# Let’s Get Started

library(tidyverse) library(tidyquant) library(timetk)

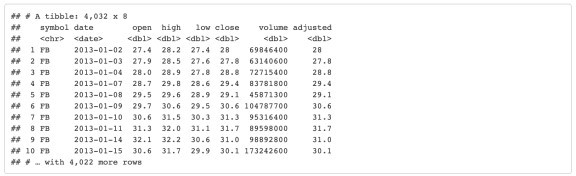
# Data

This tutorial will use the FANG dataset:

Daily

Irregular (missing business holidays and weekends) 4 groups (FB, AMZN, NFLX, and GOOG).

FANG

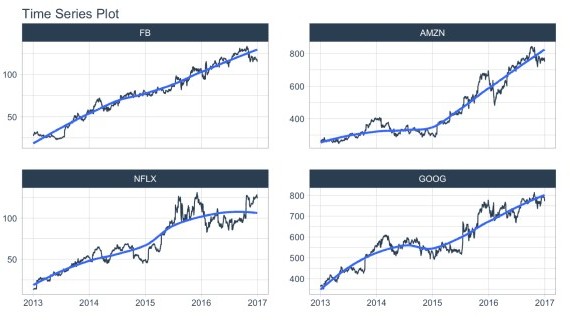


The adjusted column contains the adjusted closing prices for each day.

FANG %>%

group\_by(symbol) %>%

plot\_time\_series(date, adjusted, .facet\_ncol = 2, .interactive = FALSE)

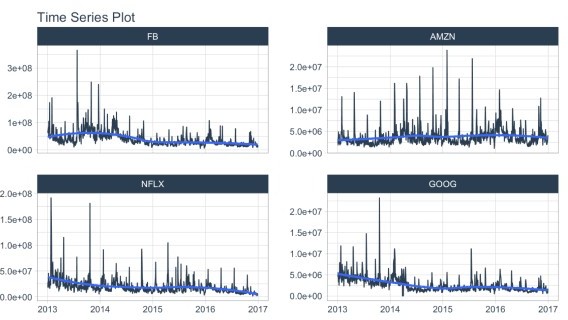


The volume column contains the trade volume (number of times the stock was transacted) for the day.

FANG %>%

group\_by(symbol) %>%

plot\_time\_series(date, volume, .facet\_ncol = 2, .interactive = FALSE)



# Summarize by Time

summarise\_by\_time() aggregates by a period. It’s great for:

Period Aggregation – SUM()

Period Smoothing – AVERAGE(), FIRST(), LAST()

## Period Summarization

Objective: Get the total trade volume by quarter

Use SUM()

Aggregate using .by = "quarter"

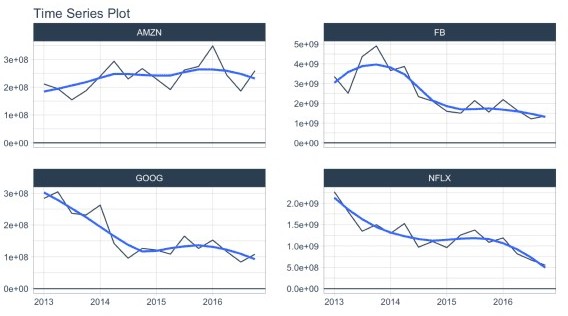
FANG %>%

group\_by(symbol) %>% summarise\_by\_time(

date, .by = "quarter", volume = SUM(volume)

) %>%

plot\_time\_series(date, volume, .facet\_ncol = 2, .interactive = FALSE, .y\_intercept = 0)



## Period Smoothing

Objective: Get the first value in each month

We can use FIRST() to get the first value, which has the effect of reducing the data (i.e. smoothing). We could use AVERAGE() or MEDIAN().

Use the summarization by time: .by = "month" to aggregate by month.

FANG %>%

group\_by(symbol) %>% summarise\_by\_time(

date, .by = "month", adjusted = FIRST(adjusted)

) %>%

plot\_time\_series(date, adjusted, .facet\_ncol = 2, .interactive = FALSE)



# Filter By Time

Used to quickly filter a continuous time range.

## Time Range Filtering

Objective: Get the adjusted stock prices in the 3rd quarter of 2013.

.start\_date = "2013-09": Converts to “2013-09-01

.end\_date = "2013": Converts to “2013-12-31

A more advanced example of filtering using %+time and %-time is shown in *“Padding Data: Low to High Frequency”*.

FANG %>%

group\_by(symbol) %>%

filter\_by\_time(date, "2013-09", "2013") %>% plot\_time\_series(date, adjusted, .facet\_ncol = 2, .interactive =

FALSE)



# Padding Data

Used to fill in (pad) gaps and to go from from low frequency to high frequency. This function uses the awesome padr library for filling and expanding timestamps.

## Fill in Gaps

Objective: Make an irregular series regular.

We will leave padded values as NA.

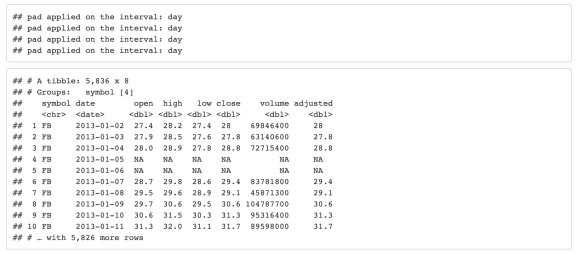
We can add a value using .pad\_value or we can impute using a function like ts\_impute\_vec()

(shown next).

FANG %>%

group\_by(symbol) %>%

pad\_by\_time(date, .by = "auto") # Guesses .by = "day"



## Low to High Frequency

Objective: Go from Daily to Hourly timestamp intervals for 1 month from the start date. Impute the missing values.

.by = "hour" pads from daily to hourly

Imputation of hourly data is accomplished with ts\_impute\_vec(), which performs linear interpolation when period = 1.

Filtering is accomplished using:

“start”: A special keyword that signals the start of a series

FIRST(date) %+time% "1 month": Selecting the first date in the sequence then using a special infix operation, %+time%, called “add time”. In this case I add “1 month”.

FANG %>%

group\_by(symbol) %>% pad\_by\_time(date, .by = "hour") %>%

mutate\_at(vars(open:adjusted), .funs = ts\_impute\_vec, period = 1)

%>%

filter\_by\_time(date, "start", FIRST(date) %+time% "1 month") %>% plot\_time\_series(date, adjusted, .facet\_ncol = 2, .interactive =

FALSE)



# Sliding (Rolling) Calculations

We have a new function, slidify() that turns any function into a sliding (rolling) window function. It takes concepts from tibbletime::rollify() and it improves them with the R package slider.

## Rolling Mean

Objective: Calculate a “centered” simple rolling average with partial window rolling and the start and end windows.

slidify() turns the AVERAGE() function into a rolling average.

# Make the rolling function

roll\_avg\_30 <- slidify(.f = AVERAGE, .period = 30, .align = "center",

.partial = TRUE)

# Apply the rolling function FANG %>%

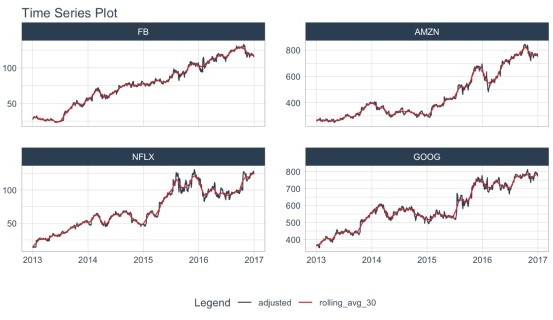
select(symbol, date, adjusted) %>% group\_by(symbol) %>%

# Apply Sliding Function

mutate(rolling\_avg\_30 = roll\_avg\_30(adjusted)) %>% pivot\_longer(cols = c(adjusted, rolling\_avg\_30)) %>% plot\_time\_series(date, value, .color\_var = name,

.facet\_ncol = 2, .smooth = FALSE,

.interactive = FALSE)



For simple rolling calculations (rolling average), we can accomplish this operation faster with

slidify\_vec() – A vectorized rolling function for simple summary rolls (e.g. mean(), sd(), sum(), etc)

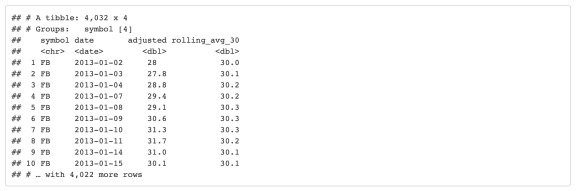
FANG %>%

select(symbol, date, adjusted) %>% group\_by(symbol) %>%

# Apply roll apply Function

mutate(rolling\_avg\_30 = slidify\_vec(adjusted, ~ AVERAGE(.),

.period = 30, .partial = TRUE))



## Rolling Regression

Objective: Calculate a rolling regression.

This is a complex sliding (rolling) calculation that requires multiple columns to be involved.

slidify() is built for this.

Use the multi-variable purrr ..1, ..2, ..3, etc notation to setup a function

# Rolling regressions are easy to implement using `.unlist = FALSE` lm\_roll <- slidify(~ lm(..1 ~ ..2 + ..3), .period = 90,

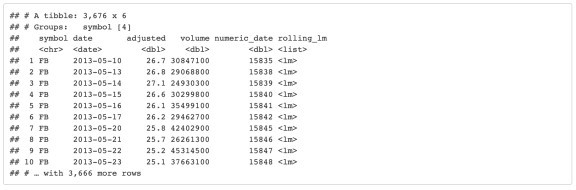
.unlist = FALSE, .align = "right")

FANG %>%

select(symbol, date, adjusted, volume) %>% group\_by(symbol) %>%

mutate(numeric\_date = as.numeric(date)) %>% # Apply rolling regression

mutate(rolling\_lm = lm\_roll(adjusted, volume, numeric\_date)) %>% filter(!is.na(rolling\_lm))



…