



Time Series Analysis: **Predicting Stock Returns** with Recurrent Neural Networks (RNN)

Adrian Halarewicz
Springboard Capstone Project 2



Problem Statement:

Investing in the stock market would be easy if an investor could predict changes in stock price.

A stock's price can be influenced by many factors:

- economic trends
- related or unrelated companies
- expected returns

Is it possible to predict future changes in a stock's price exclusively from trends in a stock's own historical data?



The Client & Possible Implementations

A single investor with a particular interest in a single stock

The model will decide whether to buy, hold, or sell a particular stock.

The model can serve as the foundation of robotic trading or financial advising software and remove the need to manually manage a stock portfolio.



The Data Set

Historical stock data can be easily collected with the **yfinance** library.

The Recurrent Neural Network will utilize Long Short Term Memory (LSTM) to look for trends in a stock's **Adjusted Closing** price.

Exponential behaviour needs to be reduced to normalize changed in closing price.

The change (delta) between two consecutive days need to be calculated.

The resulting values need to be formatted for Time Series Analysis.



Downloading data with **yfinance**

```
import yfinance as yf

import pandas as pd

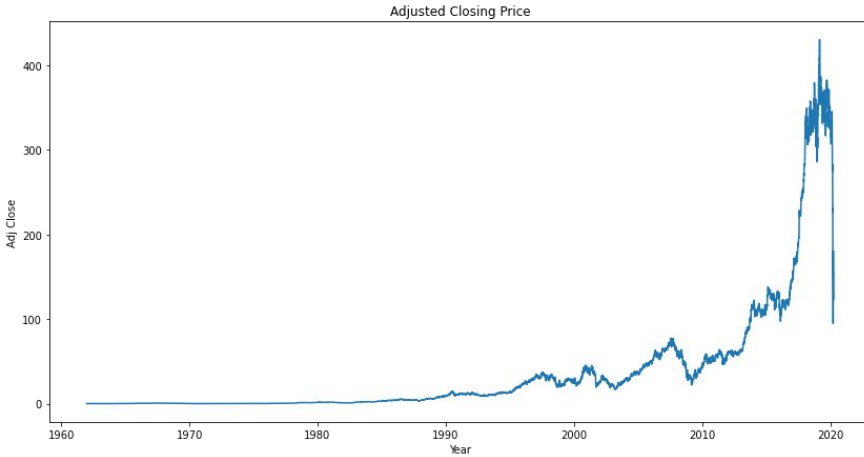
from datetime import datetime

today = datetime.date(datetime.now())

df = yf.download("GOOG", start="2004-08-19", end= today)

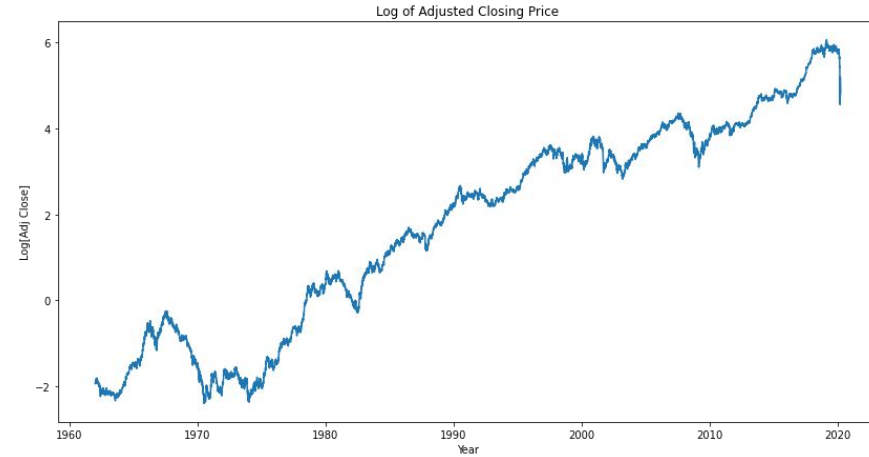
type(df) → pandas.DataFrame
```

Accounting for Exponential Behaviour



`log_scaled_df`

Changes in recent years will dominate changes when price was lower.



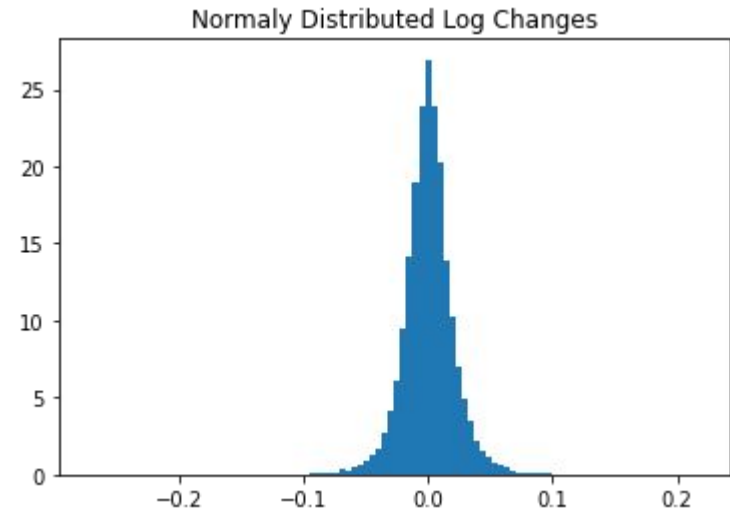
```
log_scaled_df =  
np.log(adj_close_df)
```

Obtaining a Normal Distribution

The model will predict the log of expected returns

$$\text{delta}_i = \text{Log}_e \left(\frac{\text{price}_{i+1}}{\text{price}_i} \right)$$

This results in Normally Distributed Data





Formatting for Time Series Analysis

DateTime Index	back_5 (predictor)	back_4 (predictor)	back_3 (predictor)	back_2 (predictor)	back_1 (predictor)	Log[delta(Adj Close)] (target)
<i>date</i>	delta_i-5	delta_i-4	delta_i-3	delta_i-2	delta_i-1	delta_i

Columns are created from previous target values.

LSTM remembers the returns from the 5 previous trading days

The RNN layers identify internal fluctuation trends.

The complete data set

	back_5	back_4	back_3	back_2	back_1	Adj Close	
Date							
1966-07-05	NaN	NaN	NaN	NaN	NaN	0.052056	drop row
1966-07-06	NaN	NaN	NaN	NaN	0.052056	-0.036905	drop row
1966-07-07	NaN	NaN	NaN	0.052056	-0.036905	0.011215	drop row
1966-07-08	NaN	NaN	0.052056	-0.036905	0.011215	-0.003724	drop row
1966-07-11	NaN	0.052056	-0.036905	0.011215	-0.003724	-0.015037	drop row
...	train/test
2020-03-27	-0.079807	0.166577	0.006340	0.026460	-0.020160	0.024810	train/test
2020-03-30	0.166577	0.006340	0.026460	-0.020160	0.024810	-0.016673	train/test
2020-03-31	0.006340	0.026460	-0.020160	0.024810	-0.016673	-0.044394	train/test
2020-04-01	0.026460	-0.020160	0.024810	-0.016673	-0.044394	0.020835	train/test
2020-04-02	-0.020160	0.024810	-0.016673	-0.044394	0.020835	NaN	prediction

Incomplete rows with
insufficient prior values

Rows for training and
model evaluation

Predict tomorrow's returns!



Interpreting Predictions

Predicted Return

$$e^{prediction} = \frac{Price_{today}}{Price_{tomorrow}}$$

Predicted Closing Price

$$Price_{tomorrow} = Price_{today} * e^{prediction}$$

$$e^{prediction} > 1.0 \Rightarrow + \text{positive returns expected}$$

$$e^{prediction} = 1.0 \Rightarrow Price_{tomorrow} = Price_{today}$$

$$e^{prediction} < 1.0 \Rightarrow - \text{negative returns expected}$$



Model Architecture (part 1)

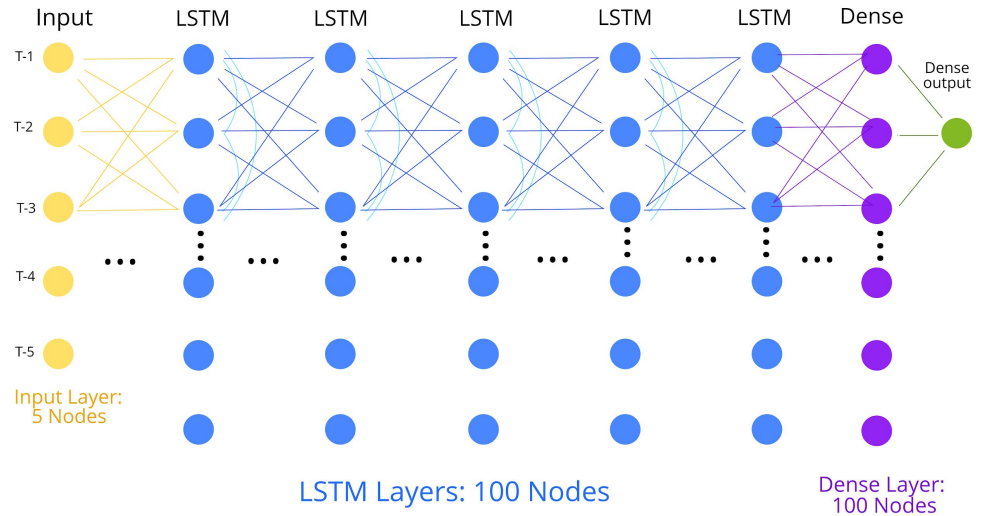
Optimized with Cross-Validation

Input Layer:	5 nodes
LSTM Layers:	4 Layers, 100 Nodes
Dense Layers:	1 Layer, 100 Nodes
Dense Output Layer:	1 Node
Nodes in each Layer:	25
Training Epochs:	2
Optimizer:	“adam”
Loss function:	“mean_squared_error”
Mean accuracy of 10 samples:	53.23%

Model Architecture (part 2)

Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
lstm_1 (LSTM)	(None, 1, 100)	42400
lstm_2 (LSTM)	(None, 1, 100)	80400
lstm_3 (LSTM)	(None, 1, 100)	80400
lstm_4 (LSTM)	(None, 100)	80400
dense_1 (Dense)	(None, 100)	10100
dense_2 (Dense)	(None, 1)	101
=====		
Total params: 293,801		
Trainable params: 293,801		
Non-trainable params: 0		





Significance of Accuracy

Results greater than 50.0% are acceptable.

It is extremely rare for a stock predictor to exceed 60%

```
Predicting change in stock price with 51.362398% accuracy
```

Any slight advantage can be scaled to make tremendous profit.

Consider the wealth Las Vegas casinos generate from games where the house win probability is only ~52%.

Bootstrapping: True Mean of Accuracy

Sequential Data split at random index.

→ test size: ranges from 5% to 30%

Eliminate influence of split on model performance.

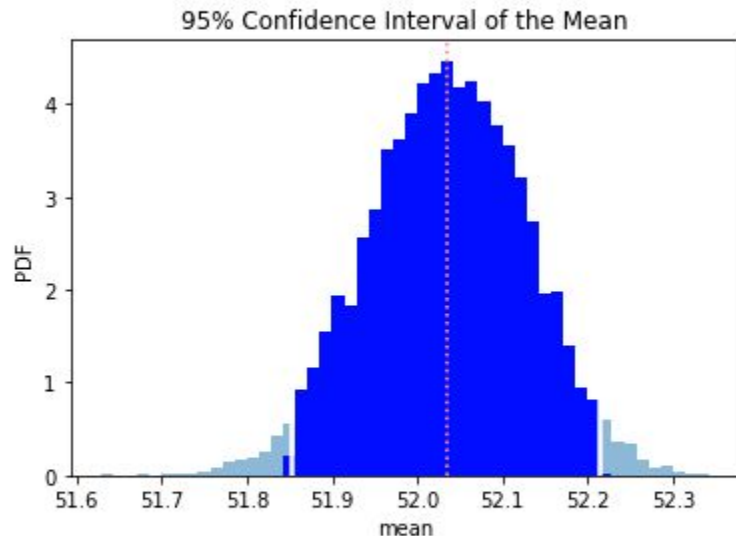
Build sample from many models:

→ each model trained on a data with a unique split

Accuracies of the models in the sample are Bootstrapped.

```
1 bootstrapping = bootstrap_mean(accuracy_samples)
```

52.035 52.035



```
1 replicates, int_min, int_max = bootstrapping  
2 np.mean(replicates)
```

52.03477468541332

Making Predictions: Evaluating against Test Data

Blue line:

- Actual stock price

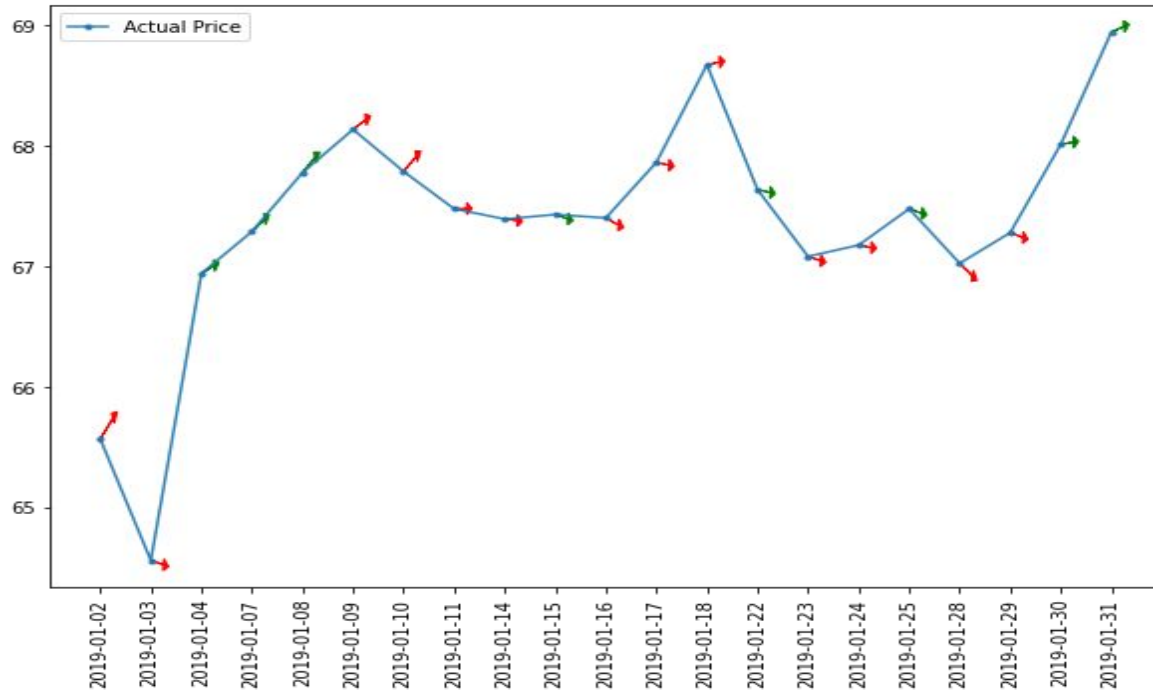
Arrow Direction:

- Predicted direction of change

Arrow Color:

- **Green:** Correct prediction

- **Red:** Incorrect prediction



Making Predictions: Predicting tomorrow

Epoch 1/2

9595/9595 [=====] - 5s 503us/step -

loss: 0.0196

Epoch 2/2

9595/9595 [=====] - 2s 234us/step -

loss: 0.0018

Prediction Quality:

min pred: 0.9908

max pred: 1.007

mean pred: 1.0009

median pred: 1.001

percentile(1.0): 99.53

Predicting change in stock price with 51.362398% accuracy

Date	price_today	price_tomorrow	Action
2020-04-15	90.79	90.93	BUY

back_5	back_4	back_3	back_2	back_1	Adj Close	
NaN	NaN	NaN	NaN	NaN	0.052056	drop row
NaN	NaN	NaN	NaN	0.052056	-0.036905	drop row
NaN	NaN	NaN	0.052056	-0.036905	0.011215	drop row
NaN	NaN	0.052056	-0.036905	0.011215	-0.003724	drop row
NaN	0.052056	-0.036905	0.011215	-0.003724	-0.015037	drop row
...	train/test
-0.079807	0.166577	0.006340	0.026460	-0.020160	0.024810	train/test
0.166577	0.006340	0.026460	-0.020160	0.024810	-0.016673	train/test
0.006340	0.026460	-0.020160	0.024810	-0.016673	-0.044394	train/test
0.026460	-0.020160	0.024810	-0.016673	-0.044394	0.020835	train/test
-0.020160	0.024810	-0.016673	-0.044394	0.020835	NaN	prediction



Thank you

<https://ahalarewicz.github.io/TimeSeriesAnalysis/>

<https://github.com/AHalarewicz/TimeSeriesAnalysis>