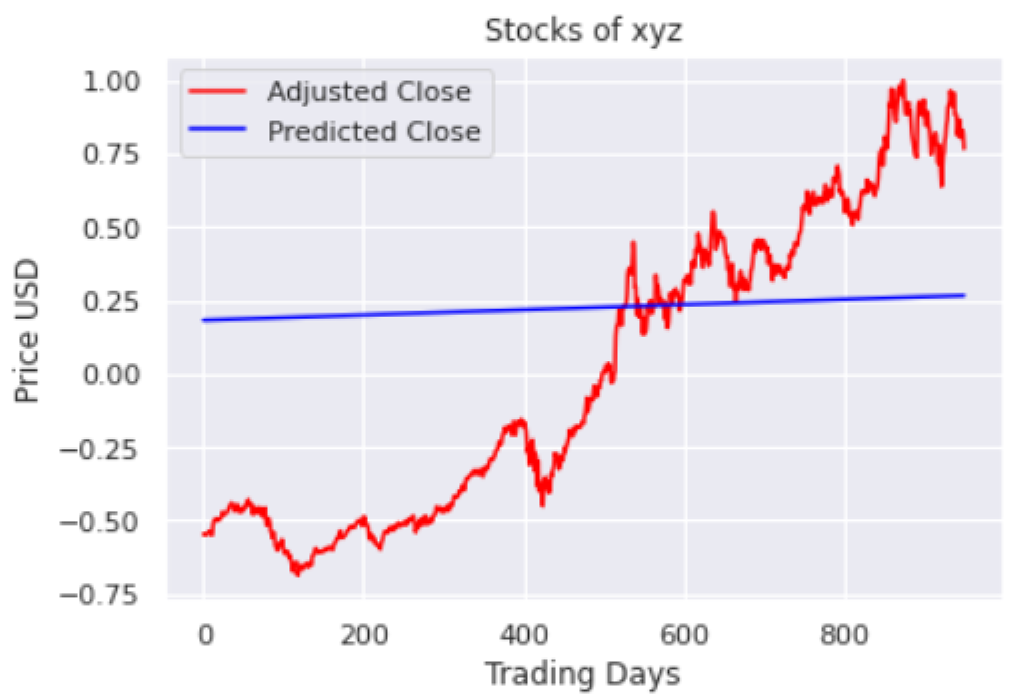
Algorithms and Techniques
The goal of this project was to study time-series data and explore as many options as possible to accurately predict the Stock Price. Through our research we came to know about Recurrent Neural Nets (RNN) which are used specifically for sequence and pattern 8 learning. As they are networks with loops in them, allowing information to persist and thus ability to memorise the data accurately. But Recurrent Neural Nets have vanishing Gradient descent problem which does not allow it to learn from past data as was expected. The remedy of this problem was solved in Long-Short Term Memory Networks , usually referred as LSTMs. These are a special kind of RNN, capable of 9 learning long-term dependencies.

In addition to adjusting the architecture of the Neural Network, the following full set of parameters can be tuned to optimize the prediction model:

• Input Parameters
• Preprocessing and Normalization (see Data Preprocessing Section)
• Neural Network Architecture
• Number of Layers (how many layers of nodes in the model; used 3)
• Number of Nodes (how many nodes per layer; tested 1,3,8, 16, 32, 64, 100,128)
• Training Parameters
• Training / Test Split (how much of dataset to train versus test model on; kept constant at 82.95% and 17.05% for benchmarks and lstm model)
• Validation Sets (kept constant at 0.05% of training sets)
• Batch Size (how many time steps to include during a single training step; kept at 1 for basic lstm model and at 512 for improved lstm model)
• Optimizer Function (which function to optimize by minimizing error; used “Adam” throughout)
• Epochs (how many times to run through the training process; kept at 1 for base

The measure of performance will be done using the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) values.

Benchmark Model
For this project we have used a Linear Regression model as its primary benchmark. As one of our goals is to understand the relative performance and implementation differences of machine learning versus deep learning models. This Linear Regressor was based on the examples presented in Udacity's Machine Learning for Trading course and was used for error rate comparison MSE and RMSE utilizing the same dataset as the deep learning models.



Following is the predicted results that we got from the benchmark model :

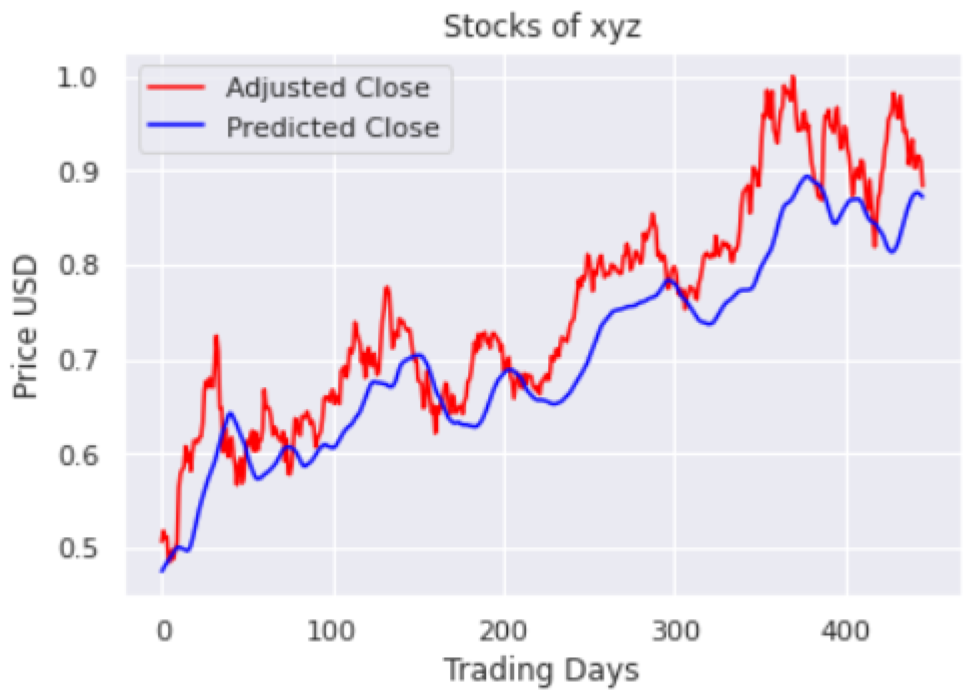
Train Score: 0.3410719086910154 MSE
Test Score:0.3410719086910154 MSE

Data Preprocessing
Acquiring and preprocessing the data for this project has been modularized into the preprocessing.py file for importing and use.

Refinement
For this project we have worked on fine tuning parameters of LSTM to get better predictions. We did the improvement by testing and analysing each parameter and then
selecting the final value for each of them.
To improve LSTM we have done following:

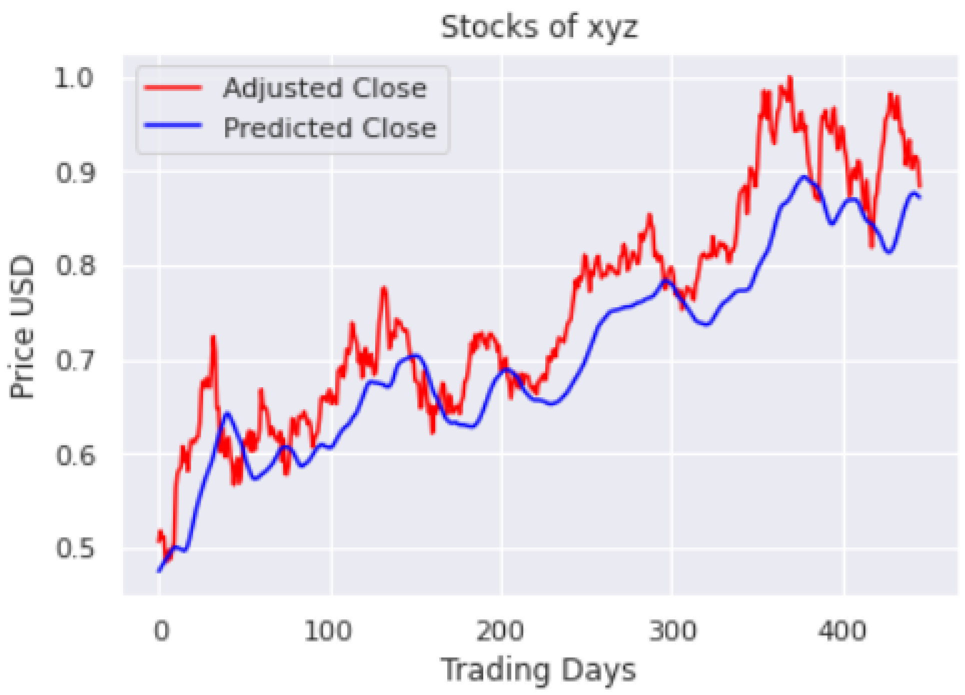
- Making batch size 100
- Changing number of epochs to 2

Train Score: 0.00012575 MSE (0.01121394 RMSE)
Test Score: 0.00420683 MSE (0.06486009 RMSE)



Improved LSTM:

Train Score: 0.0001434119330951944 MSE
Test Score: 0.0034942899364978075 MSE


Stocks of xyz
Adjusted Close
Predicted Close
Price USD
Trading Days
1.0
0.9
0.8
0.7
0.6
0.5
0
100
200
300
400