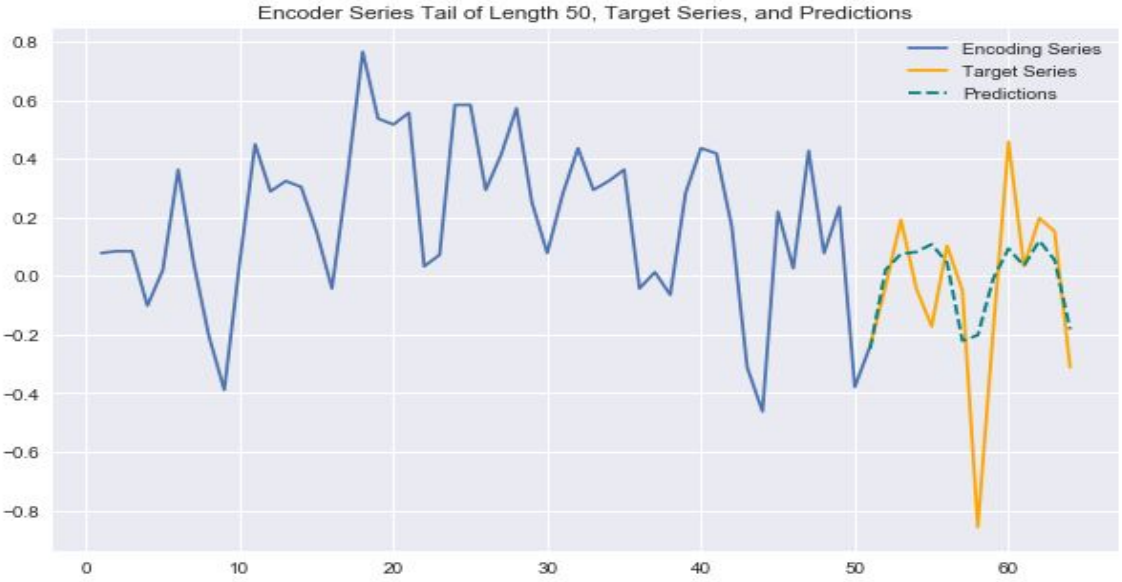
Sponsored by:

DATA SCIENCE INSTITUTE

The World Data Science Institute is a Specialized Consulting Agency offering DSaaS (Data Science as a Service)

Stock price forecasting model (For 7 days)



Data Science Team 1

Anade Davis - Data Science Manager - Linkedin

Tanjeel Ahmed - Data Science Researcher (Project Lead)

Christopher Rutherford - Data Scientist - LinkedIn

Berkalp Altay - Data Analyst - LinkedIn

Gabe Smithline - Quantitative Analyst- LinkedIn

Zain Dwiat - Data Scientist

Stock price forecasting model (For 7 days)

We are going to make a machine learning model that can predict stock prices for 7 days. As we are getting our data through an API and by analyzing those data (Historical data) we are going to predict the next 7 days stock price, we can categorize this machine learning model as a supervised machine learning model. This particular problem can also be defined as time series data prediction model as the data is changing chronologically. The data we are getting from the API is labelled data.

Stock price forecasting model (For 7 days)

The labels are

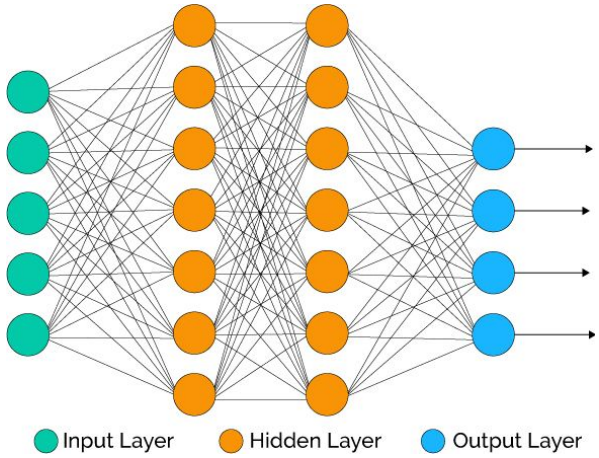
1. Open
2. High
3. Low
4. Close
5. Volume

We are going to predict the Closing price.

Index	1. open	2. high	3. low	4. close	5. volume
2020-04-29	173.2200	177.6800	171.8800	177.4300	51286559
2020-04-30	180.0000	180.4000	176.2300	179.2100	53875857
2020-05-01	175.8000	178.6400	174.0100	174.5700	39370474
2020-05-04	174.4900	179.0000	173.8000	178.8400	30372862

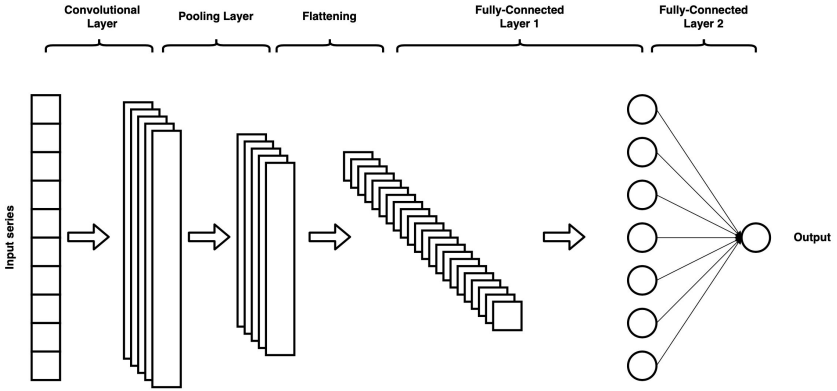
Stock price forecasting model (For 7 days)

There are many popular algorithms established in the market that can predict stock prices like MLP (Multilayer perceptions), RNN (Recursive neural networks), LSTM (Long short-term memory), CNN (Convolutional neural networks). For this project we are going to use CNN algorithm-based model.



CNN Model

CNN algorithm is a deep learning algorithm that was originally developed to process image data but recently it shows amazing results in predicting sequential data analysis, exactly the one we are doing in this project.



Overview of process

1. Getting the data through the alphavantage API
2. Preprocessing/cleaning the data
3. EDA (exploratory data analysis)
4. Creating features, preparing the data for the model
5. Setting up the model: (1) Construct the model layers (2) Fit the model with the stock data downloaded through the API
6. Predicting the stock price
7. Visualizing our prediction
8. Backtesting the model
9. Deploying the model

Obtaining and cleaning the data

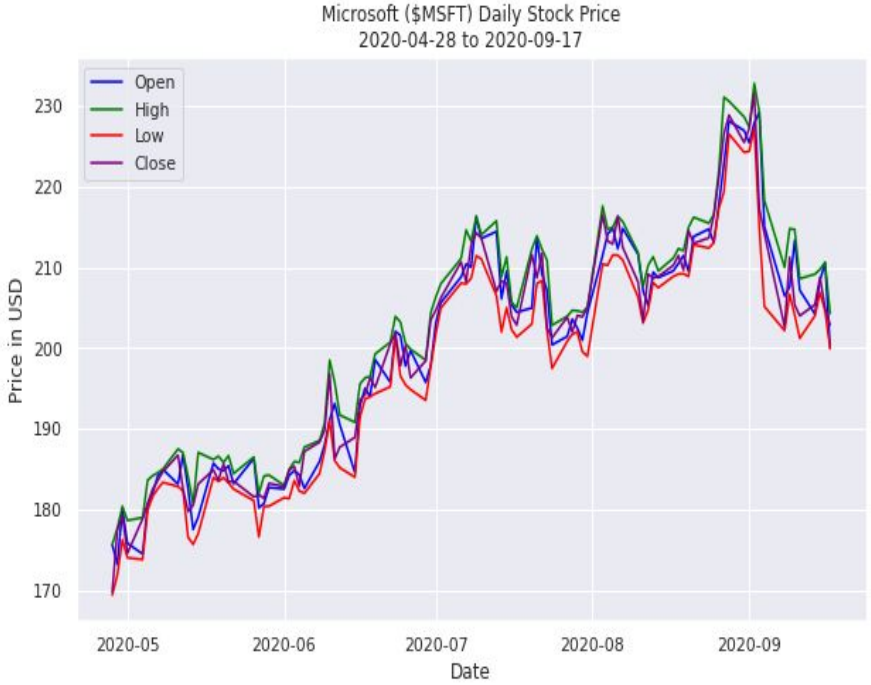
Data is imported using the alphavantage API as a JSON (JavaScript object notation) file

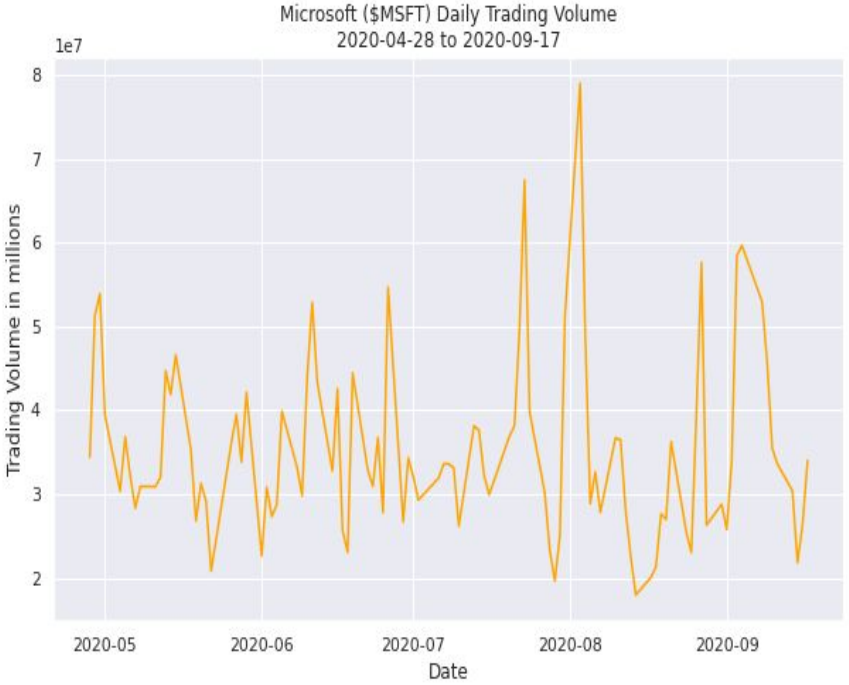
For this example and demonstration of the model, we will use Microsoft's historical stock data

Since the data is imported in the JSON format, it has to be cleaned and converted into a pandas dataframe to quickly and effectively perform analyses on it

```
import requests
API_KEY = '[redacted]
ticker="MSFT"
r = requests.get("https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol="+ticker+"&apikey="+API_KEY)
if (r.status_code == 200):
  print(r.json())
```

Visualizing the data



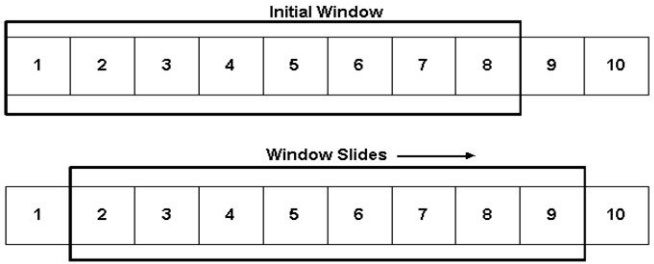


Preparing the data for the model

Before constructing the model, we must prepare the data for being input into the model. We will split our data into “windows” to help the model see any possible trends or patterns in the data.

Stock price data is a pretty unique type of time series data due to how volatile it is. The most recent data point is usually the best predictor of the next data point.

We create windows of length 5 to use as our X variable, with the following point in time being our Y. For example, the first window in X is the price for days 1-5, and its corresponding Y is the price on day 6.

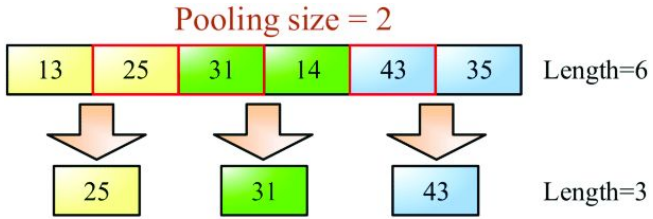


Putting Together The CNN: Convolutional Layers

The model setup begins by initializing the **Sequential** class, which is like a foundation that allows us to add the layers to our network.

The first layer is a **1D convolutional** layer to attempt to capture any patterns that may be present in the data - using more layers (in our case, 64) can help capture more of these patterns. A kernel size of 2 inputs 2 windows of data at a time.

The second layer, **MaxPooling1D** with a pool size of 2, takes the maximum value of each consecutive pair of data points.



Putting Together The CNN: Fully Connected Layers

The first fully connected layer is a **Flatten** layer, which converts (flattens) our input into a 1D vector. Even though our input data is 1 dimensional, running it through the convolutional layers usually alters the dimensions of our data.

Then we add a **Dense** layer with 50 units and the ReLU activation function

- The dense layer computes a dot product between inputs and weights (kernel). As such, the number of units specifies the output dimension of this dot product
- Values passed through the ReLU function remain unchanged if positive, but are changed to zero if they are negative

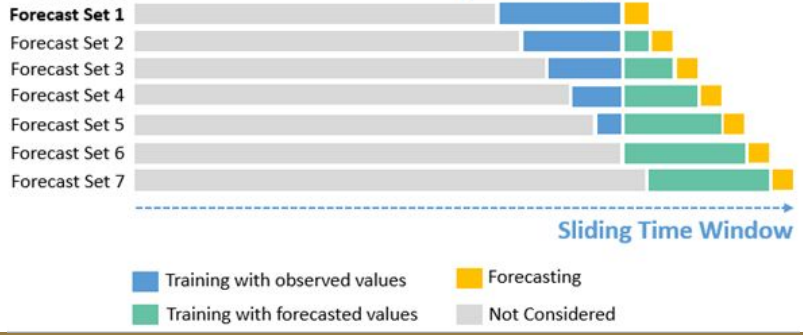
We add another **Dense** layer with 1 unit, which outputs a single value - the predicted price

Fitting the Model and Creating Predictions

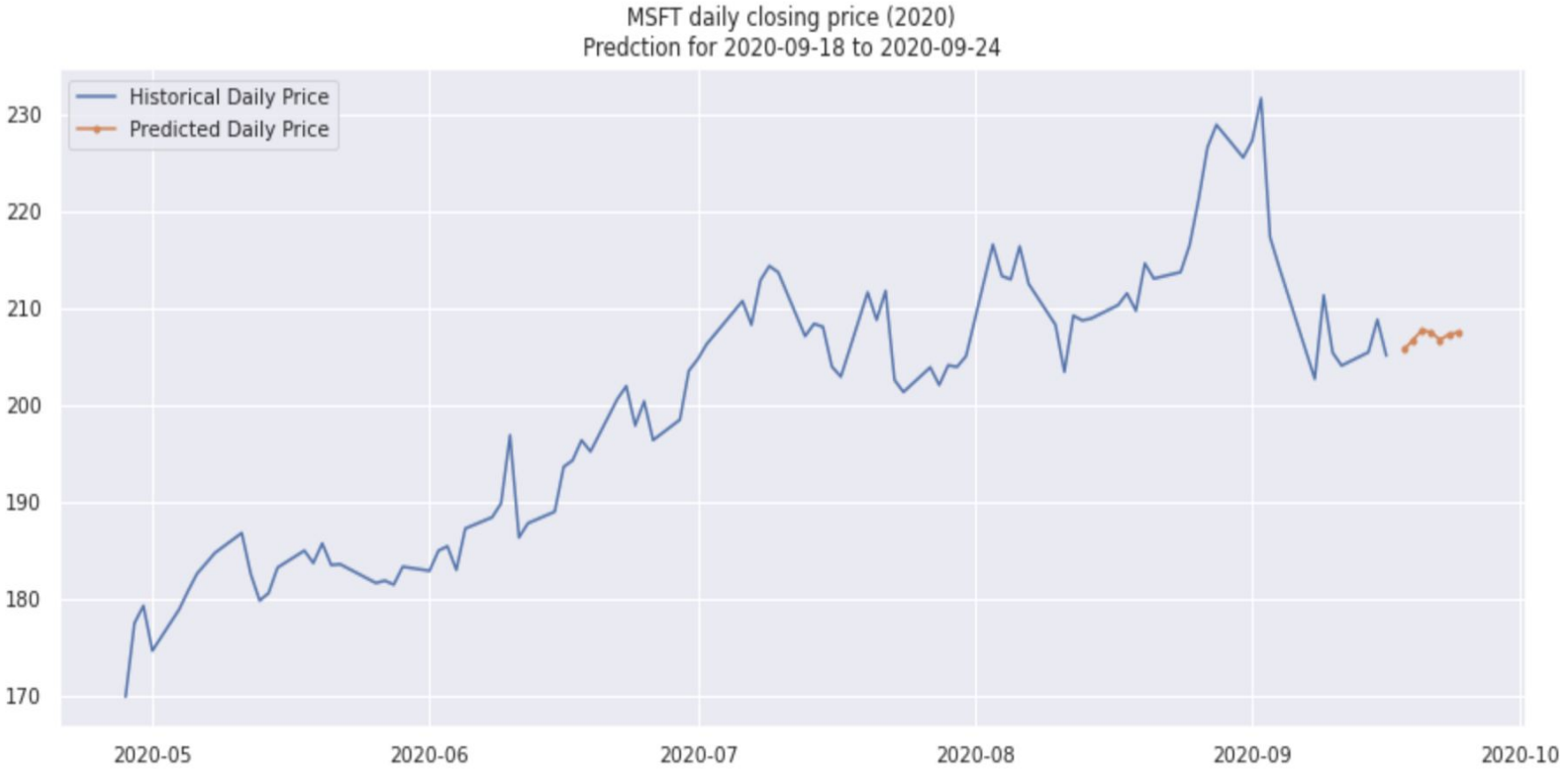
We train our CNN using the windows as our X and time following this windows as our Y. For instance, if the first X value is the window for days 1-5, then the corresponding Y value is the value for day 6. The second X value is the window for days 2-6 with the second Y being for day 7, and so on.

The model is trained over 100 epochs to obtain a better fit on the data. The model loss seems to level off after around 35 epochs. Error is measured using MSE (mean squared error)

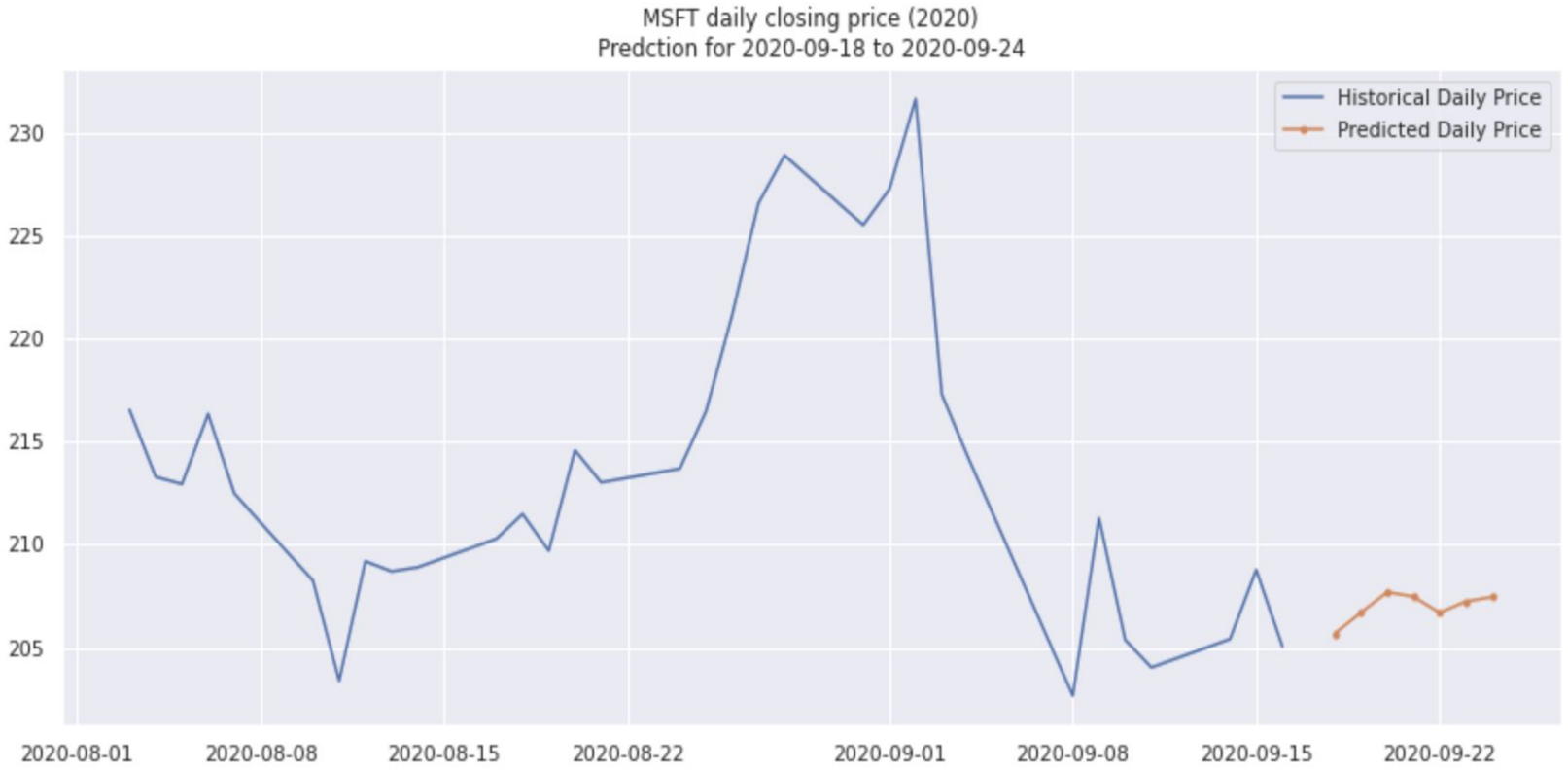
After training, we feed the most recent window (the last 5 data points) into our model for creating our predictions. This new future value, $\mathbf{y}_{t+1}$, will also be used for predicting the next values. That is, we use the last FOUR known data points and $\mathbf{y}_{t+1}$ to predict $\mathbf{y}_{t+2}$. We repeat this process until we have 7 days of prices predicted.



Interpreting Our Results



Interpreting Our Results



Interpreting Our Results

- In blue we see the historical price stock price and in orange we see the prediction for 7 days into the future.
- Since the shutdown because of COVID, Microsoft's stock has steadily risen over time. This is expected with everything moving to the cloud, as there is an even greater reliance on online services and products.
- However, Microsoft dropped from around $230 to just above $200 during the first week of September. Our model predicts a slight increase within the next week to about $207.

Interpreting Results Farther

- The first week of September Microsoft's stock fell 11% since peaking on September 2nd, experts predict it could go even lower
- However, shares are up by 35% YTD and are outpacing the S&P 500 by 6.1%
- The stock broke a technical uptrend on the chart which has supported its upward trend since April
- The breaking of this trend signifies possibly further fall
- If this support for upward trend breaks then Microsoft is expected to drop about 10% to a price of $188

Backtesting

- We had to do break up our data for backtesting in an untraditional way. This is because of the way the stocks behave; the price is too sporadic to predict from just an 80/20 train/test split. What we ended up doing is removing the last week's worth of data, running that dataframe through the model, then comparing the predicted values to the actual values.

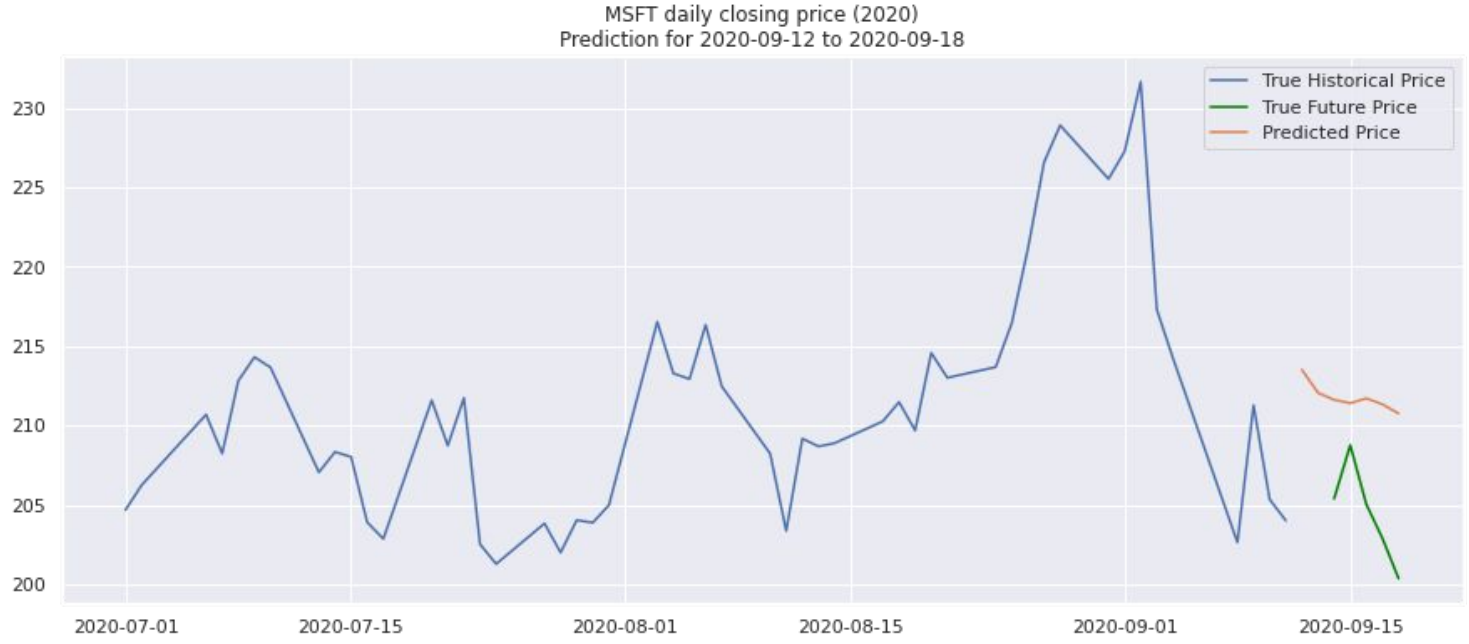
Backtesting

Our dataframe for backtesting is truncated to only have stock prices through September 11 - we want to predict the values for the next 7 days and immediately be able to compare them to the known values

The data is split in a similar manner to the full dataset (X with window of length 5, Y being the value after the window) and the neural network is identical

We run the data through the model to train it and create our predictions in an identical manner

Backtesting



Backtesting

The backtest verified that our model can at least make reasonable predictions about where a stock's price may be heading. The volatility of a stock's price can make it nearly impossible to make accurate forecasts using only the historical price

Rumors, earnings calls, news, the general health/outlook of the market, etc., all have a significant impact on a stock's price and can easily affect any possible trend the stock may be having

Before Deploying the model

The outline for deployment we developed was as follows:

- Step 1. User enters a ticker (e.g. AAPL) and presses "Predict".
- Step 2. The ticker is extracted.
- Step 3. Supply the ticker into the model and get the predicted results from it.
- Step 4. Return the results back to the user on the web.

Before Deploying the model

As per Step 3, the ticker would be input to the model and then return 7-day price prediction. Thus, we needed to encapsulate all the code for the model in a function called our_model.

```
def our_model(ticker, DAYS_TO_PREDICT):
```

This way, we could supply the ticker for each stock and make the model train using historical data for the stock and predict the 7-day stock prices.

Deploying the model

Flask

- The deployment of the model will be accomplished with using Flask
- Flask is a web framework for python - this is a pythonic way for developing Web applications that are written in python.
- It makes it very easy to establish your server using python and with that, it also makes it easier for the user to navigate through a page that you created with using flask. Whether a Get or Post request is being called, flask makes it really simple to organize the roll out your code onto a server.
- For the sake of what we are doing, flask came in handy because we could make a call directly to our machine learning model once the user inputs the required parameter, and the output can be viewed at ‘/Predict ‘

Flask + Ngrok

- Once our server was created with flask, we needed a method in order to make it accessible to users on the internet, and that was achieved with Ngrok.
- Ngrok makes it a lot easier to expose the server running on our machine and with a few lines of code since a library with a flask + ngrok has already been integrated.
- With these few lines of code below + an ngrok account - we were able to have ngrok listening to our server and hosting it onto the internet.

```
from flask_ngrok import run_with_ngrok
from flask import Flask, request
app = Flask(__name__)
run_with_ngrok(app)   #starts ngrok when the app is run
```

Flask + Ngrok

- Ngrok allows us to deploy a model online using a temporary webpage that is accessible as long as the code block that ran Ngrok continues to run. If you choose to use Ngrok's paid services, you can also deploy your model to a permanent webpage.

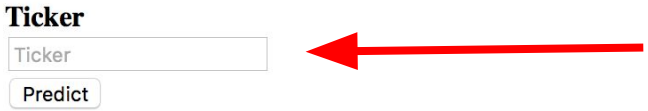
Accessing the temporary webpage by Ngrok

After running all the codes in the Google Colab, the last code block gives the following output:

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://cefeafc81f24.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

Clicking the URL indicated by the red arrow, we access the temporary webpage.

Webpage



We enter the ticker for a stock we want to get the 7-day prediction for. For example, let's enter AAPL and press "Predict" to get the 7-day stock price prediction for Apple Stocks.

Result on Webpage

An exemplary result table for AAPL is given on the right.

Ticker

Ticker

Predict

	Predicted Close Price
2020-09-18	111.824020
2020-09-19	111.032547
2020-09-20	111.316521
2020-09-21	110.624352
2020-09-22	110.710297
2020-09-23	110.562569
2020-09-24	110.670189

References

https://medium.com/@kshitijvijay271199/flask-on-google-colab-f6525986797b

https://www.youtube.com/watch?v=Pc8WdnIdXZg