Calendar heatmaps are a neglected, but valuable, way of representing time series data. Their chief advantage is in allowing the viewer to visually process trends in categorical or continuous data over a period of time, while relating these values to their month, week, and weekday context – something that simple line plots do not efficiently allow for. If you are displaying data on staffing levels, stock returns (as we will do here), on-time performance for transit systems, or any other one dimensional data, a calendar heatmap can do wonders for helping your stakeholders note patterns in the interaction between those variables and their calendar context.

In this post, I will use stock data in the form of daily closing prices for the **SPY – SPDR S&P 500** ETF, the most popular exchange traded fund in the world. For the purposes of this blog post, it’s not necessary to have any familiarity with ETF’s or stocks in general. Some knowledge of tidyverse packages and basic R will be helpful, though.

First, we import the necessary packages. Tidyverse will load in the packages that allow us to do most of our data manipulations (e.g., dplyr for filter, mutate) and plotting (ggplot2), Lubridate will provide helpful datetime functions (e.g. week, year), Quantmod allows us to retrieve stock data through the Yahoo API, Scales provides the helpful percent function for changing double (float) values to percentages, and the Zoo package creates and operates on XTS time series class objects.

library(tidyverse)

library(lubridate)

library(quantmod)

library(scales)

library(zoo)

**Retrieving our Data**

The Quantmod function, getSymbols takes a stock ticker parameter, a flag to determine whether we want a variable containing the data generated automatically without arrow assignment (auto.assign), and the start date of the time series of stock prices that we want to retrieve (from).

stock\_xts <- getSymbols("SPY", auto.assign = FALSE, from="2018-01-01")

Quantmod returns an XTS object, and not a dataframe, as we see when we call class on this object. XTS objects are similar in nature to dataframes and tibbles, but are optimized for time series data, and unlike tibbles and dataframes, are not compatible with tidyverse functions.

class(stock\_xts)

## [1] "xts" "zoo"

In order to use tidyverse (dplyr) functions on this data, we will first need to convert it to a dataframe using the fortify.zoo function. Note that while it is possible to convert an xts object to a dataframe with the as\_data\_frame or as.data.frame functions, these will drop the index, whereas fortify.zoo retains the index as the first column when it is converted.

stock\_df <- fortify.zoo(stock\_xts)

class(stock\_df)

## [1] "data.frame"

Our stock dataframe:

| **Index** | **SPY.Open** | **SPY.High** | **SPY.Low** | **SPY.Close** | **SPY.Volume** | **SPY.Adjusted** |
| --- | --- | --- | --- | --- | --- | --- |
| 2018-01-02 | 267.84 | 268.81 | 267.40 | 268.77 | 86655700 | 261.3322 |
| 2018-01-03 | 268.96 | 270.64 | 268.96 | 270.47 | 90070400 | 262.9851 |
| 2018-01-04 | 271.20 | 272.16 | 270.54 | 271.61 | 80636400 | 264.0936 |
| 2018-01-05 | 272.51 | 273.56 | 271.95 | 273.42 | 83524000 | 265.8535 |
| 2018-01-08 | 273.31 | 274.10 | 272.98 | 273.92 | 57319200 | 266.3397 |
| 2018-01-09 | 274.40 | 275.25 | 274.08 | 274.54 | 57254000 | 266.9426 |

**Calculating and Transforming Columns**

Once we have our stock dataframe, we need to produce year and month columns that we will use to facet (split) the plot by; a wkday column that will form the y axis of our calendar heatmap; a day column representing day of month (the values from which will appear within the squares of the plot); a wk column (week of year) that will form the x axis; and finally a returns column that will show the percentage change in value from the previous day when we map it to a color palette. We will then select the columns we created, leaving the original ones out.

The entirety of this chunk of code (and output) will look like the following (we will break this down in the next section):

stock\_df %>%

mutate(year = year(Index),

month = month(Index, label = TRUE),

wkday = fct\_relevel(wday(Index, label=TRUE),

c("Mon", "Tue","Wed","Thu","Fri","Sat","Sun")

),

day = day(Index),

wk = format(Index, "%W"),

returns = (.[[5]] - lag(.[[5]]))/lag(.[[5]])) %>%

select(year, month, wkday, day, wk, returns)

| **year** | **month** | **wkday** | **day** | **wk** | **returns** |
| --- | --- | --- | --- | --- | --- |
| 2018 | Jan | Tue | 2 | 01 | NA |
| 2018 | Jan | Wed | 3 | 01 | 0.0063252 |
| 2018 | Jan | Thu | 4 | 01 | 0.0042148 |
| 2018 | Jan | Fri | 5 | 01 | 0.0066641 |
| 2018 | Jan | Mon | 8 | 02 | 0.0018287 |
| 2018 | Jan | Tue | 9 | 02 | 0.0022634 |

The lubridate package provides some handy functions that I use on an almost daily basis: year, month, wday, and day. Pass these functions a date or datetime column and they give you the year, the month (specify label = TRUE to get the abbreviated text form e.g., ‘Jan’, ‘Feb’), weekday, and day of month associated with each date.

Note that we use fct\_relevel on the output of our wday call in order to specify the order of the days. On the y axis of the calendar heatmap, we want the days to start from monday at the bottom, and not Sunday (which is the default ordering that wday produces).

wkday = fct\_relevel(wday(Index, label=TRUE),

c("Mon", "Tue","Wed","Thu","Fri","Sat","Sun")

),

The second important thing to notice is the call to format which can be used to extract and rearrange components of dates (among other uses). Here, we are passing it the Index column, like we did with the lubridate functions above it, and the parameter "%W".

wk = format(Index, "%W"),

Finally, we calculate the daily returns. Since quantmod returns an object that has column names with the ticker in the names (e.g., SPY.Close), it would be difficult if you decided later to change the ticker in the initial getSymbols call to see the returns for, say, Disney, since you would then need to go into mutate and change the column names there, each time, as well. Because of this, we will reference the closing price columns by *position* when calculating the returns column. The dots you see stand in for the dataframe, itself, and we use the double bracket indexing we use for lists and vectors to get the fifth column.

In order to compare the difference in price from one date to another, we will use the lag function. lag shifts the column down by one position so that the value originally in position n is now found in position n+1. The returns column calculates the difference between SPY.Closing on day n+1 and SPY.Closing on day n, then divides this by the price on day n.

returns = (.[[5]] - lag(.[[5]]))/lag(.[[5]]))

**Building the Plot**

Once those columns have been created, we will select and pipe them to ggplot. The wk column, representing weeks of the year, is passed to the x axis, while the wkday column, representing the day of the week, is passed to the y axis. We set fill=returns since we are coloring an area by this variable. geom\_tile is added, with color='black' since we want the *borders* of each square to be black (remember that fill colors in *areas* such as bars, and color adds color to single dimension objects such as dots and lines). We then add geom\_text with label=day in order to have the day of the month overlayed on each square that has a value (return) associated with it.

select(year, month, wkday, day, wk, 5, returns) %>%

ggplot(aes(wk, wkday, fill=returns)) +

geom\_tile(color='black') +

geom\_text(aes(label=day), size=3) +

labs(x='',

y='',

title="SPY") +

scale\_fill\_distiller(type="div"

,palette=7

,na.value = 'white'

,limits=c(-.055, .055)

,labels = percent

,direction=1

) +

theme(panel.background = element\_blank(),

axis.ticks = element\_blank(),

axis.text.x = element\_blank(),

strip.background = element\_rect("grey92")

) +

facet\_grid(year~month, scales="free", space="free")

There are a couple of key parts of this section of code to pay attention to: the call to scale\_fill\_distiller and facet\_grid.

**scale\_fill\_distiller**

scale\_fill\_distiller determines how the numbers in the returns column map onto colors we will use for the squares you see. We use a scale\_fill\_\* function for this because we specified earlier in our ggplot() call that we wanted to map a variable to fill (returns). scale\_fill\_distiller will allow us to specify exactly how our returns column will be represented in color.

Since we are using stock returns, the values will always range from negative to positive, and the full range is of interest to us. Therefore, we will use a *diverging* color palette. type="div" tells scale\_fill\_distiller that we want a color palette that ranges from a dark shade of one color to a dark shade of another – in other words, a diverging palette. As you can see in the calendar heatmap, darker blue represents greater positive daily returns, whereas darker red represents greater negative returns.

palette=7 is the red to blue palette, which is appropriate here since negative returns are typically seen as bad, and positive good (a red to green may have been more appropriate considering this)

na='white' tells scale\_fill\_distiller to fill with white any days with NA values. This will be the first value (January 2) since an NA was introduced when we calculated returns (the first date has no prior date to substract the price from).

limits=c(-.055, .055) sets the boundaries of the color range. Since all values of returns fall within this range, and we want 0 to be white, I’ve set this manually.

labels=percent uses the percent function from the scales package to convert the returns to percent format on the legend.

direction=1 indicates the direction of the colors. direction=-1 would run from blue to red

**facet\_grid**

The facet\_grid function, in ggplot, is what splits the figure up into two year rows and 12 month columns: year~month.

scales="free" allows each month section to show only the week numbers that are present for that month, and not the full 52 of the year repeated for each month.

**Summary**

We now have a calendar heatmap of daily stock returns for the SPY ETF across 2018 and 2019. We can easily see that no weekday patterns are present, that there was a high degree of volatility from October through early January, and that February 5 and 8 were brutal days for this ETF (and the market in general since it tracks the S&P 500).

Calendar heatmaps can be a great way of displaying your data if you want others to be able to easily note precise values at the day level as well as to detect patterns across the week, month, and year in a timeseries. With the flexibility of ggplot and other tidyverse packages, this can be applied to a wide variety of data sets, addressing many different goals.

The code:

library(tidyverse)

library(lubridate)

library(quantmod)

library(scales)

library(zoo)

library(kableExtra)

stock\_xts <- getSymbols("SPY", auto.assign = FALSE, from="2018-01-01")

stock\_df <- fortify.zoo(stock\_xts)

stock\_df %>%

mutate(year = year(Index),

month = month(Index, label = TRUE),

wkday = fct\_relevel(wday(Index, label=TRUE),

c("Mon", "Tue","Wed","Thu","Fri","Sat","Sun")

),

day = day(Index),

wk = format(Index, "%W"),

returns = (.[[5]] - lag(.[[5]]))/lag(.[[5]])) %>%

select(year, month, wkday, day, wk, returns) %>%

ggplot(aes(wk, wkday, fill=returns)) +

geom\_tile(color='black') +

geom\_text(aes(label=day), size=3) +

labs(x='',

y='',

title="SPY") +

scale\_fill\_distiller(type="div"

,palette=7

,na.value = 'white'

,limits=c(-.055, .055)

,labels = percent

,direction=1

) +

theme(panel.background = element\_blank(),

axis.ticks = element\_blank(),

axis.text.x = element\_blank(),

strip.background = element\_rect("grey92")

) +

facet\_grid(year~month, scales="free", space="free")