# CS419 Project on Stock Price Prediction

Arjun Agrawal|150110015
Jay Narale|150110014
Amogha HA|150040102

# Data Used

- We have used the past stock price-volume data for some of the equities traded. This data includes the adjusted open price, close price, high, low, volume traded, split and dividend data. All these have been inputted as a CSV file to our code.
- Adjusted prices are prices amended to include any distributions and corporate actions such as stock splits (splitting one stock into two which would halve the price), dividends (giving stockholders cash as a fraction of profits) that occurred at any time before the next day's open.

# Characteristics of the data

- Time-series data.
- Noisy data
- Datapoints (prices of different stocks) are not independent of each other -> Naive Bayes is not appropriate
- Many features. (Daily open, high, low, adjusted close for many stocks) ->
- Regression problem (**continuous output**).
- Training cost or time: it is not critical to keep this lower than 12 hours because we are predicting daily prices based

# Methodology Adopted

1. Time-series train-test split
    - We will train our model on what we'll call the training set, a subset of the data that we have.
    - To make sure our model generalises, we need to test it on some data it has not seen before and evaluate how well it does predicting on that data.
    - To do this, we need to set aside data for testing our model - the test set.
    - Because our data is time series data (there is some ordering to it and the ordering influences prices), we cannot shuffle the data.
    - If the data were shuffled, e.g. the adjusted close price for 1 Nov 2015 might be in the training set. We might then be asked to predict the adjusted close prices for the 7 days after 31 Oct 2015, which would include the price for 1 Nov 2015 which we'd have seen before.
    - So we cannot use sklearn's `train_test_split` function which automatically shuffles the data. Instead, we have written our own function.

## Time-series cross-validation

- But testing on only one test set and training on only one training set isn't robust enough. What if the test or training sets we choose have special characteristics that aren't common to other datasets?
- To make our evaluation more robust so we choose the best model, it's better if we can run multiple train-test cycles.
- To do this, we wrote the function `execute()`. In this function, we set a number of train-test cycles (`steps`), a total length of the train-test data (`periods` data-points) and a number of datapoints between the starting points of each consecutive train-test cycle (`buffer_step`).

We performed the following steps:

1) Import data (CSV) and format it as a Pandas Dataframe
2) Create features dataframe: Select features we wanted to use and put it into a separate data-frame
3) Create target dataframe (Prices for 7 days following the last date provided in the features).
4) Split into training and testing sets. (**No shuffle because we are dealing with time series data**.)
5) Train chosen classifier.
6) Predict test target.
7) Evaluate test target and print evaluation metrics.

# Results

| Days after last training date | Mean Root mean squared daily percentage error (across 8 distinct train-test sets) |
|---|---|
| 1 | 1.306 |
| 2 | 1.954 |
| 3 | 2.354 |
| 4 | 2.747 |
| 5 | 3.039 |
| 6 | 3.267 |
| 7 | 3.434 |

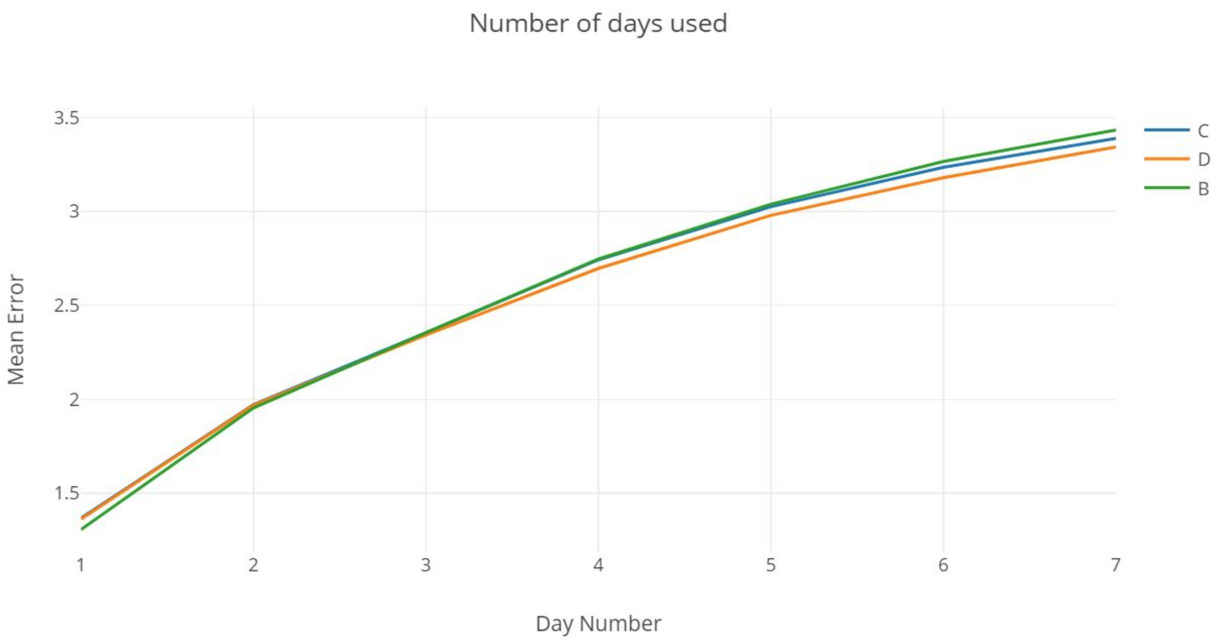
Mean R2 score: 0.838.

| Days after last training date | Mean Root mean squared daily percentage error (across 8 distinct train-test sets) |
|---|---|
| 1 | 26.317 |
| 2 | 26.391 |
| 3 | 26.455 |
| 4 | 26.495 |
| 5 | 26.537 |
| 6 | 26.591 |
| 7 | 26.693 |

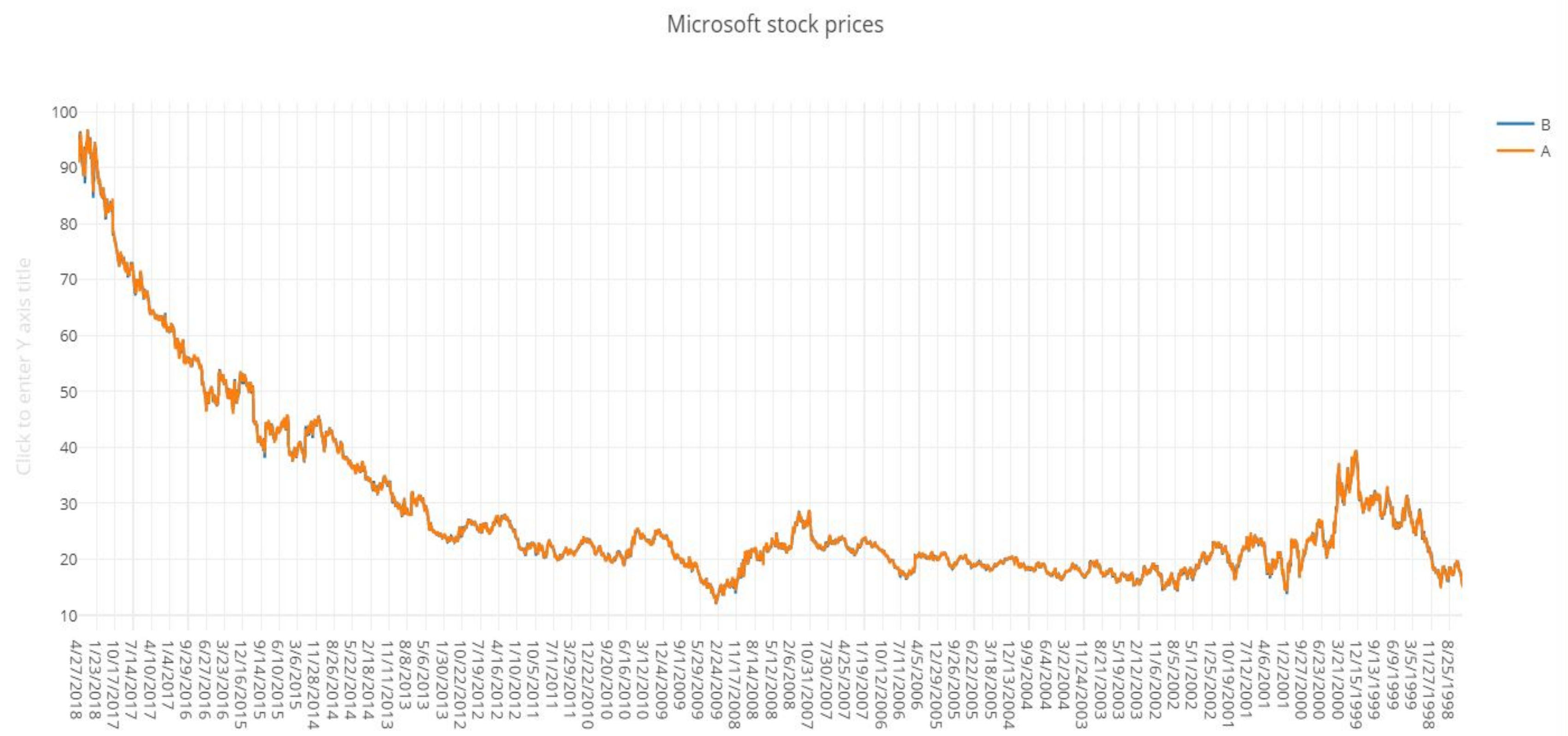
**SVM.SVR**

Mean R2 score: -11.83

| Day to predict | 7d (used) | 10d | 14d | 21d | 30d | 100d |
|---|---|---|---|---|---|---|
| 1 | 1.306 | 1.366 | 1.361 | 1.1584 | 1.162 | 1.924 |
| 2 | 1.954 | 1.971 | 1.969 | 1.639 | 1.676 | 2.768 |
| 3 | 2.354 | 2.355 | 2.343 | 1.904 | 1.951 | 3.370 |
| 4 | 2.747 | 2.743 | 2.697 | 2.193 | 2.214 | 3.890 |
| 5 | 3.039 | 3.026 | 2.980 | 2.434 | 2.480 | 4.355 |
| 6 | 3.267 | 4.236 | 3.180 | 2.646 | 2.706 | 4.769 |
| 7 | 3.434 | 3.390 | 3.344 | 2.837 | 2.913 | 5.163 |



Number of days used
Mean Error
3.5
3
2.5
2
1.5
1
2
3
4
5
6
7
Day Number
C
D
B


Microsoft stock prices
B
A
Click to enter Y axis title
100
90
80
70
60
50
40
30
20
10
4/27/2018
1/23/2018
10/17/2017
7/14/2017
4/10/2017
1/4/2017
9/29/2016
6/27/2016
3/23/2016
12/16/2015
9/14/2015
6/10/2015
3/6/2015
11/28/2014
8/26/2014
5/22/2014
2/18/2014
11/11/2013
8/8/2013
5/6/2013
1/30/2013
10/22/2012
7/19/2012
4/16/2012
1/10/2012
10/5/2011
7/1/2011
3/29/2011
12/22/2010
9/20/2010
6/16/2010
3/12/2010
12/4/2009
9/1/2009
5/29/2009
2/24/2009
11/17/2008
8/14/2008
5/12/2008
2/6/2008
10/31/2007
7/30/2007
4/25/2007
1/19/2007
10/12/2006
7/11/2006
4/5/2006
12/29/2005
9/26/2005
6/22/2005
3/18/2005
12/13/2004
9/9/2004
6/4/2004
3/2/2004
11/24/2003
8/21/2003
5/19/2003
2/12/2003
11/6/2002
8/5/2002
5/1/2002
1/25/2002
10/19/2001
7/12/2001
4/6/2001
1/2/2001
9/27/2000
6/23/2000
3/21/2000
12/15/1999
9/13/1999
6/9/1999
3/5/1999
11/27/1998
8/25/1998

# Challenges Faced

1. The model has to be run for dates not within the training set for the model to be 'fair'. But given there may be big shifts in how people view the markets from year to year, it may be hard for the model to generalise from one year to the next.
2. Some of the companies' stock prices are volatile so they may be harder to predict.