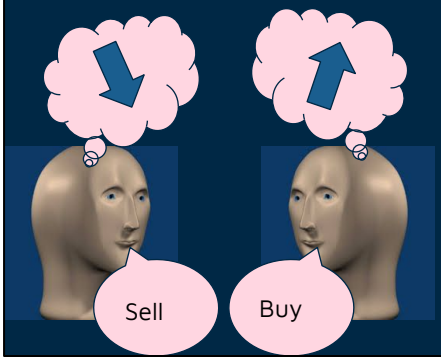
Hello
I'm Matthew Andrews
And today I'm talking about Predicting Stocks' Hidden Factors

Predicting Stock Prices

First Stock Exchange: Amsterdam, 1602

Competing philosophies:
Fundamental analysis: Study the company vs Technical analysis: Study the past

Arguably Impossible:
Burton Malkiel: A Random Walk Down Wall Street



Trade as disagreement,
Any tool used to gain insight

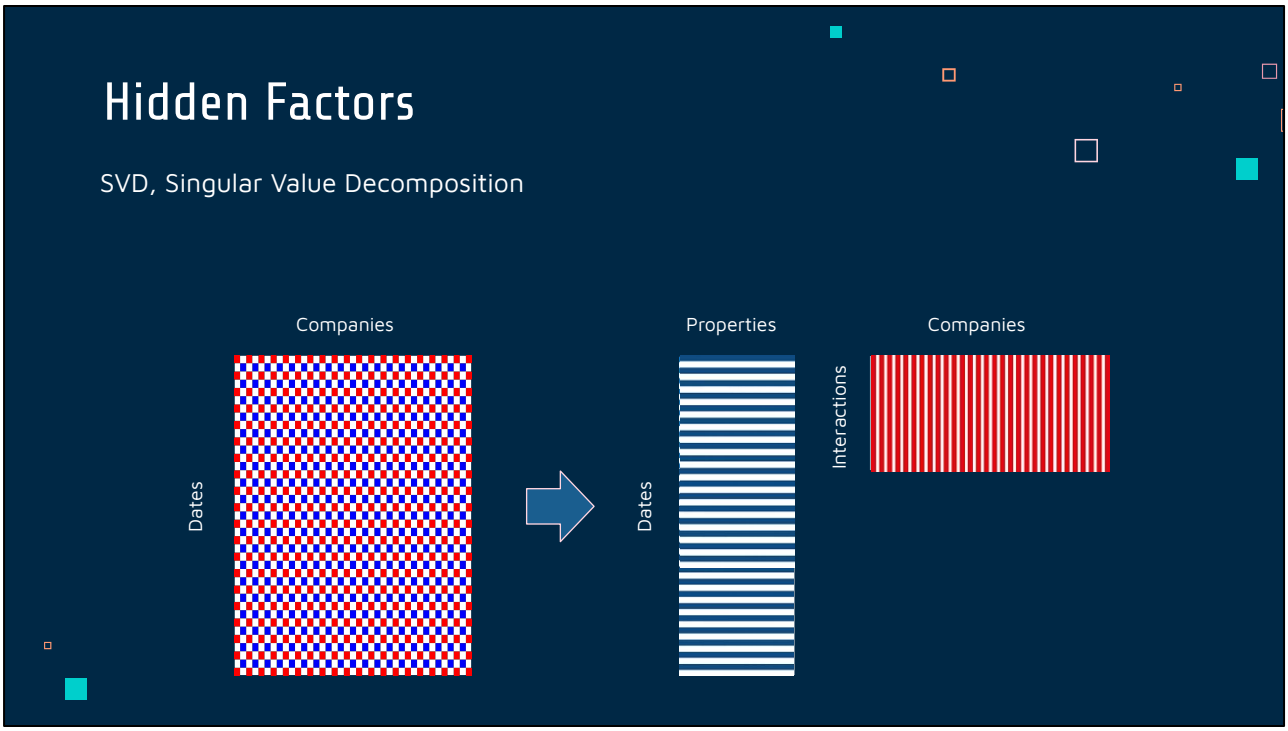
.

Neural Networks

Predicting stock prices is a tradition that goes back centuries
There have been competing philosophies over the years like: Need to study the company, or all necessary data is in past values
There is even debate over whether it is fundamentally possible
And yet, it is still a very profitable industry

How?
For the most part a trade can be considered a disagreement in a stock's future.
Hence there is an aspect to it of knowledge and the belief that you know something the other person doesn't.
And so the latest innovations in maths, science and technology are utilised in the attempt to get an edge
Bringing us to today's use of Neural Networks

But first, what do I mean by hidden factors?



For my final project in this Data SCIENCE course I've done an experiment with data and data science methods.
SVD, Singular Value Decomposition, is a procedure which decomposes 1 matrix into 2 smaller matrices
For example, a table of stock values with the companies along the top and dates on the side, can be split into:
1 table with dates on the side and, say, 100 columns and 1 table with the companies along the top and 100 rows
Such that, through matrix multiplication, they can recombine to create a table as close as possible to the original.

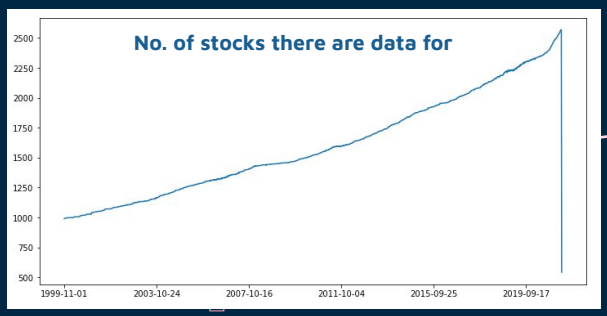
But why?
Well now you have each date described by 100 values, every day has these 100 properties
(and each company is described by their 100 values -or how they interact with the properties of each day)

Stock prices are extremely chaotic. But surely they aren't entirely disconnected from the world?
And there are things in the world, which should affect stock prices, that have predictable patterns, even if it's just the world going around the sun
I am trying to find some predictability, certainly not in all, but hopefully in some of these factors which Singular Value Decomposition exposes to us

But to start, I need the first matrix, the table of stock values

Data

Acquired using Alpha Vantage API



Data gathered

2570 stocks, from NYSE

Data used

2169 stocks

From 1999 to present

2004 -2018

Engineered Features:
midpoint value, change in midpoint value

I used the Alpha Vantage API to get data for over 2500 stocks, on the New York Stock Exchange, over the last 20 years

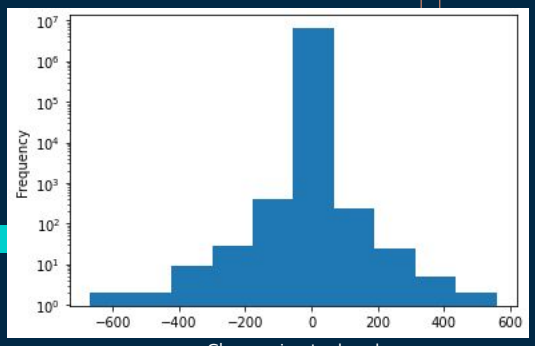
1 issue I found was, as shown in the graph, of course,not all companies on the New York Stock Exchange have been there since 1999
Therefore I changed the range I was looking at to 2004 -2018, which gave me over 2100 stocks
As a professional API, providing professional data, the data was reliable and reliably formatted

Now the API had a limit of 500 stocks per day so it took about a week to get all the data
That's also why the graph drops off suddenly, as, for the last day I gathered data on, there can't be more than 500 stocks

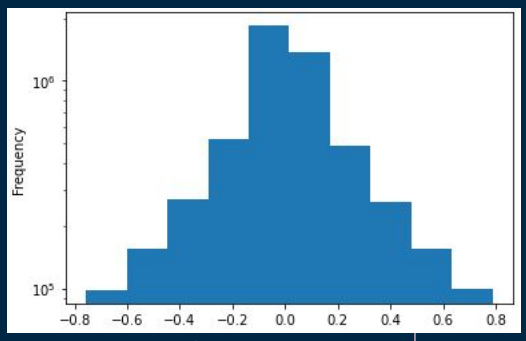
I used this data to find the change in each stock's midpoint value for each day

SVD

Input With Outliers



Input Without Outliers



RMSE: 0.194 (between actual Stocks x Days table and the table made when the resultant tables are recombined)

Actual Std: 0.250 Recomposed Std: 0.135

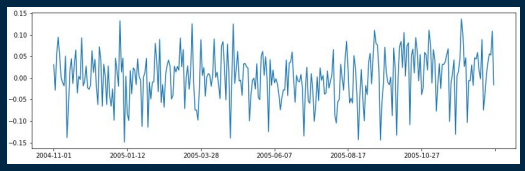
Once I'd gathered the data and processed it, I performed the Singular Value Decomposition on it
At first I found the outliers were broadening the range of values too far
So I removed the highest and lowest 5% of values from the SVD input,
On the graphs here, do note that the y-scale is logarithmic

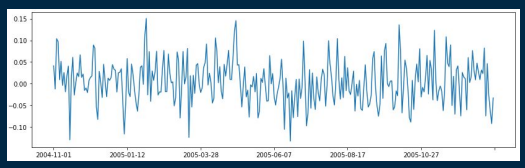
Between the original Stocks by Dates table and the table made when the resultant separate tables are recombined, there's an Root Mean Squared Error of 0.194
This isn't ideal when compared to a standard deviation of 0.25 -i.e. if every cell in the new table was the average value, the error would be 0.25
Note also that the standard deviation of the recomposed table is roughly half that of the original
This is common in stock prediction, there's often a smoothing as larger changes are less likely and so are rarely predicted

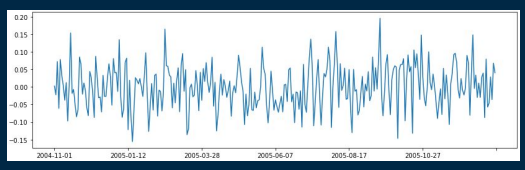
I also tried this using the stocks' relative changes instead of absolute, but the results weren't as good

EDA

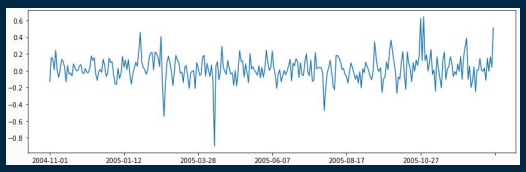
Day Factors

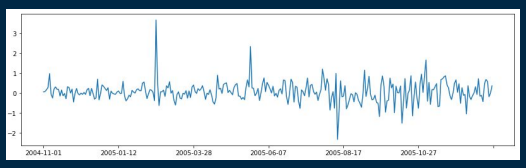


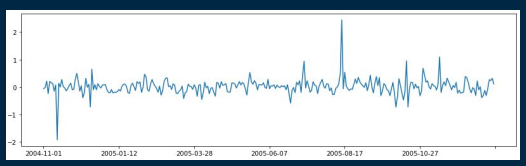




Stock Changes

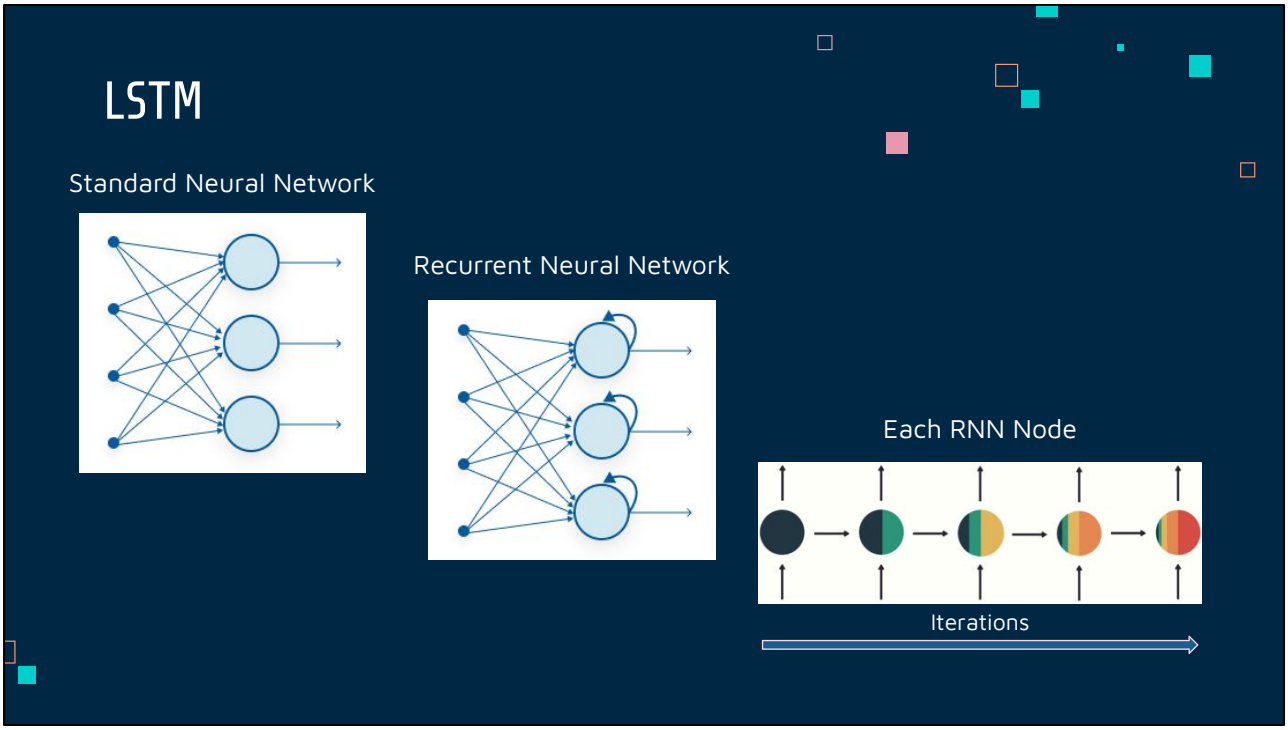






After performing the SVD, I had the table of the dates' factors
Here's a look at, just a few, of them over time, on the left; as well as some of the stock changes over time on the right

Not too much to glean but-
The first graph looks like it could have some seasonality, lending it some predictability
Whereas the graphs on the right do seem more chaotic



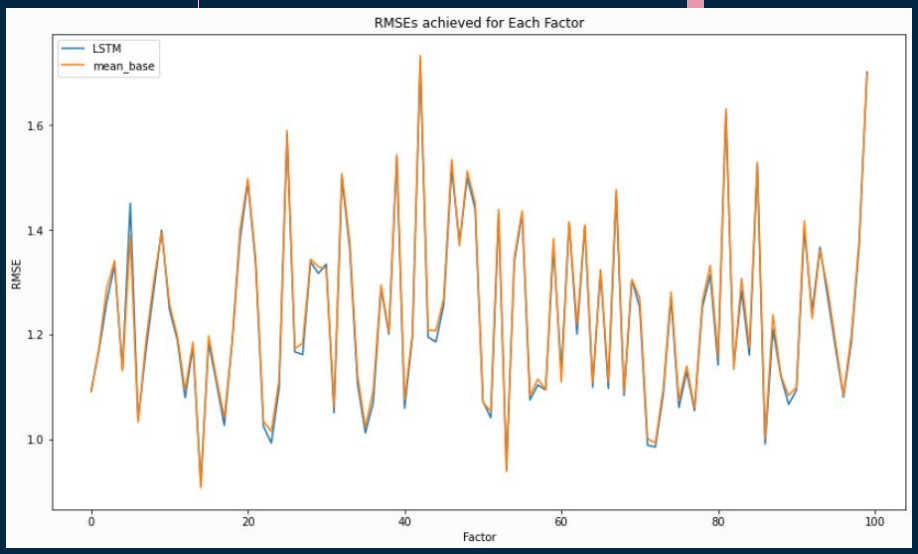
Having had a look at the Day Factors I now wanted to predict the value of each factor for each day in the testset
To accomplish this I decided to use a type of neural network called an LSTM which stands for Long Short Term Memory
Your run-of-the-mill neural network is made up of layers of nodes
Every node takes in the output from every node in the previous layer, performs its unique calculation, and sends its output to every node in the next layer
The calculation each node performs is then refined and it tries again

But an LSTM is slightly different. LSTMs belong to a family called RNNs -Recurrent Neural Networks
What's special about an RNN is that each node has an extra input -its own previous output
And as its output is a function of its previous output, and thence its previous input, its output MUST be a function of ALL its previous outputs
However this does have some drawbacks: Although previous outputs ARE included in each input, it's still heavily weighted towards the most recent
The LSTM combats this by being able to forget previous outputs that weren't significant.

This makes them the go-to Neural Network for working with time series.

My next task was to train 100 of them, one for each factor.
This may be a lot, but doing it for each stock separately would be over 2500

Results

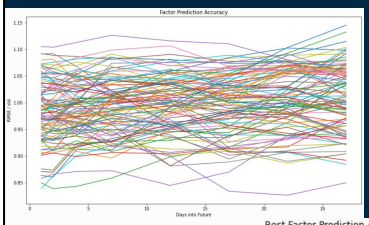


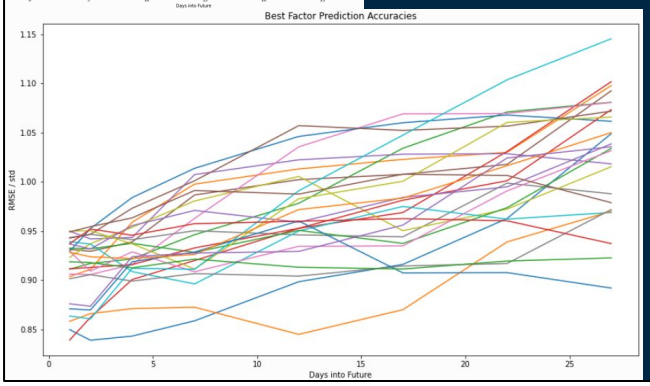
This graph shows the Root Mean Squared Errors achieved for each factor by both the LSTM model and a mean-baseline.
Firstly, although the data may fit more readily to being presented in a bar chart,
With this many points it's actually much clearer as a line graph
We can see that in most cases the blue line is below the orange, showing that the model performs better, though only very slightly, than the mean-baseline:
The mean-baseline is the result if we used the average of the previous 90 days as the prediction

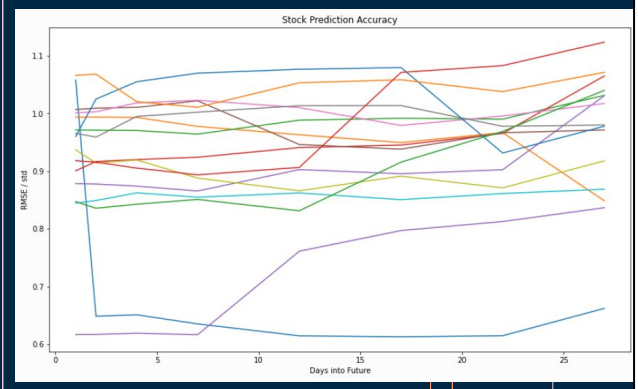
After making a prediction for a single day ahead I went on to look further into the future

Comparison

Method 1: Divide RMSEs by standard deviations







However I also needed something to compare this hidden factor method against:
a quick google returned many articles where an LSTM is trained directly on the stock values
Though doing all 2500 would be far too much, I chose 15 stocks and performed the prediction on them

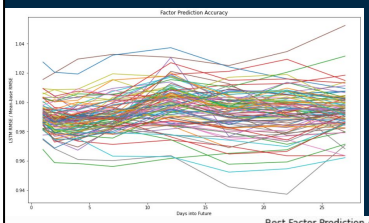
When you calculate Root Mean Squared Error the result is of the same scale as the original data:
Not only can't you compare results between the stocks and the factors,
you can't compare results between different stocks or between different factors

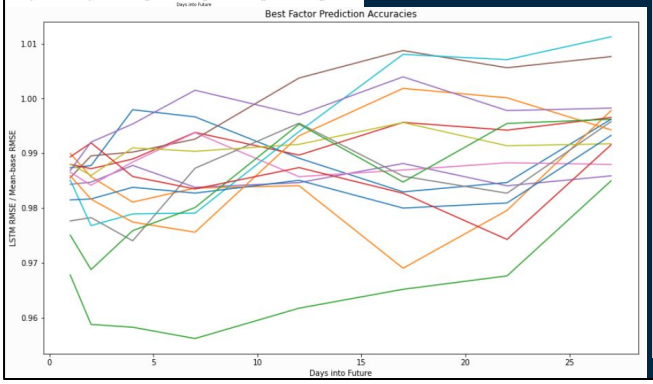
So I'll present a couple of methods I used to compare them:
Firstly, dividing each stock and factor's RMSE by its standard deviation.
In the top left are the LSTM results for all factors but,as it's a bit crowded, below it are the 26 most accurately predicted factors
Again, the lower on the graph it is the less error there is and so it is more accurate
As expected, the further into the future we try to predict, the accuracy decreases

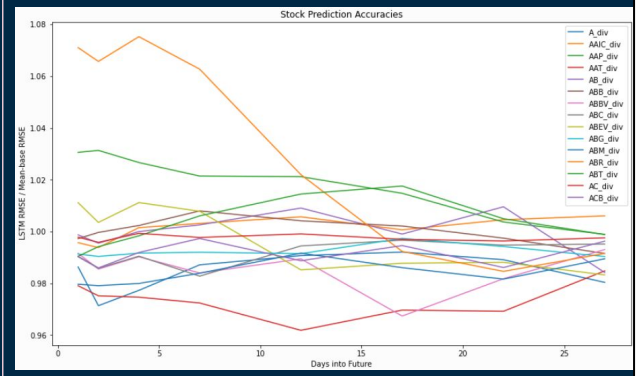
The right graph shows the results from predicting the stocks directly
Although some seem to be predicted quite well, the blue stock at the bottom shows that outliers can increase the standard deviation enough to improve the result undeservedly

Comparison

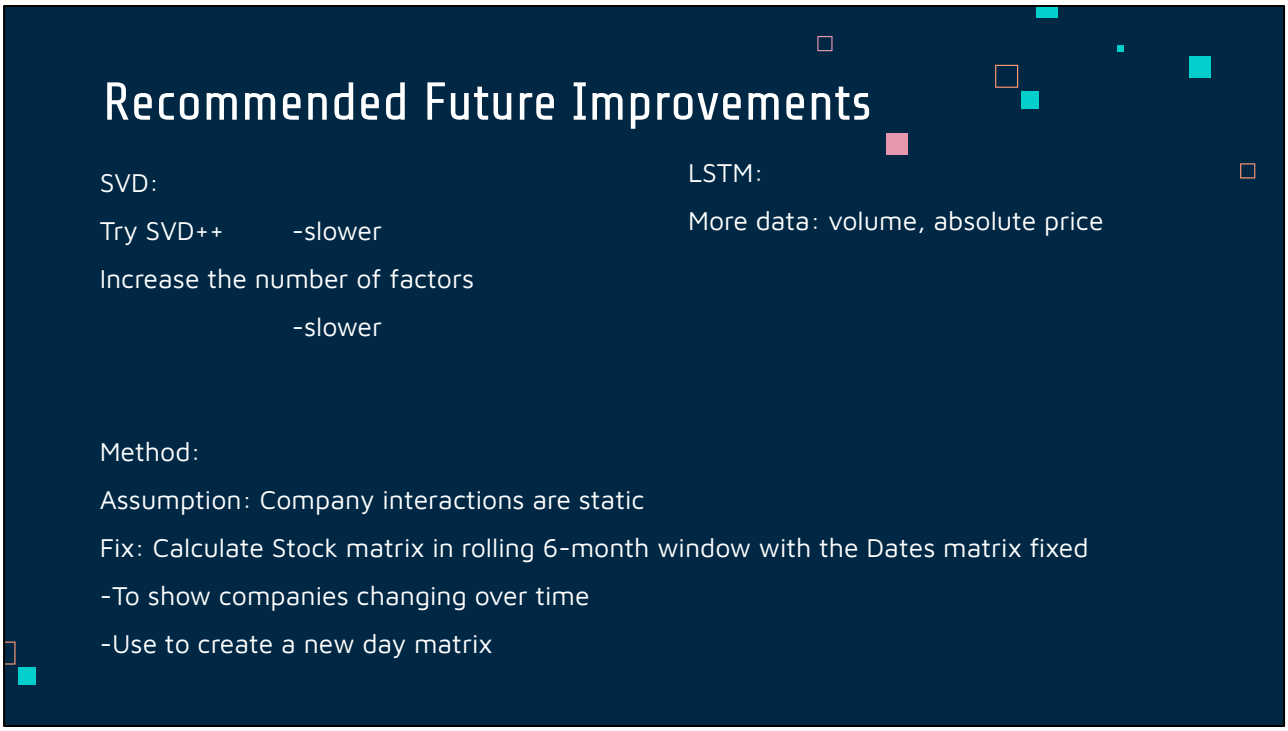
Method : Divide LSTM RMSEs by mean-base RMSEs







So I also compared how well the model performed against the mean-baseline
In the top left we can see, at least, that most of the results are below 1 -meaning that the model performed better than the baseline
However, although some of the factor predictions performed better than the stock predictions:
-the benefit gained using this method is only slight



Recommended Future Improvements:
I think the first area to make improvements is with the Singular Value Decomposition
For one thing I'd like to try a variant of the SVD algorithm called SVD++, I didn't use it here, though, as it can take much longer to run than SVD
The SVD could also be improved by adding more factors, but again each factor would be another LSTM to run and make the process slower

Changes could also be made to the LSTM like including more data such as the volume and the absolute price

But I think there's a more fundamental problem:
My method makes an inherent assumption that the way companies interact with these Date-properties is static,
-Whereas that is unlikely to be the case: companies change over time
But fortunately, they tend to change slowly
And so I suggest calculating the stock matrix with a 6-month rolling window, but keeping the dates matrix fixed from the SVD for all days.
This will show you how the companies change over time -How they react differently to the same stimuli
This can go toward predicting how the companies will change and also fed back, used to produce a better day matrix

Thank you for listening
I've been Matthew Andrews
Do you have any Questions?