# Forecasting Stock Returns from Historical Price Data

Christopher C. Smith

3/3/2022

## Introduction

In this paper, prepared as a capstone project for the HarvardX "Data Science Professional Certificate" series on <edx.org>, I will build a forecasting model for stock market returns using historical price data. I will limit our analysis to US "large cap" stocks belonging to the S&P 500 stock index, an index of five hundred of the US's largest publicly traded companies. To mitigate survivorship bias, I attempt to include price data not only for companies currently in the index, but also for companies that have been removed from the index. This proves difficult, however, due to the limited availability of data on these companies. For purposes of this analysis, I only forecast prices for a stock during the period(s) of time when it is part of the S&P 500 index. I do not attempt to forecast prices before or after its admission to the index. For removed stocks, the post-removal price is treated as the terminal price of the stock.

My goal in this study will be to use the price information from any given day to forecast returns for the next 21 trading days (1 month) and for the next 253 trading days (1 year). In quantitative finance, there are two common approaches to using past price data to forecast future price data: the "mean reversion" approach and the "momentum" approach. "Mean reversion" traders seek to "buy the dip" on stocks that have experienced a sharp decrease on price, on the expectation that the selling is driven by irrational panic and that the price will rise sharply as the panic subsides. "Momentum" traders, on the other hand, seek to buy stocks that have already made strong price gains, on the assumption that whatever secular tailwinds or intangible advantages have driven a company's prior outperformance will continue to do so. In this study, I will use machine learning to build a model that combines these two approaches to trading.

A perpetual problem in both approaches is how to *measure* momentum and mean reversion. I will use the historical distribution of a stock price's distance (in percentage terms) from various simple moving averages. For instance, if a stock's price has, on average, traded 1% above its 20-day simple moving average with a 2% standard deviation, then we can use that information to judge whether the stock price on any given day is low or high in its historical distribution. If the stock is trading several standard deviations below or above its mean, then it might be a candidate for a momentum or mean reversion trade.

### Data Acquisition

I have already obtained and pre-processed the data necessary for this analysis and made it available as an R workspace file ("stockreturnscapstone.RData"). To access this data, you may simply place the R workspace file in your working directory and import it with the following code:

However, to demonstrate the steps by which I obtained and cleaned this data, I will describe the steps and provide the necessary code.

I wanted historical price data for all S&P 500 constituents, both current and former. Fortunately, I was able to easily scrape a list of stock ticker symbols for both current and past index constituents from https://en.wikipedia.org/wiki/List_of_S%26P_500_companies using the rvest package. The list of past index constituents appears comprehensive back to 2007. Prior to 2007, the list is spotty. Thus, I will make 2007 the starting point for my analysis.

I will limit my analysis to stocks in the S&P 500 index. Closely examining the Wikipedia list, I immediately run into a problem: 8 stock was removed from the index, added back into the index, and then removed again. Later, for this stock, I will create two duplicate price series—one for the stock's first stint in the index, and another for its second stint. To distinguish the two duplicate series, I will add numbers to the end of the ticker (e.g. "MXIM1" and "MXIM2"). I also find that for many stocks in the index, I am missing the date that the stock was added to the index. This means that for stocks added to the index after 2007, my analysis is going to include some price data from before the stocks were added to the index. Without doing a lot of manual research and data entry, this is unavoidable, and I accept it as a limitation of my data.

Next, I downloaded historical price data for each S&P 500 ticker symbol from the Yahoo! Finance API using the tidyquant package, which is available through CRAN. Of the downloaded data, I keep only the ticker symbol, the date, and the adjusted closing price. (The adjusted closing price variable subtracts dividend payouts from prior historical prices so that the price series for a given stock reflects the stock's total return, including dividends. For the industry standard adjustment methodology as outlined by the Center for Research in Security Prices, see https://www.crsp.org/products/documentation/crsp-calculations.)

Unfortunately, the Yahoo! Finance API does not provide price data for delisted stocks. Comparing my downloaded price data to my full list of S&P 500 ticker symbols revealed that I had failed to obtain price data for 96 of my 774 ticker symbols. I therefore had to supplement the Yahoo! Finance data with data from another source. I found what I was looking for on Kaggle, uploaded by user Evan Hallmark and downloadable as a .zip archive from https://www.kaggle.com/ehallmar/daily-historical-stock-prices-1970-2018. After downloading the .zip archive, I extracted it to my working directory. It contained two .csv files: one containing price data for thousands of ticker symbols, and the other containing information about each ticker symbol, including the company's full name and sector and industry classification. I imported the price data file into R as a data frame and filtered it to keep only S&P 500 ticker symbols for which I had failed to import data from the Yahoo! Finance API.

Unfortunately, even after importing the Kaggle data, I am still missing historical price data for 55 of the 774 ticker symbols on my original list. Thus, I tried one more Kaggle data set uploaded by user Boris Marjanovic and downloadable as a .zip archive from https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs. I extracted the data to a "Stocks" folder in my working directory and executed the following code:

Unfortunately, this data set has prices for only 4 of my missing ticker symbols; I am still missing historical price data for 51 of the 774 ticker symbols on my original list.

Ideally I would have liked to find a source of free dividend-adjusted historical price data for the remaining ticker symbols so that they could be included in my analysis, but I was unable to locate one. Access to datasets with prices of delisted stocks generally costs upwards of 500 dollars. Fortunately, close examination of the remaining list of ticker symbols reveals that all but one belong to companies that ended in an acquisition, merger, or spin-off. In other words, these companies didn't fail; they transformed. Thus, they can be excluded from the analysis without too much fear that their exclusion will cause large survivorship bias in my forecasting model. I was, unfortunately, unable to find free historical price data for one company that ended in bankruptcy: Lehman Brothers. This does introduce some survivorship bias. In the scope of this project, with 723 included ticker symbols, hopefully the bias caused by the 51 excluded ticker symbols will be relatively small.

## Data Cleaning

Now that I have acquired my data, I will clean it. First, I will create my duplicate time series for stocks that have done more than one stint in the index since 2007 (i.e., were removed from the index and then added back into the index):

Next, I will remove from my data frame any ticker symbols without in-sample price data. For instance, if the data I acquired is all from a period when the stock was not in the S&P 500, then the data is all out-of-sample and I will delete the ticker symbol from my data frame.

After excluding out-of-sample data, the total number of S&P 500 ticker symbols for which I have no data rises from 51 to 124. Unofrtunately, this greatly increases the possibility of survivorship bias in my data.

I next filter my data to exclude all data points for each stock that are dated more than one day after the stock's removal from the S&P 500 index:

A limitation of the data I acquired from Kaggle is that the first data set only includes prices up to 2018-08-24, and the second data set only includes prices up to 2017-11-10. Checking the final dates for every ticker symbol to see if any of them are equal to these dates, I find that 22, price series are indeed truncated at the data set's end date. The truncation of these 22 price series will further exacerbate the survivorship bias caused by the 124 ticker symbols that are wholly excluded from the data. This is a real limitation of my data which I unfortunately have not been wholly able to overcome, despite my best efforts. Thus, the conclusions drawn from this study must ultimately be tentative.

As a final step, I check to see whether my data accurately reflects the terminal value of companies delisted from the S&P 500 index. Did any of these companies become untradeable or go to zero when they were removed from the index? In all, there are 3 companies that went bankrupt or were delisted from their stock exchange at the time of their removal from the index: PCG, LEH, MXIM1. Of these, the only one we have data for is the PG&E Corporation, with the ticker symbol "PCG". PG&E did not go to zero or become untradeable, and our data set accurately reflects its terminal price of $7.23 after index removal.

## Data Processing

With my data acquired and cleaned, it's time to process the data and to add my outcome variables and predictors.

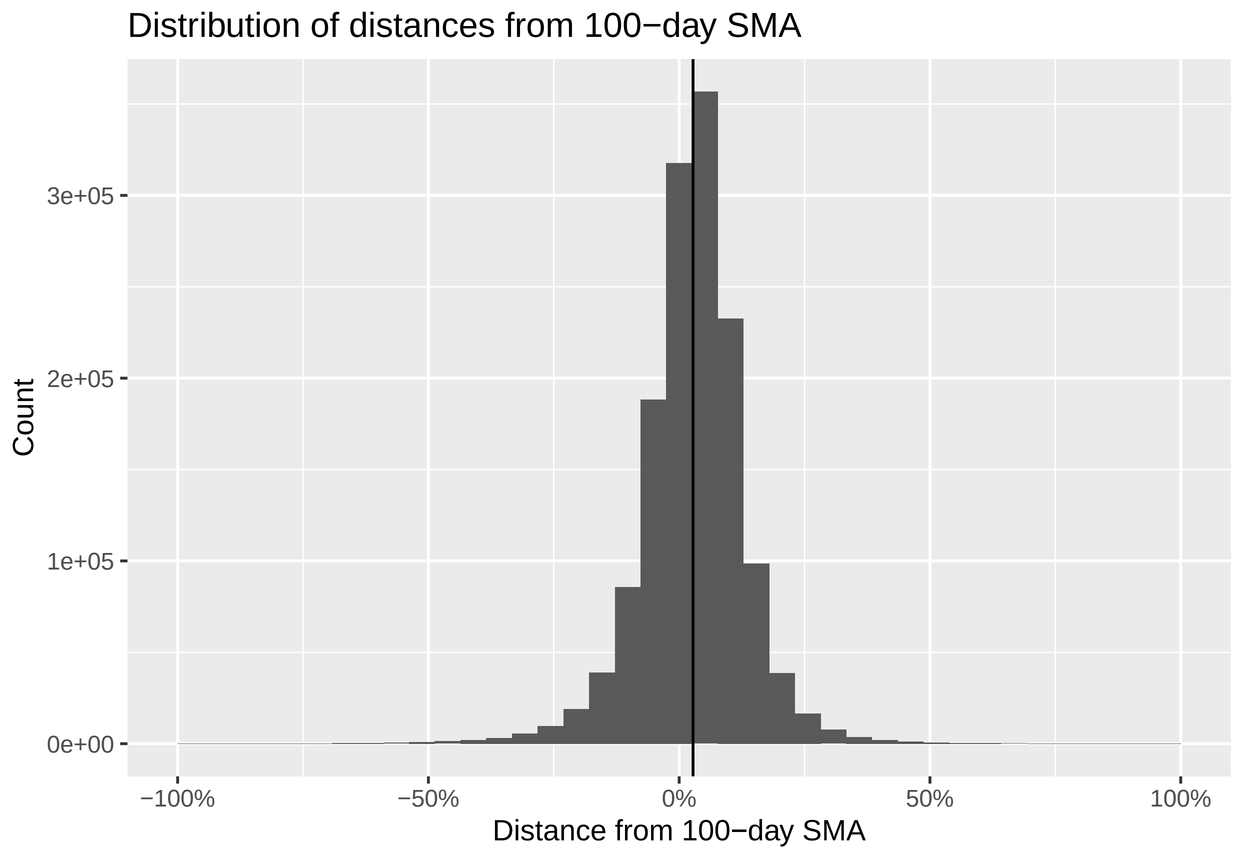
First, for every date in our data set after January 1, 2007, I calculate the return over the next 21 and 253 trading days (1-month and 1-year return). If fewer than 21 or 253 trading days remain until a stock's removal from the index, I calculate the 21-day or 253-day return as difference from terminal value. Because I'm comparing returns over different time periods, I convert 21-day and 253-day returns to compound daily returns (CDRs). These CDR variables are the outcome variables that I will attempt to forecast.

Sometimes, data entry errors in our price series can cause apparently anomalous returns. Searching our data for 21-day returns greater than 600% or less than -85% reveals a few possible problem ticker symbols: "CZR", "GGP", "PENN", "LLL", "CBE2", "NYX", and "TIE". Further exploration of data for these tickers reveals that the data for "CZR", "GGP", and "PENN" is accurate, but that there are errors in the price data for "LLL", "CBE2", "NYX", and "TIE". I thus exclude these ticker symbols from my data.

Next, I add several simple moving averages of various lengths (7-day, 20-day, 50-day, 100-day, 200-day, and 500-day). Simple moving averages are averages of some trailing number of data points. They're commonly used for "technical analysis" by stock traders, primarily in identifying trends. I will use them for gauging whether a stock price is low or high relative to its recent history. Thus, I also add variables showing the difference between the current price and each of the simple moving averages.

Note that the percentage distances from each simple moving average are approximately normally distributed. I illustrate this with a histogram plot. The plot below shows all distances, for all tickers, from the 100-day simple moving average. I could make similar plots for the other moving averages and for individual stocks, though each stock's plot would have its own mean and standard deviation, and the distributions for individual stocks would be noisier and less perfectly normal than the plot below.

```r
#Plot distribution of distances to illustrate that they are normally distributed
df %>%
  filter(!is.na(distance_from_100) & between(distance_from_100,-1,1)) %>%
  ggplot(aes(x=distance_from_100)) +
  geom_histogram(bins=40) +
  geom_vline(xintercept=mean(df$distance_from_100,na.rm=TRUE)) +
  scale_x_continuous(limits = c(-1,1),
                     labels = scales::label_percent()) +
  labs(title = "Distribution of distances from 100-day SMA",
       x="Distance from 100-day SMA",
       y="Count")
```



Because these distributions are approximately normal, I can convert my percent distances to z-scores. Since I am trying to use past price data to predict the future, I must recalculate the mean and standard deviation for each individual date, using only data that preceded that date. I calculate my z-scores using the historical distributions for individual stocks, recognizing that individual companies have different features and their stock prices have different variability. (Ideally we would do some normalization here, but the calculation of these z-scores is already too computationally intensive.)

Many “mean reversion” traders argue that one should wait until a stock’s price “finds a bottom” and begins to level out or move upward before “buying the dip.” If this theory is correct, then the z-score on any given date is not the only relevant variable; we also need to know if it has improved over the z-score from previous dates. To assess this, I will add a 20-day moving average for each of my z-scores and calculate the difference between today’s z-score and the average of the previous 20 days.

I will also add a few final variables. First, each stock’s compound daily return to-date over its lifetime to-date and the total number of days that the stock has traded to-date over its lifetime. I also calculate the average compound daily return (CDR) to-date, although later I will experiment with calculating a normalized version of this average. Second, I also add the total average daily return of all S&P 500 stocks in the data set over their lifetimes to-date. And finally, I will add the average z-score for the whole index for each date, as well each stock’s z-score percentile rank (a measure of relative strength).

### Data Partitioning

With predictor variables and outcome variables added to the data frame, I now remove any rows without NA values for my outcome variables (which will include all dates prior to 2007). Then I partition my data into training, testing, and validation sets. My training set contains 80% of the data, while my testing and validation sets each contain 10%. To better simulate the forecasting problem I'm trying to solve (using past data to forecast future data), I don't partition my data randomly. Rather, I partition chronologically. The first 80% of my data goes into the training set, the next 10% into the test set, and the final 10% into the validation set.

I also define a function for evaluating the accuracy of my predictions using residual mean squared error (RMSE):

As I train my models, I will seek to minimize this RMSE metric.

Finally, I save my workspace as "stockreturnscapstone.RData" to make it easy to load without re-running all this computationally intensive code.

## Methods

### Data Exploration

I begin with some simple data exploration. First, I examine whether a stock's historical compound daily return has any predictive power for future returns. First I look at correlation coefficients:

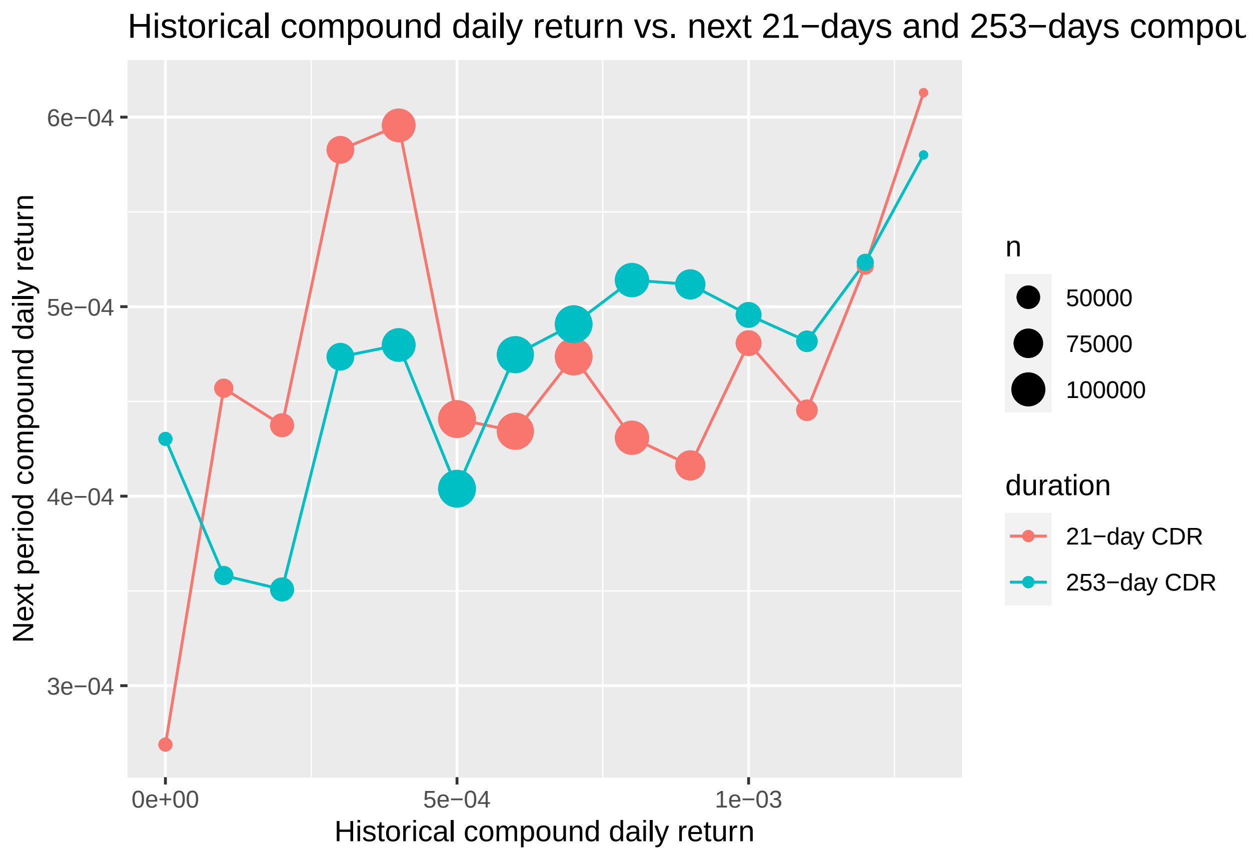
```r
#Plot historical CDR against future return
data.frame(`return 21-day` = cor(train$lifetime_cdr_to_date,train$cdr_21,use="complete.obs"),
           `return 253-day` = cor(train$lifetime_cdr_to_date,train$cdr_253,use="complete.obs")) %>%
  knitr::kable(caption="Correlation coefficients for historical CDR vs. future CDR")
```

Table 1: Correlation coefficients for historical CDR vs. future CDR

| return.21.day | return.253.day |
|---:|---:|
| 0.0021498 | 0.0181968 |

Both correlation coefficients are positive, but both are small, and the coefficient for 21-day return is basically indistinguishable from noise. We can also stratify by historical CDR and see that there does appear to be a positive relationship, though the data are very noisy:

```r
#Plot historical CDR against future return
facet_plot <- train %>%
  ungroup() %>%
  mutate(historical_cdr = round(lifetime_cdr_to_date*10000)/10000) %>%
  group_by(historical_cdr) %>%
  summarize(mean_cdr_21 = mean(cdr_21),
            mean_cdr_253 = mean(cdr_253,na.rm=TRUE),
            n=n()) %>%
  filter(n>20000)
facet_plot <- gather(facet_plot,duration,cdr,2:3)
facet_plot$duration[facet_plot$duration == "mean_cdr_21"] <- "21-day CDR"
facet_plot$duration[facet_plot$duration == "mean_cdr_253"] <- "253-day CDR"
facet_plot %>%
  ggplot(aes(x=historical_cdr,y=cdr,col=duration)) +
  geom_line() +
  geom_point(aes(size=n)) +
  labs(title="Historical compound daily return vs. next 21-days and 253-days compound daily return",
       x="Historical compound daily return",
       y="Next period compound daily return")
```



Next, I explore the data to see if there's any evidence for a mean reversion or momentum effect, as measured by z-score. First, I calculate simple correlation coefficients:

```
#calculate correlation coefficients for z-score vs. 21-day cdr
data.frame(z_score_7 = cor(train$z_score_7,train$cdr_21,use="complete.obs"),
           z_score_20 = cor(train$z_score_20,train$cdr_21,use="complete.obs"),
           z_score_50 = cor(train$z_score_50,train$cdr_21,use="complete.obs"),
           z_score_100 = cor(train$z_score_100,train$cdr_21,use="complete.obs"),
           z_score_200 = cor(train$z_score_200,train$cdr_21,use="complete.obs"),
           z_score_500 = cor(train$z_score_500,train$cdr_21,use="complete.obs")) %>%
  knitr::kable(caption="Correlation coefficients for z-score vs. 21-day return")
```

Table 2: Correlation coefficients for z-score vs. 21-day return

| z_score_7 | z_score_20 | z_score_50 | z_score_100 | z_score_200 | z_score_500 |
|---|---|---|---|---|---|
| -0.023309 | -0.0360996 | -0.0473899 | -0.0528853 | -0.0391271 | -0.0451393 |

```r
#calculate correlation coefficients  for z-score vs. 253-day return
data.frame(z_score_7 = cor(train$z_score_7,train$cdr_253,use="complete.obs"),
           z_score_20 = cor(train$z_score_20,train$cdr_253,use="complete.obs"),
           z_score_50 = cor(train$z_score_50,train$cdr_253,use="complete.obs"),
           z_score_100 = cor(train$z_score_100,train$cdr_253,use="complete.obs"),
           z_score_200 = cor(train$z_score_200,train$cdr_253,use="complete.obs"),
           z_score_500 = cor(train$z_score_500,train$cdr_253,use="complete.obs")) %>%
  knitr::kable(caption="Correlation coefficients for z-score vs. 253-day return")
```

Table 3: Correlation coefficients for z-score vs. 253-day return

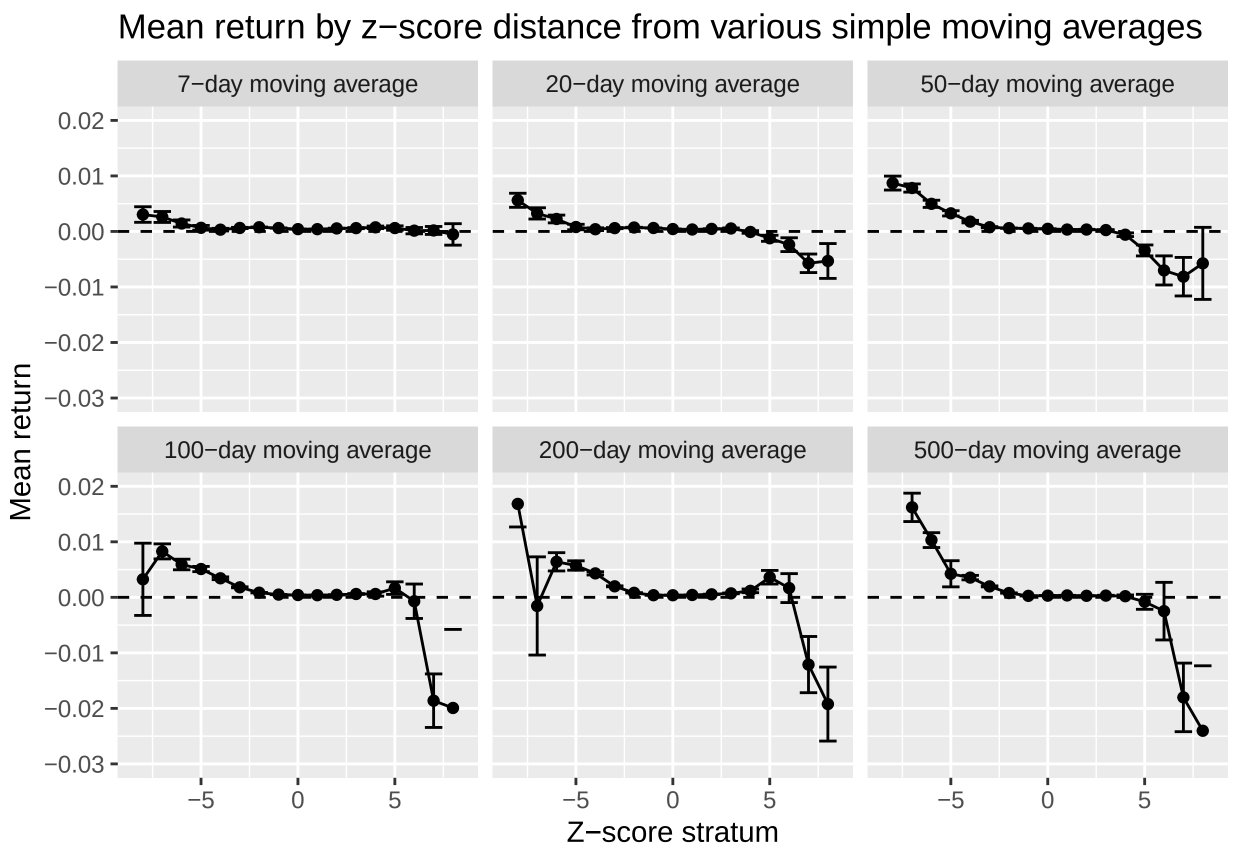
| z_score_7 | z_score_20 | z_score_50 | z_score_100 | z_score_200 | z_score_500 |
|---|---|---|---|---|---|
| -0.0168965 | -0.0231184 | -0.0310306 | -0.0244984 | -0.0226816 | -0.0549955 |

It appears that z-score is negatively correlated with mean return for all moving averages. This provides evidence for a "mean reversion" effect and for the strategy of "buying the dip." But is a linear model the best model for this relationship? To find out, I stratify my data by z-score and create a few test plots of mean return (with 95% confidence interval) by z-score stratum for different moving averages:

```r
#stratify by z-score and plot mean and 95% CI for return for each stratum
facet_plot <- train %>%
  ungroup() %>%
  mutate(`7-day moving average` = round(z_score_7),
         `20-day moving average` = round(z_score_20),
         `50-day moving average` = round(z_score_50),
         `100-day moving average` = round(z_score_100),
         `200-day moving average` = round(z_score_200),
         `500-day moving average` = round(z_score_500)) %>%
  select(`7-day moving average`,`20-day moving average`,`50-day moving average`,
         `100-day moving average`,`200-day moving average`,`500-day moving average`,cdr_21)
facet_plot <- gather(facet_plot,moving_average,z_score,1:6) %>%
  mutate(moving_average = factor(moving_average,levels=c("7-day moving average",
                                                         "20-day moving average",
                                                         "50-day moving average",
                                                         "100-day moving average",
                                                         "200-day moving average",
                                                         "500-day moving average"))) %>%
  group_by(moving_average,z_score) %>%
  summarize(n = n(),
            mean_return = mean(cdr_21),
            se_return = sd(cdr_21)/sqrt(n)) %>%
  filter(between(z_score,-8,8))
```

```
## 'summarise()' has grouped output by 'moving_average'. You can override using the '.groups' argument.
```

```r
facet_plot %>%
  ggplot(aes(x=z_score,y=mean_return)) +
  geom_line() +
  geom_point() +
  geom_hline(aes(yintercept=0),linetype=2) +
  geom_errorbar(aes(ymin = mean_return - 2*se_return, ymax = mean_return + 2*se_return)) +
  labs(title = "Mean return by z-score distance from various simple moving averages",
       x="Z-score stratum",
       y="Mean return") +
  scale_y_continuous(limits = c(-.03,.02)) +
  facet_wrap( ~ moving_average)
```



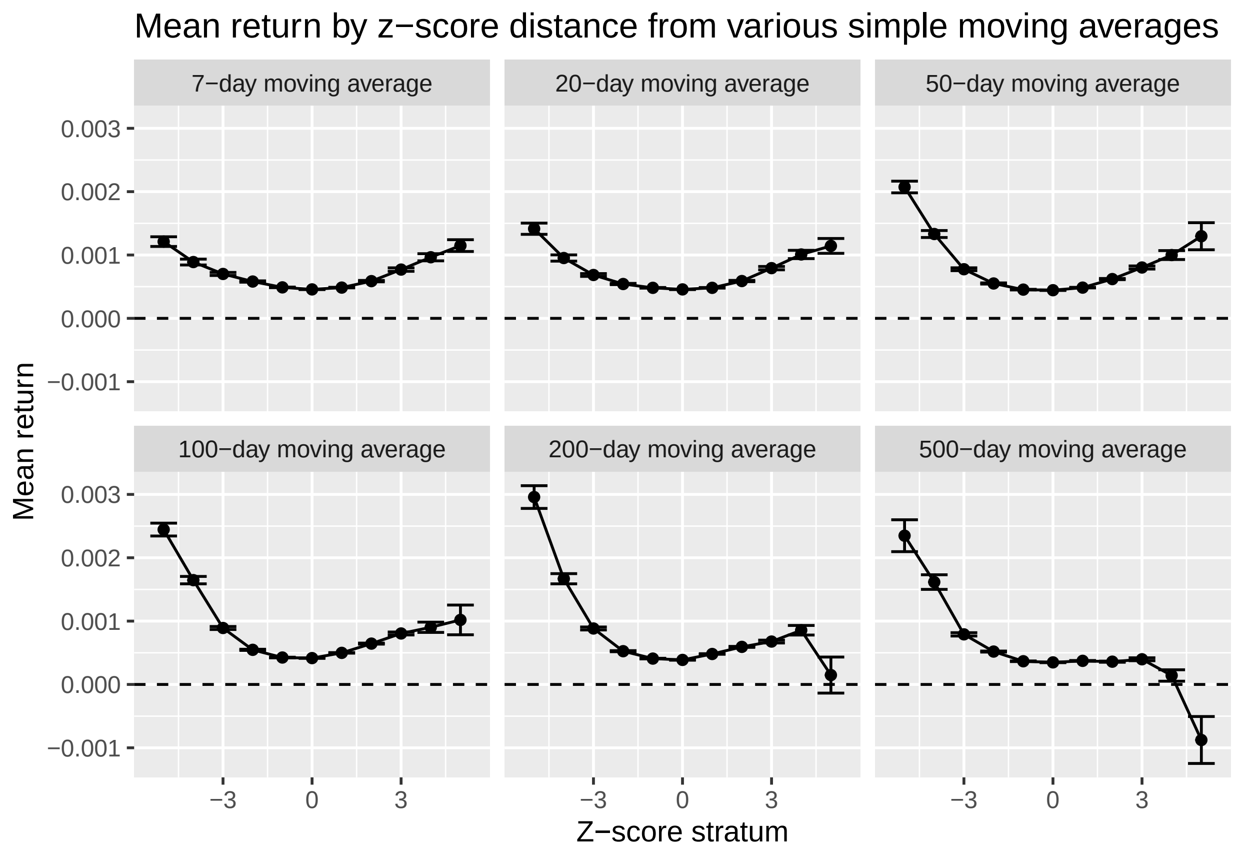
Here I find strong evidence for a "mean reversion" effect on both the upside and the downside; stocks that move too far from their moving averages tend to move decisively back toward those averages in my data. I also find weaker evidence for a "momentum" effect, particularly on the 100-day and 200-day moving average plots. Stocks priced 5 standard deviations above their historical mean in relationship to these two moving averages tend to enjoy outsize returns over the next 21 days.

Repeating this analysis for 253-day return, I find much stronger evidence for a momentum effect on this longer timeline:

```
#Recreate the above plot for 253-day return

#stratify by z-score and plot mean and 95% CI for return for each stratum
facet_plot <- train %>%
  ungroup() %>%
  filter(!is.na(return_253)) %>%
  mutate(`7-day moving average` = round(z_score_7),
         `20-day moving average` = round(z_score_20),
         `50-day moving average` = round(z_score_50),
         `100-day moving average` = round(z_score_100),
         `200-day moving average` = round(z_score_200),
         `500-day moving average` = round(z_score_500)) %>%
  select(`7-day moving average`,`20-day moving average`,`50-day moving average`,
         `100-day moving average`,`200-day moving average`,`500-day moving average`,cdr_253)
facet_plot <- gather(facet_plot,moving_average,z_score,1:6) %>%
  mutate(moving_average = factor(moving_average,levels=c("7-day moving average",
                                                         "20-day moving average",
                                                         "50-day moving average",
                                                         "100-day moving average",
                                                         "200-day moving average",
                                                         "500-day moving average"))) %>%
  group_by(moving_average,z_score) %>%
  summarize(n = n(),
            mean_return = mean(cdr_253),
            se_return = sd(cdr_253)/sqrt(n)) %>%
  filter(between(z_score,-5,5))
```

```
## `summarise()` has grouped output by 'moving_average'. You can override using the `.groups` argument.
```

```
facet_plot %>%
  ggplot(aes(x=z_score,y=mean_return)) +
  geom_line() +
  geom_point() +
  geom_hline(aes(yintercept=0),linetype=2) +
  geom_errorbar(aes(ymin = mean_return - 2*se_return, ymax = mean_return + 2*se_return)) +
  labs(title = "Mean return by z-score distance from various simple moving averages",
       x="Z-score stratum",
       y="Mean return") +
  scale_y_continuous() +
  facet_wrap( ~ moving_average)
```



Thus, while an elevated z-score may tend to be followed by “mean reversion” to the downside on a one-month timeframe, it tends to create momentum to the upside on a one-year timeframe.

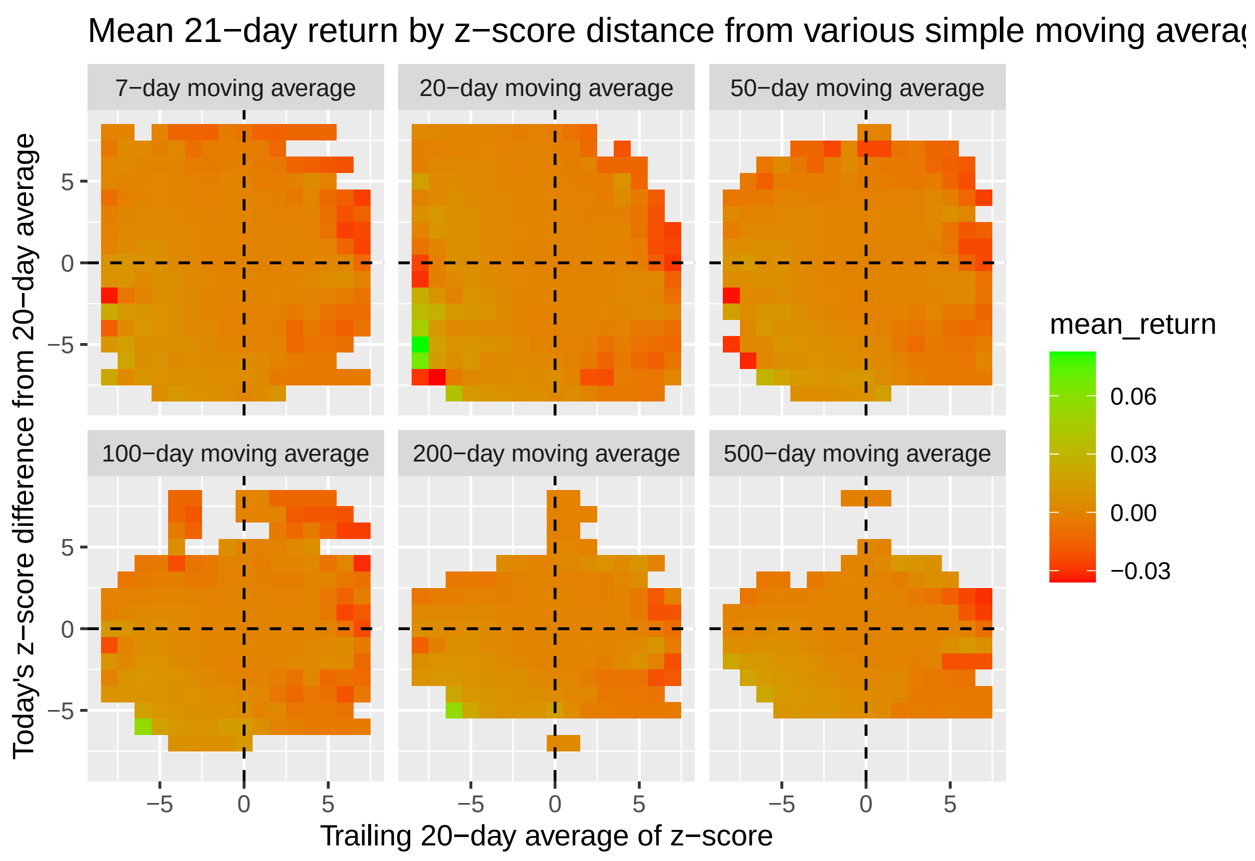
Obviously, it does not appear that a linear model would capture the shape of our data very well. This, I will lean toward more granular models like loess and k-nearest neighbors.

What about our simple moving average of z-score for the last 20 days? Does this variable have predictive power? And can the difference between today’s z-score and the last 20 days z-score tell us anything useful? To find out, I try double-stratifying my data by the 20-day average and the difference from today’s z-score:

```
#Double stratify 20-day moving average of z-score and its difference from same-day z-score
#and create a heat map showing 21-day return
facet_plot <- train %>%
  ungroup() %>%
  mutate(`7-day moving average avg` = round(zscore_average_7),
         `20-day moving average avg` = round(zscore_average_20),
         `50-day moving average avg` = round(zscore_average_50),
         `100-day moving average avg` = round(zscore_average_100),
         `200-day moving average avg` = round(zscore_average_200),
         `500-day moving average avg` = round(zscore_average_500)) %>%
  mutate(`7-day moving average` = round(diff_7),
         `20-day moving average` = round(diff_20),
         `50-day moving average` = round(diff_50),
         `100-day moving average` = round(diff_100),
         `200-day moving average` = round(diff_200),
         `500-day moving average` = round(diff_500)) %>%
  select(`7-day moving average avg`,`20-day moving average avg`,`50-day moving average avg`,
         `100-day moving average avg`,`200-day moving average avg`,`500-day moving average avg`,
         `7-day moving average`,`20-day moving average`,
         `50-day moving average`,`100-day moving average`,
         `200-day moving average`,`500-day moving average`,
         cdr_21,cdr_253)
facet_plot <- gather(facet_plot,moving_average,ma_z_score,1:6)
facet_plot <- gather(facet_plot,moving_average,diff_z_score,1:6) %>%
  mutate(moving_average = factor(moving_average,levels=c("7-day moving average",
                                                         "20-day moving average",
                                                         "50-day moving average",
                                                         "100-day moving average",
                                                         "200-day moving average",
                                                         "500-day moving average")))
facet_plot <- facet_plot %>%
  group_by(moving_average,ma_z_score,diff_z_score) %>%
  summarize(n = n(),
            mean_return = mean(cdr_21),
            se_return = sd(cdr_21)/sqrt(n)) %>%
  filter(between(ma_z_score,-8,8) & between(diff_z_score,-8,8))
```

```
## ‘summarise()‘ has grouped output by ’moving_average’, ’ma_z_score’. You can override using the ‘.grou
```

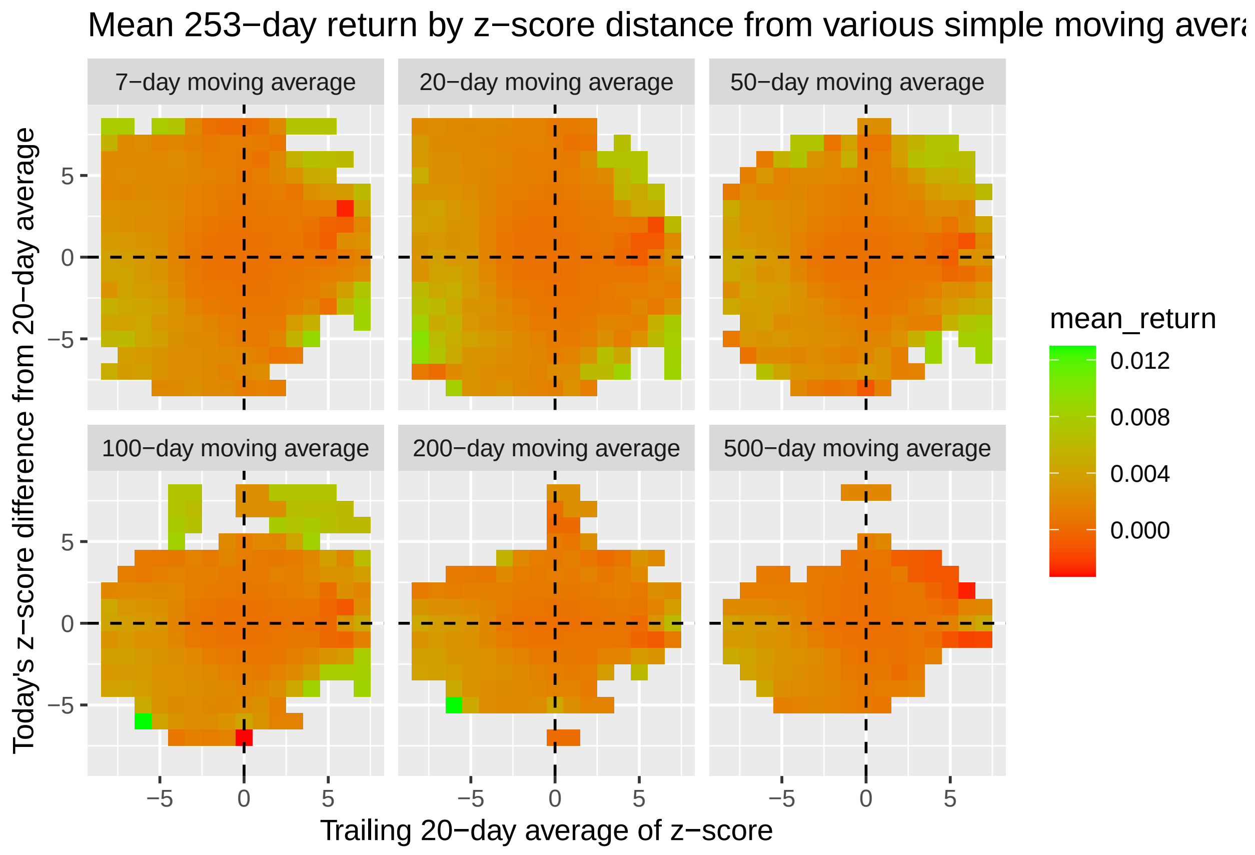
```
facet_plot %>%
  ggplot(aes(x=ma_z_score,y=diff_z_score,fill=mean_return)) +
  geom_tile() +
  geom_hline(aes(yintercept=0),linetype=2) +
  geom_vline(aes(xintercept=0),linetype=2) +
  labs(title = "Mean 21-day return by z-score distance from various simple moving averages",
       x="Trailing 20-day average of z-score",
       y="Today's z-score difference from 20-day average") +
  scale_fill_gradient(low="red",high="green") +
  facet_wrap( ~ moving_average)
```



```
#Double stratify 20-day moving average of z-score and its difference from same-day z-score
#and create a heat map showing 253-day return
facet_plot <- train %>%
  filter(!is.na(cdr_253)) %>%
  ungroup() %>%
  mutate(`7-day moving average avg` = round(zscore_average_7),
         `20-day moving average avg` = round(zscore_average_20),
         `50-day moving average avg` = round(zscore_average_50),
         `100-day moving average avg` = round(zscore_average_100),
         `200-day moving average avg` = round(zscore_average_200),
         `500-day moving average avg` = round(zscore_average_500)) %>%
  mutate(`7-day moving average` = round(diff_7),
         `20-day moving average` = round(diff_20),
         `50-day moving average` = round(diff_50),
         `100-day moving average` = round(diff_100),
         `200-day moving average` = round(diff_200),
         `500-day moving average` = round(diff_500)) %>%
  select(`7-day moving average avg`,`20-day moving average avg`,`50-day moving average avg`,
         `100-day moving average avg`,`200-day moving average avg`,`500-day moving average avg`,
         `7-day moving average`,`20-day moving average`,
         `50-day moving average`,`100-day moving average`,
         `200-day moving average`,`500-day moving average`,
         cdr_21,cdr_253)
facet_plot <- gather(facet_plot,moving_average,ma_z_score,1:6)
facet_plot <- gather(facet_plot,moving_average,diff_z_score,1:6) %>%
  mutate(moving_average = factor(moving_average,levels=c("7-day moving average",
                                                         "20-day moving average",
                                                         "50-day moving average",
                                                         "100-day moving average",
                                                         "200-day moving average",
                                                         "500-day moving average")))
facet_plot <- facet_plot %>%
  group_by(moving_average,ma_z_score,diff_z_score) %>%
  summarize(n = n(),
            mean_return = mean(cdr_253),
            se_return = sd(cdr_253)/sqrt(n)) %>%
  filter(between(ma_z_score,-8,8) & between(diff_z_score,-8,8))
```

```
## ‘summarise()‘ has grouped output by ’moving_average’, ’ma_z_score’. You can override using the ‘.grou
```

```
facet_plot %>%
  ggplot(aes(x=ma_z_score,y=diff_z_score,fill=mean_return)) +
  geom_tile() +
  geom_hline(aes(yintercept=0),linetype=2) +
  geom_vline(aes(xintercept=0),linetype=2) +
  labs(title = "Mean 253-day return by z-score distance from various simple moving averages",
       x="Trailing 20-day average of z-score",
       y="Today's z-score difference from 20-day average") +
  scale_fill_gradient(low="red",high="green") +
  facet_wrap( ~ moving_average)
```



These heatmaps do offer some reason for optimism about using these two variables in combination, but they may prove difficult to operationalize in a predictive model.

### Building a forecasting model

As a baseline to compare against, I estimate the RMSE of forecasting a CDR of zero for all data points in my test set.

```r
#Calculate RMSE using a forecast of zero for all CDR values in test set
actual_21 <- test$cdr_21
actual_253 <- test$cdr_253[!is.na(test$cdr_253)]
model_rmses <- data.frame(model = "Zero",rmse_21_day = RMSE(actual_21,0),rmse_253_day = RMSE(actual_253
model_rmses %>%
  knitr::kable(caption="Model RMSEs")
```

Table 4: Model RMSEs

| model | rmse_21_day | rmse_253_day |
|---|---:|---:|
| Zero | 0.0042559 | 0.0013287 |

As an initial step in building a forecasting model, I use average CDR for the entire index to-date. Because of the restriction that I can only use past data as a predictor, I don't use exactly the same average for all of my predictions. The mean value used is $6.9341648 \times 10^{-4}$, and the standard deviation of the values used is $3.5905335 \times 10^{-4}$. To use such a wide range of values is sub-optimal, and the value would be more stable if I had more historical data, but I work with what I have.

```r
#Calculate RMSE using average S&P 500 CDR to-date
forecast_21 <- test$avg_sp500_cdr_to_date
forecast_253 <- test$avg_sp500_cdr_to_date[!is.na(test$cdr_253)]
model_rmses <- bind_rows(model_rmses,data.frame(model = "Mean S&P 500 CDR to-date",rmse_21_day = RMSE(ac
model_rmses %>%
  knitr::kable(caption="Model RMSEs")
```

Table 5: Model RMSEs

| model | rmse_21_day | rmse_253_day |
|---|---:|---:|
| Zero | 0.0042559 | 0.0013287 |
| Mean S&P 500 CDR to-date | 0.0042372 | 0.0012758 |

Using the index mean slightly improves accuracy over using zero. So far, so good. Next, I will try using the mean CDR to-date for each individual stock.

```r
#Calculate RMSE using individual stock's average CDR to-date
forecast_21 <- test$lifetime_cdr_to_date
forecast_253 <- test$lifetime_cdr_to_date[!is.na(test$cdr_253)]
model_rmses <- bind_rows(model_rmses,data.frame(model = "Individual stock mean CDR to-date",rmse_21_day
model_rmses %>%
  knitr::kable(caption="Model RMSEs")
```

Table 6: Model RMSEs

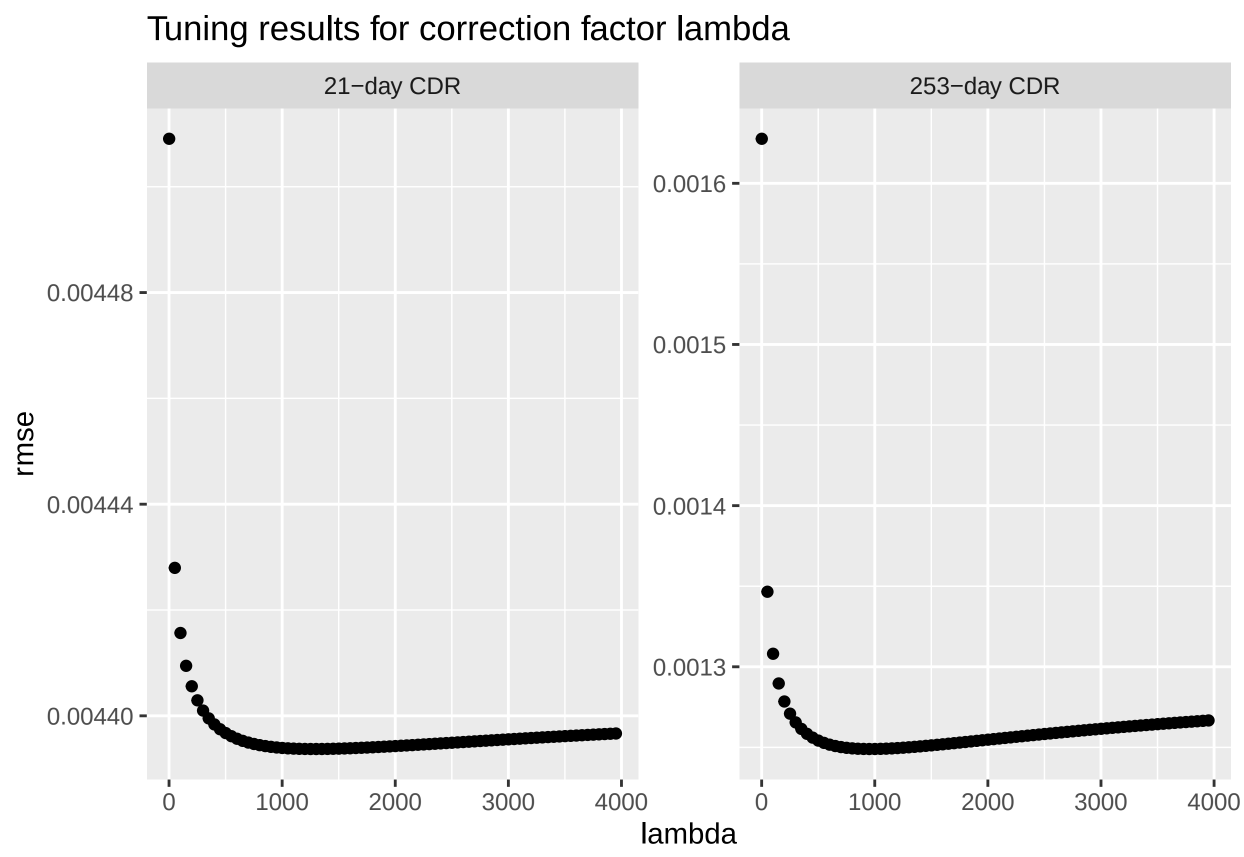
| model | rmse_21_day | rmse_253_day |
|---|---|---|
| Zero | 0.0042559 | 0.0013287 |
| Mean S&P 500 CDR to-date | 0.0042372 | 0.0012758 |
| Individual stock mean CDR to-date | 0.0043830 | 0.0017412 |

This is less accurate than using either zero or the average for the index. It's possible that I could improve this approach using regularization to regress it toward the index average for stocks with a small number of historical data points. To test this, I use five-fold cross-validation to tune a correction factor (lambda) on my training set. I then average the rmse from each of the five folds for each tested value of lambda, and I plot the results:

```
#Attempt to forecast using regularization & five-fold cross-validation for tuning
#Set range of values to try for lambda
lambda <- seq(1, 4000, 50)

#Create folds for 5-fold cross validation to tune lambda
set.seed(1,sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
folds <- createFolds(train$cdr_21, k = 5, list = TRUE, returnTrain = FALSE)

#Perform a five-fold cross-validation to tune lambda
lambdas <- map_dfr(.x=1:5,.f=function(curr_fold){
  #Create fold
  tune_set <- train[folds[[curr_fold]],]

  #See which value of lambda minimizes RMSE when predicting on tune_set
  lambdas <- map_dfr(.x=lambda,.f=function(lambda){

    #sweep out S&P 500 averages from the actuals
    actual_21 <- tune_set$cdr_21 - tune_set$avg_sp500_cdr_to_date
    actual_21 <- actual_21[!is.na(tune_set$cdr_21)]
    actual_253 <- tune_set$cdr_253 - tune_set$avg_sp500_cdr_to_date
    actual_253 <- actual_253[!is.na(tune_set$cdr_253)]

    #sweep out S&P 500 averages from the forecasts
    forecast_21 <- ((tune_set$adjusted/(tune_set$adjusted/(1+tune_set$lifetime_return_to_date)))^(1/(tu
    forecast_21 <- forecast_21[!is.na(tune_set$cdr_21)]
    forecast_253 <- ((tune_set$adjusted/(tune_set$adjusted/(1+tune_set$lifetime_return_to_date)))^(1/(t
    forecast_253 <- forecast_253[!is.na(tune_set$cdr_253)]

    #calculate rmses
    rmses <- data.frame(fold = curr_fold,lambda,rmse_21_day = RMSE(actual_21,forecast_21),rmse_253_day =
  })
})

#Average rmses over the five folds
lambdas <- lambdas %>%
  group_by(lambda) %>%
  summarize(rmse_21_day = mean(rmse_21_day),rmse_253_day = mean(rmse_253_day))

#Plot values of lambda against rmses
lambdas <- gather(lambdas,cdr,rmse,2:3)
lambdas$cdr[lambdas$cdr == "rmse_21_day"] <- "21-day CDR"
lambdas$cdr[lambdas$cdr == "rmse_253_day"] <- "253-day CDR"
lambdas %>% ggplot(aes(x=lambda,y=rmse)) +
  geom_point() +
  facet_wrap( ~ cdr,scales="free") +
  labs(title = "Tuning results for correction factor lambda")
```



```
#Get value of lambda that minimizes RMSE
lambda <- lambdas %>%
  filter(cdr=="253-day CDR") %>%
  filter(rmse == min(rmse)) %>%
  pull(lambda)
```

RMSE is minimized with a correction factor of 951. When we apply this correction factor to get a regularized mean lifetime CDR for each stock and then use this (in combination with the S&P 500 mean) to predict CDR for the test set, we find that, indeed, RMSE improves over the previous models:

```
#use the best value of lambda to forecast on the test set and calculate RMSE
#sweep out S&P 500 averages from the actuals
actual_21 <- test$cdr_21 - test$avg_sp500_cdr_to_date
```

```r
actual_21 <- actual_21[!is.na(test$cdr_21)]
actual_253 <- test$cdr_253 - test$avg_sp500_cdr_to_date
actual_253 <- actual_253[!is.na(test$cdr_253)]

#sweep out S&P 500 averages from the forecasts
forecast_21 <- ((test$adjusted/(test$adjusted/(1+test$lifetime_return_to_date)))^(1/(test$trading_days_
forecast_21 <- forecast_21[!is.na(test$cdr_21)]
forecast_253 <- ((test$adjusted/(test$adjusted/(1+test$lifetime_return_to_date)))^(1/(test$trading_days_
forecast_253 <- forecast_253[!is.na(test$cdr_253)]

#calculate rmses
model_rmses <- bind_rows(model_rmses,data.frame(model = "Individual stock regularized mean CDR to-date"
model_rmses %>%
  knitr::kable(caption="Model RMSEs")
```

Table 7: Model RMSEs

| model | rmse_21_day | rmse_253_day |
|---|---|---|
| Zero | 0.0042559 | 0.0013287 |
| Mean S&P 500 CDR to-date | 0.0042372 | 0.0012758 |
| Individual stock mean CDR to-date | 0.0043830 | 0.0017412 |
| Individual stock regularized mean CDR to-date | 0.0042325 | 0.0012440 |

Finally, I try training a loess model using z-score distance from various moving averages. Unfortunately, this model is too computationally intensive to train on the entire data set. Instead, I stratify the data in much the same way I did during data exploration, and I fit my loess model to the stratified data.

```r
#sweep out averages and stratify by z-score
train_loess <- train %>%
  ungroup() %>%
  mutate(`7-day moving average` = round(z_score_7),
         `20-day moving average` = round(z_score_20),
         `50-day moving average` = round(z_score_50),
         `100-day moving average` = round(z_score_100),
         `200-day moving average` = round(z_score_200),
         `500-day moving average` = round(z_score_500))
train_loess$swept_21 <- train_loess$cdr_21 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_loess
train_loess$swept_253 <- train_loess$cdr_253 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_lo
train_loess <- train_loess %>% select(`7-day moving average`,`20-day moving average`,`50-day moving ave
         `100-day moving average`,`200-day moving average`,`500-day moving average`,swept_21)
train_loess <- gather(train_loess,moving_average,z_score,1:6)
train_loess <- train_loess %>% mutate(moving_average = factor(moving_average,levels=c("7-day moving ave
                                                          "20-day moving average",
                                                          "50-day moving average",
                                                          "100-day moving average",
                                                          "200-day moving average",
                                                          "500-day moving average"))) %>%
  group_by(moving_average,z_score) %>%
  summarize(n = n(),
            mean_return = mean(swept_21),
            se_return = sd(swept_21)/sqrt(n))
```

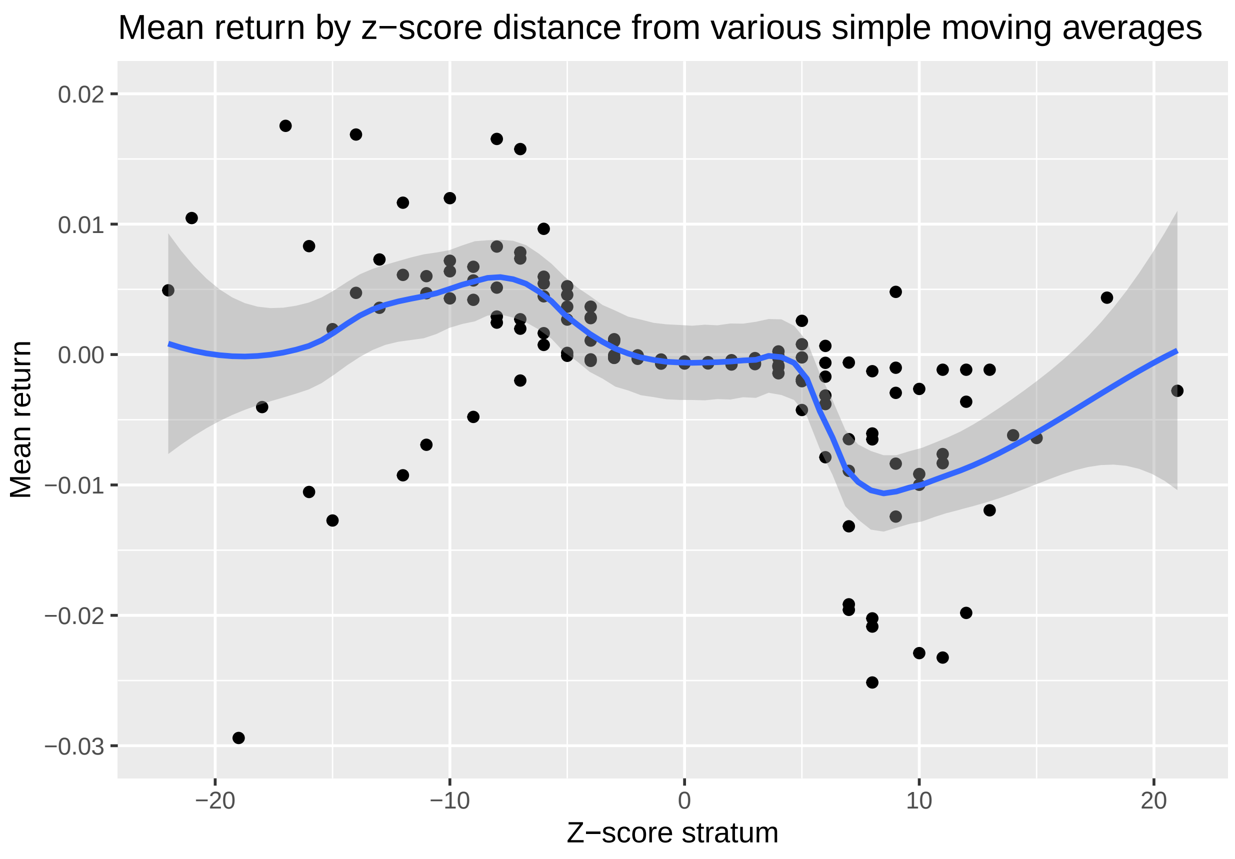
```
## ‘summarise()‘ has grouped output by ’moving_average’. You can override using the ‘.groups‘ argument.
```

```
#Plot all moving average zscores on the same chart and run an experimental loess fit
train_loess %>%
  ggplot(aes(x=z_score,y=mean_return)) +
  geom_point() +
  geom_smooth(span=.3) +
  labs(title = "Mean return by z-score distance from various simple moving averages",
       x="Z-score stratum",
       y="Mean return") +
  scale_y_continuous(limits = c(-.03,.02))
```

```
## ‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```

```
## Warning: Removed 13 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 13 rows containing missing values (geom_point).
```

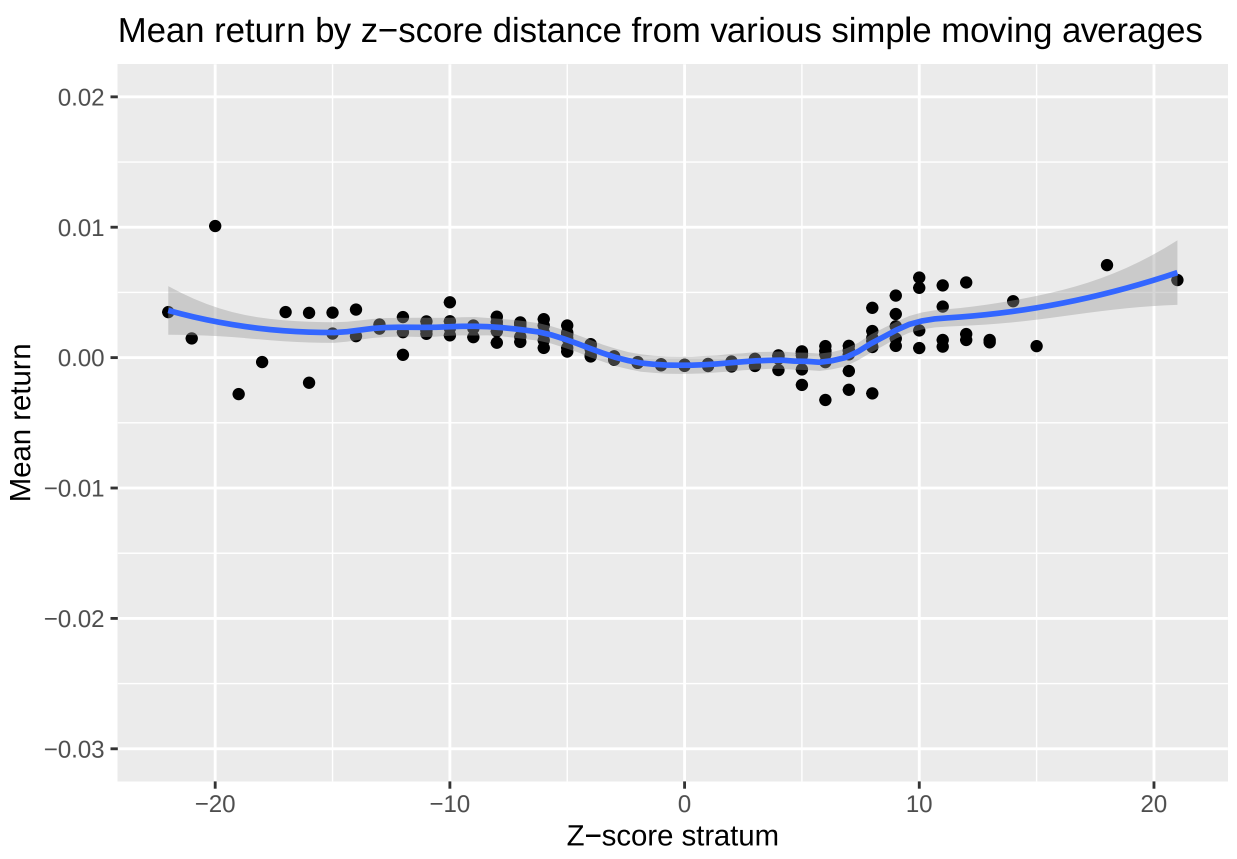


```
#Fit a loess model
loess_model <- loess(mean_return ~ z_score,data=train_loess,span = .3)

#Predict returns for test data set using loess model
test_loess <- test %>% ungroup()
test_loess$swept_21 <- test_loess$cdr_21 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$lif
test_loess$swept_253 <- test_loess$cdr_253 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$li
test_loess <- test_loess %>%
  mutate(average_z_score = (z_score_7+z_score_20+z_score_50+z_score_100+z_score_200+z_score_500)/6)
test_loess <- test_loess %>% select(swept_21,z_score = average_z_score)
test_loess <- test_loess %>% drop_na()
test_loess$forecast <- predict(loess_model,test_loess)
model_rmses <- bind_rows(model_rmses,data.frame(model = "Loess model",rmse_21_day = RMSE(test_loess$swep


#Repeat the above analysis for 253-day return
#sweep out averages and stratify by z-score
train_loess <- train %>%
  ungroup() %>%
  mutate(`7-day moving average` = round(z_score_7),
         `20-day moving average` = round(z_score_20),
         `50-day moving average` = round(z_score_50),
         `100-day moving average` = round(z_score_100),
         `200-day moving average` = round(z_score_200),
         `500-day moving average` = round(z_score_500))
train_loess$swept_21 <- train_loess$cdr_21 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_loess
train_loess$swept_253 <- train_loess$cdr_253 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_loe
train_loess <- train_loess %>% select(`7-day moving average`,`20-day moving average`,`50-day moving aver
                                      `100-day moving average`,`200-day moving average`,`500-day moving
  filter(!is.na(swept_253))
train_loess <- gather(train_loess,moving_average,z_score,1:6)
train_loess <- train_loess %>% mutate(moving_average = factor(moving_average,levels=c("7-day moving aver
                                                                                      "20-day moving ave
                                                                                      "50-day moving ave
                                                                                      "100-day moving av
                                                                                      "200-day moving av
                                                                                      "500-day moving av
  group_by(moving_average,z_score) %>%
  summarize(n = n(),
            mean_return = mean(swept_253),
            se_return = sd(swept_253)/sqrt(n))
```

```
## `summarise()` has grouped output by 'moving_average'. You can override using the `.groups` argument.
```

```
#Plot all moving average zscores on the same chart and run an experimental loess fit
train_loess %>%
  ggplot(aes(x=z_score,y=mean_return)) +
  geom_point() +
  geom_smooth(span=.3) +
  labs(title = "Mean return by z-score distance from various simple moving averages",
       x="Z-score stratum",
       y="Mean return") +
  scale_y_continuous(limits = c(-.03,.02))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'

## Warning: Removed 12 rows containing non-finite values (stat_smooth).

## Warning: Removed 12 rows containing missing values (geom_point).
```



```
#Fit a loess model
loess_model <- loess(mean_return ~ z_score,data=train_loess,span = .3)

#Predict returns for test data set using loess model
test_loess <- test %>% ungroup()
test_loess$swept_21 <- test_loess$cdr_21 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$lif
test_loess$swept_253 <- test_loess$cdr_253 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$li
test_loess <- test_loess %>%
  mutate(average_z_score = (z_score_7+z_score_20+z_score_50+z_score_100+z_score_200+z_score_500)/6)
test_loess <- test_loess %>% select(swept_253,z_score = average_z_score)
test_loess <- test_loess %>% drop_na()
test_loess$forecast <- predict(loess_model,test_loess)
model_rmses$rmse_253_day[nrow(model_rmses)] <- RMSE(test_loess$swept_253,test_loess$forecast)
model_rmses %>%
  knitr::kable(caption="Model RMSEs")
```

Table 8: Model RMSEs

| model | rmse_21_day | rmse_253_day |
|---|---|---|
| Zero | 0.0042559 | 0.0013287 |
| Mean S&P 500 CDR to-date | 0.0042372 | 0.0012758 |
| Individual stock mean CDR to-date | 0.0043830 | 0.0017412 |
| Individual stock regularized mean CDR to-date | 0.0042325 | 0.0012440 |
| Loess model | 0.0043288 | 0.0012387 |

This method results in a slightly better RMSE for forecasting 253-day CDR, but a slightly worse one for forecasting 21-day CDR.

## Results

```r
#Validate model
#sweep out averages and stratify by z-score
train_loess <- train %>%
  ungroup() %>%
  mutate(`7-day moving average` = round(z_score_7),
         `20-day moving average` = round(z_score_20),
         `50-day moving average` = round(z_score_50),
         `100-day moving average` = round(z_score_100),
         `200-day moving average` = round(z_score_200),
         `500-day moving average` = round(z_score_500))
train_loess$swept_21 <- train_loess$cdr_21 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_loess
train_loess$swept_253 <- train_loess$cdr_253 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_loe
train_loess <- train_loess %>% select(`7-day moving average`,`20-day moving average`,`50-day moving aver
                                      `100-day moving average`,`200-day moving average`,`500-day moving
train_loess <- gather(train_loess,moving_average,z_score,1:6)
train_loess <- train_loess %>% mutate(moving_average = factor(moving_average,levels=c("7-day moving aver
                                                                                      "20-day moving ave
                                                                                      "50-day moving ave
                                                                                      "100-day moving av
                                                                                      "200-day moving av
                                                                                      "500-day moving av
  group_by(moving_average,z_score) %>%
  summarize(n = n(),
            mean_return = mean(swept_21),
            se_return = sd(swept_21)/sqrt(n))
```

```
## 'summarise()' has grouped output by 'moving_average'. You can override using the '.groups' argument.
```

```r
#Fit a loess model
loess_model <- loess(mean_return ~ z_score,data=train_loess,span = .3)

#Predict returns for validation data set using loess model
test_loess <- validation %>% ungroup()
test_loess$swept_21 <- test_loess$cdr_21 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$life
test_loess$swept_253 <- test_loess$cdr_253 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$li
test_loess <- test_loess %>%
  mutate(average_z_score = (z_score_7+z_score_20+z_score_50+z_score_100+z_score_200+z_score_500)/6)
test_loess <- test_loess %>% select(swept_21,z_score = average_z_score)
test_loess <- test_loess %>% drop_na()
test_loess$forecast <- predict(loess_model,test_loess)
model_rmses <- bind_rows(model_rmses,data.frame(model = "Final model validation",rmse_21_day = RMSE(tes


#Repeat the above analysis for 253-day return
#sweep out averages and stratify by z-score
train_loess <- train %>%
  ungroup() %>%
  mutate(`7-day moving average` = round(z_score_7),
         `20-day moving average` = round(z_score_20),
         `50-day moving average` = round(z_score_50),
         `100-day moving average` = round(z_score_100),
         `200-day moving average` = round(z_score_200),
         `500-day moving average` = round(z_score_500))
train_loess$swept_21 <- train_loess$cdr_21 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_loess
train_loess$swept_253 <- train_loess$cdr_253 - ((train_loess$adjusted/(train_loess$adjusted/(1+train_loe
train_loess <- train_loess %>% select(`7-day moving average`,`20-day moving average`,`50-day moving aver
                                      `100-day moving average`,`200-day moving average`,`500-day moving
  filter(!is.na(swept_253))
train_loess <- gather(train_loess,moving_average,z_score,1:6)
train_loess <- train_loess %>% mutate(moving_average = factor(moving_average,levels=c("7-day moving aver
                                                                                      "20-day moving ave
                                                                                      "50-day moving ave
                                                                                      "100-day moving av
                                                                                      "200-day moving av
                                                                                      "500-day moving av

  group_by(moving_average,z_score) %>%
  summarize(n = n(),
            mean_return = mean(swept_253),
            se_return = sd(swept_253)/sqrt(n))
```

```
## `summarise()` has grouped output by 'moving_average'. You can override using the `.groups` argument.
```

```
#Fit a loess model
loess_model <- loess(mean_return ~ z_score,data=train_loess,span = .3)

#Predict returns for validation data set using loess model
test_loess <- validation %>% ungroup()
test_loess$swept_21 <- test_loess$cdr_21 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$life
test_loess$swept_253 <- test_loess$cdr_253 - ((test_loess$adjusted/(test_loess$adjusted/(1+test_loess$li
test_loess <- test_loess %>%
  mutate(average_z_score = (z_score_7+z_score_20+z_score_50+z_score_100+z_score_200+z_score_500)/6)
test_loess <- test_loess %>% select(swept_253,z_score = average_z_score)
test_loess <- test_loess %>% drop_na()
test_loess$forecast <- predict(loess_model,test_loess)
model_rmses$rmse_253_day[nrow(model_rmses)] <- RMSE(test_loess$swept_253,test_loess$forecast)
model_rmses %>%
  knitr::kable(caption="Model RMSEs")
```

Table 9: Model RMSEs

| model | rmse_21_day | rmse_253_day |
|---|---:|---:|
| Zero | 0.0042559 | 0.0013287 |
| Mean S&P 500 CDR to-date | 0.0042372 | 0.0012758 |
| Individual stock mean CDR to-date | 0.0043830 | 0.0017412 |
| Individual stock regularized mean CDR to-date | 0.0042325 | 0.0012440 |
| Loess model | 0.0043288 | 0.0012387 |
| Final model validation | 0.0045631 | 0.0012509 |

Using my final model to forecast compound daily return for the validation data set produces a fairly bad RMSE for 21-day return, which points to the challenge of predicting stock prices over a short time frame. The result for 253-day return is somewhat better, although not greatly better than what we would expect from simply using the index average.

## Conclusion

This study result points to the inherent difficulty of forecasting out-of-sample stock price returns using in-sample data. Although we found some fairly compelling patterns, we were not able to produce accurate forecasts of future stock prices from those patterns. Any "edge" we obtained was small, at best.