

# Time series & financial analysis in the tidyverse

Davis Vaughan  
@dvaughan32

Software Engineer  
@rstudio

# Disclaimer:

Most of what you see here  
is not a product of RStudio...

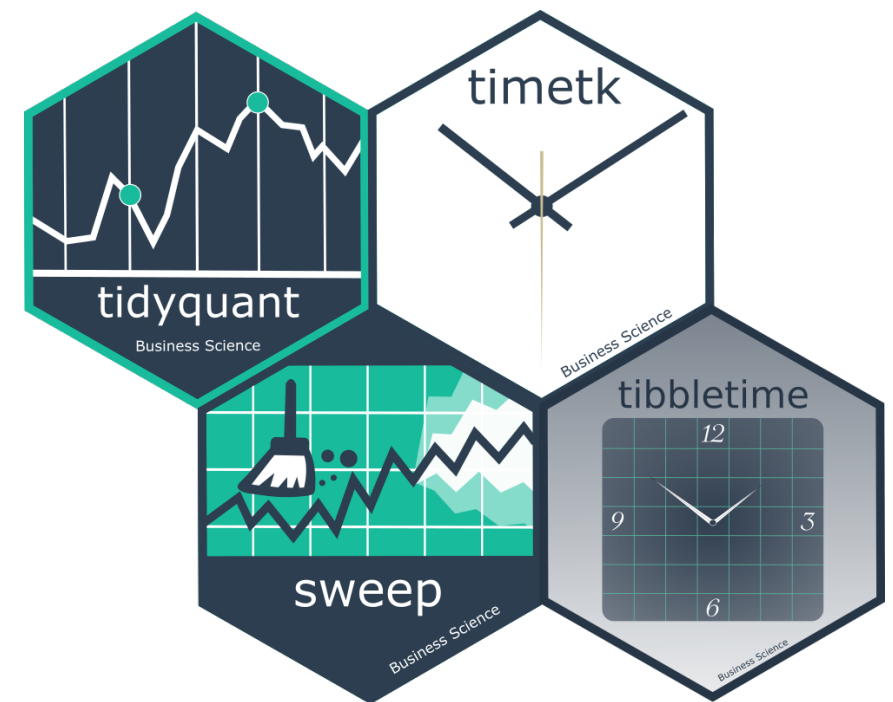
# Disclaimer:

Most of what you see here  
is not a product of RStudio...

...because I just started.

# Who am I?

- ▶ Davis Vaughan
  - ▶ Software engineer @ RStudio
  - ▶ Quantitative finance
    - ▶ Master's @ UNC Charlotte
  - ▶ **Obsessed with making your life easier**



# Agenda: Package overload

# Agenda: Package overload

- ▶ tidyquant

# Agenda: Package overload

- ▶ tidyquant
- ▶ tsibble

# Agenda: Package overload

- ▶ `tidyquant`
- ▶ `tsibble`
- ▶ `rsample` + `furrr`



# The current state of the world

xts

tibble

Native time-index support

Specialized (& fast)  
time-based  
manipulation

Homogeneous data  
(built on matrices)

Packages for financial  
analysis (quantmod,  
PerformanceAnalytics, ...)

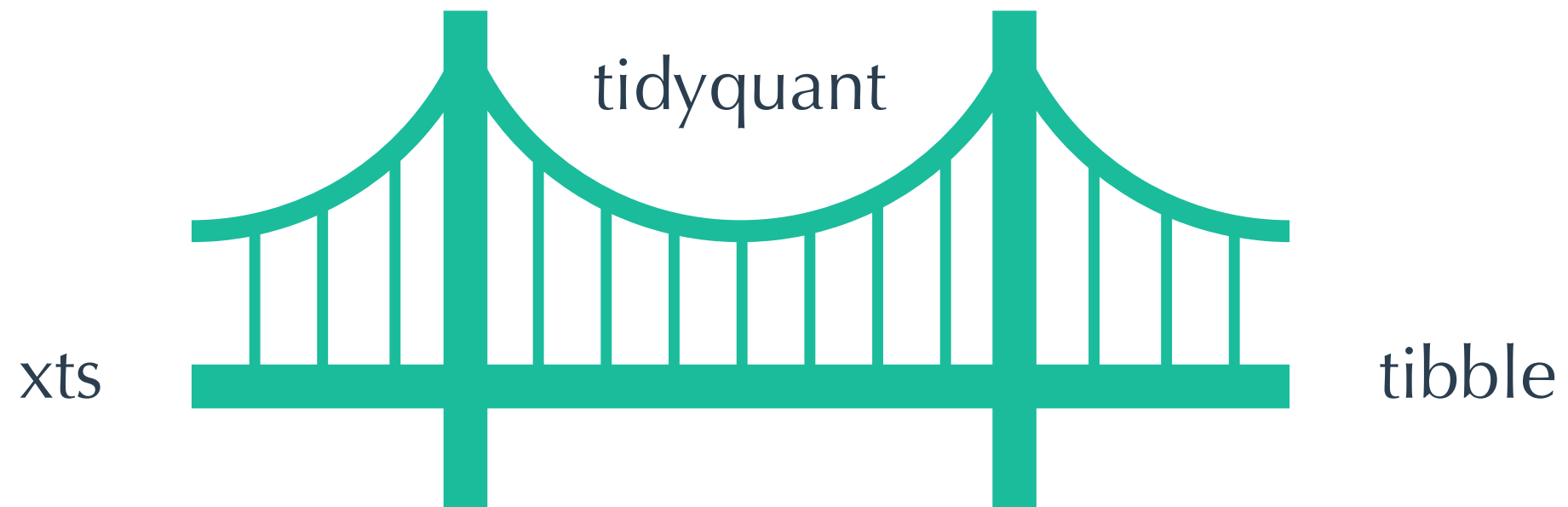
Powerful generalized  
data manipulation

Grouped analysis

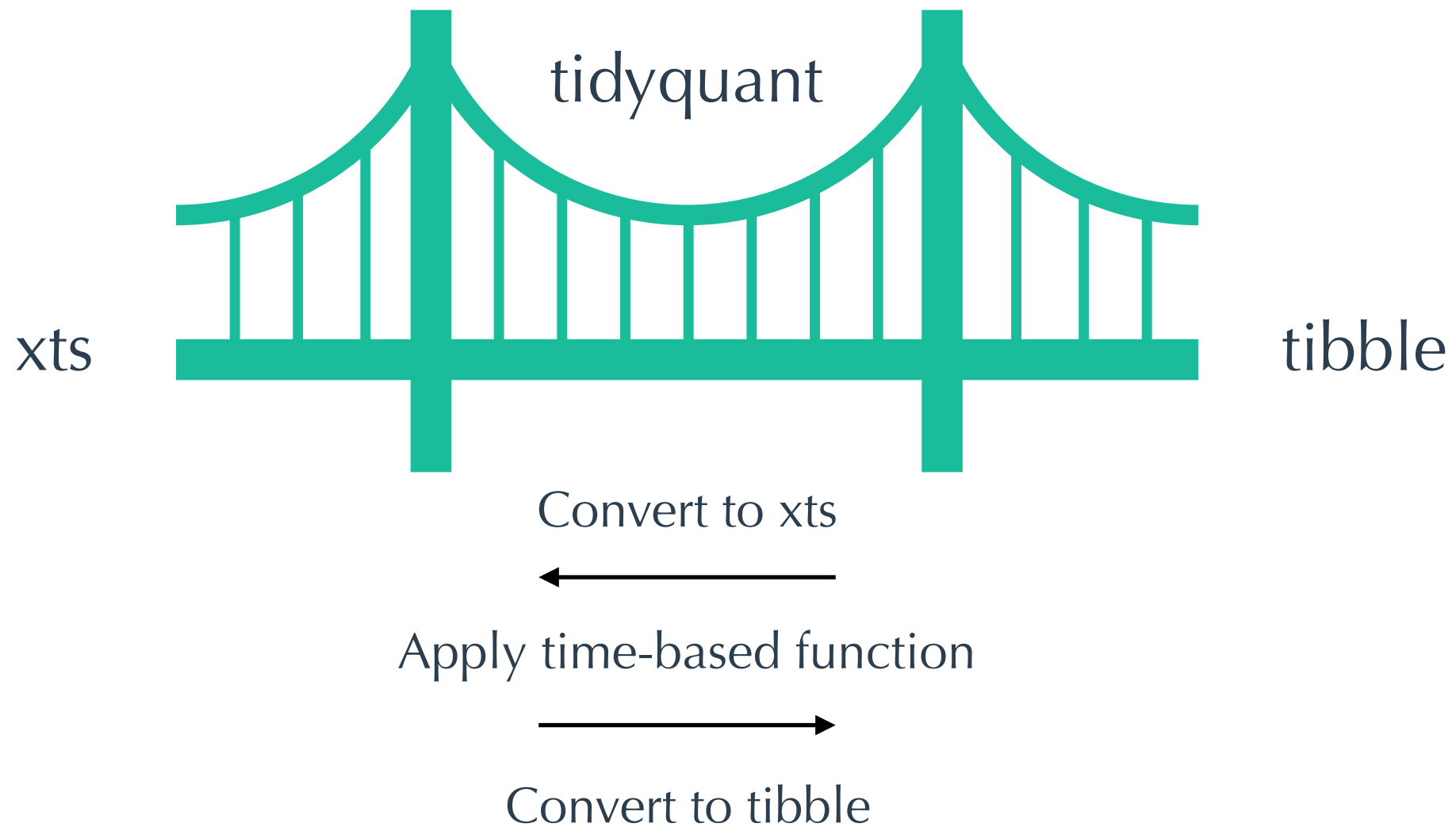
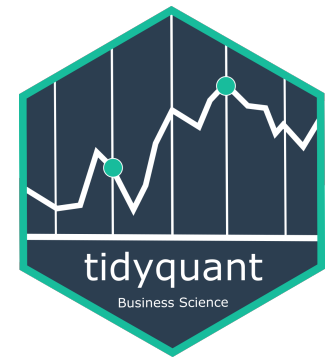
Readability > Performance

Heterogeneous data +  
list-column support

# tidyquant



# tidyquant

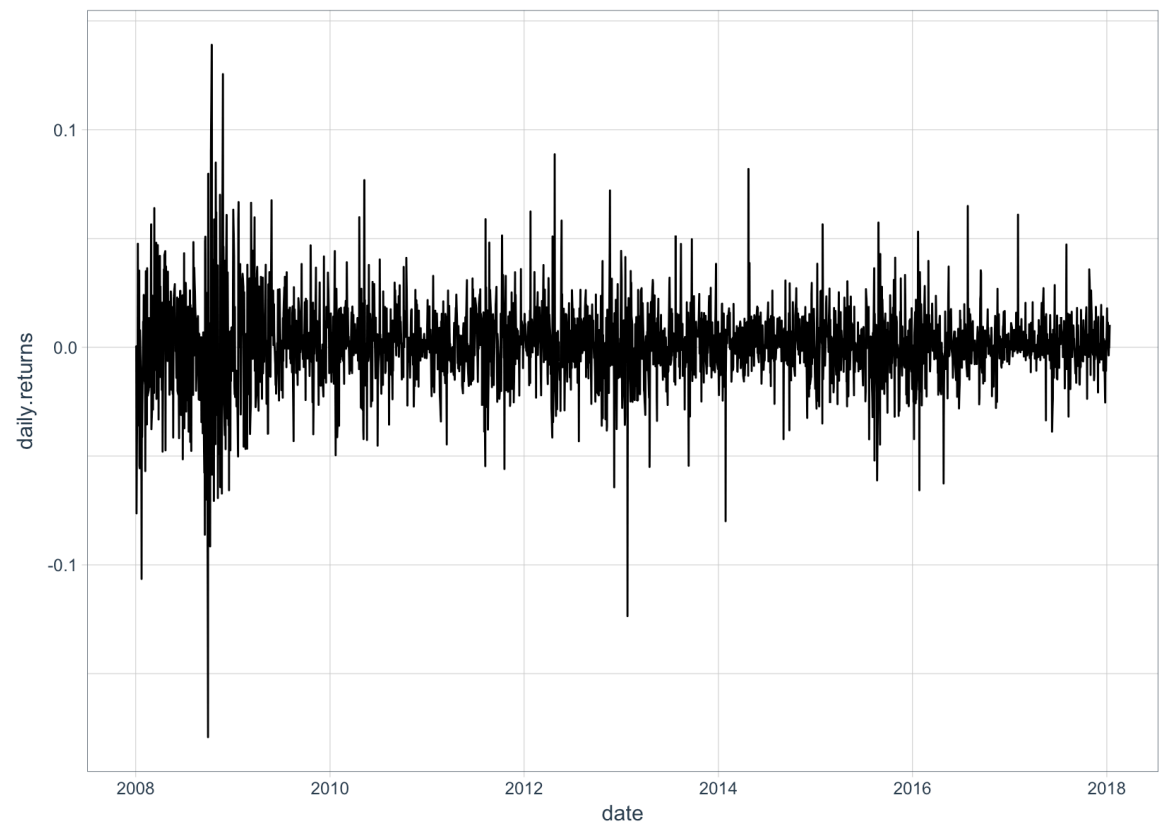


# tidyquant



```
tq_get("AAPL") %>%  
  tq_mutate(select = adjusted, mutate_fun = dailyReturn) %>%  
  ggplot(aes(x = date, y = daily.returns)) +  
  geom_line() +  
  theme_tq()
```

- ▶ Quickly pull financial data as a tibble
- ▶ Apply any xts, quantmod, TTR, and PerformanceAnalytics function
- ▶ Pipe the result straight into other tidyverse packages





**Mara Averick**

@dataandme

Following

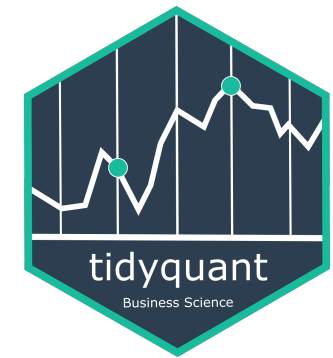


Replying to [@mdancho84](#)

What an overly gratifying activity. Like, pulling that stock data and plotting it was a minute-long affair (with thinking and getting distracted)! 🤖 package \*and\* tutorial!

4:18 PM - 4 Jan 2018

# Lots of functionality for free



```
> tq_mutate_fun_options()
```

## \$zoo

[1]	" <b>rollapply</b> "	"rollapplyr"	"rollmax"	"rollmax.default"	"rollmaxr"	" <b>rollmean</b> "
[7]	"rollmean.default"	"rollmeanr"	"rollmedian"	"rollmedian.default"	"rollmedianr"	"rollsum"
[13]	"rollsum.default"	"rollsumr"				

## \$xts

[1]	"apply.daily"	" <b>apply.monthly</b> "	"apply.quarterly"	"apply.weekly"	"apply.yearly"	"diff.xts"	"lag.xts"
[8]	" <b>period.apply</b> "	"period.max"	"period.min"	"period.prod"	"period.sum"	"periodicity"	"to_period"
[15]	"to.daily"	"to.hourly"	"to.minutes"	"to.minutes10"	"to.minutes15"	"to.minutes3"	"to.minutes30"
[22]	"to.minutes5"	"to.monthly"	"to.period"	"to.quarterly"	"to.weekly"	"to.yearly"	

## \$quantmod

[1]	"allReturns"	"annualReturn"	"ClCl"	" <b>dailyReturn</b> "	"Delt"	"HiCl"	"Lag"
[8]	"LoCl"	"LoHi"	"monthlyReturn"	"Next"	"OpCl"	"OpHi"	"OpLo"
[15]	"OpOp"	"periodReturn"	"quarterlyReturn"	"seriesAccel"	"seriesDecel"	"seriesDecr"	"seriesHi"
[22]	"seriesIncr"	"seriesLo"	"weeklyReturn"	"yearlyReturn"			

## \$TTR

[1]	"adjRatios"	"ADX"	"ALMA"	"aroon"	"ATR"	"BBands"
[7]	"CCI"	"chaikinAD"	"chaikinVolatility"	"CLV"	"CMF"	"CMO"
[13]	"DEMA"	"DonchianChannel"	"DPO"	"DVI"	"EMA"	"EMV"
[19]	"EVWMA"	"GMMA"	"growth"	"HMA"	"KST"	"lags"
[25]	"MACD"	"MFI"	"momentum"	"OBV"	"PBands"	"ROC"
[31]	"rollSFM"	"RSI"	"runCor"	"runCov"	"runMAD"	"runMax"
[37]	"runMean"	"runMedian"	"runMin"	"runPercentRank"	"runSD"	"runSum"
[43]	"runVar"	"SAR"	"SMA"	"SMI"	"SNR"	"stoch"
[49]	"TDI"	"TRIX"	"ultimateOscillator"	"VHF"	"VMA"	"volatility"
[55]	"VWAP"	"VWMA"	"wilderSum"	"williamsAD"	"WMA"	"WPR"
[61]	"ZigZag"	"ZLEMA"				

## \$PerformanceAnalytics

[1]	"Return.annualized"	"Return.annualized.excess"	"Return.clean"	"Return.cumulative"
[5]	"Return.excess"	"Return.Geltner"	"zerofill"	

# What are we missing?



Conversion is **slow**

**Limited** in functionality

**Indirectly** using both the tidyverse and xts

No support for a **time-based index**

Wouldn't it be nice to have a **tibble**  
with **time-index support**,  
**fully leveraging** the tools of the tidyverse?



ts

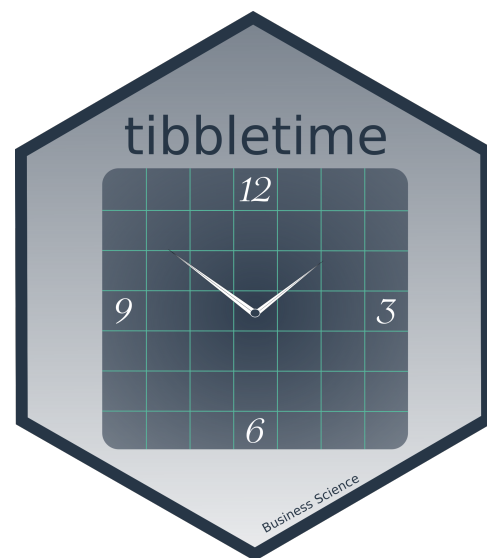
+

tibble

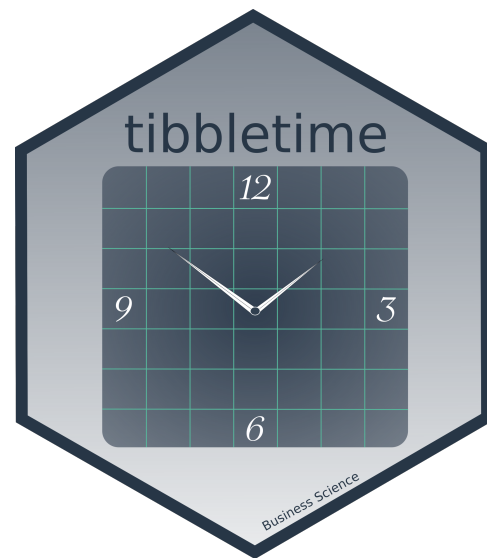
=



# A little history



# A little history



Earo Wang  
@earowang

<https://github.com/earowang>

# What?



A **tsibble** consists of a time **index**, **key**  
and other measured **variables** in a data-centric format,  
which is built on top of the **tibble**.

# What?



Utilizes extra  
knowledge



A **tsibble** consists of a time **index**, **key**  
and other measured **variables** in a data-centric format,  
which is built on top of the **tibble**.

# What?



Utilizes extra  
knowledge



A **tsibble** consists of a time **index**, **key**  
and other measured **variables** in a data-centric format,  
which is built on top of the **tibble**.



Underlying  
data type  
is the same

# Creation



```
as_tsibble(df, key = id(Key), index = Date)
```

Key	Date	Col1	Col2	Col3

# Why?

1. Perform time-based manipulations on tibbles
2. Work more naturally with time series in the tidyverse



# San Diego Airbnb bookings



```
airbnb
```

```
# A tsibble: 9,111 x 5 [1s]
```

```
# Key:           room_id [9,111]
```

	room_id	last_modified	price	latitude	longitude
	<int>	<dtm>	<dbl>	<dbl>	<dbl>
1	6	2017-07-11 18:08:36	169	32.8	-117.
2	5570	2017-07-11 20:01:30	205	32.8	-117.
3	9731	2017-07-11 15:51:35	65	32.9	-117.
4	14668	2017-07-11 15:09:38	55	32.9	-117.
5	37149	2017-07-11 15:09:56	55	32.8	-117.
6	38245	2017-07-11 15:18:00	50	32.7	-117.
7	39516	2017-07-11 17:19:11	70	32.7	-117.
8	45429	2017-07-11 18:18:08	160	32.7	-117.
9	54001	2017-07-11 16:31:55	125	32.8	-117.
10	62274	2017-07-11 15:49:21	69	32.8	-117.

```
# ... with 9,101 more rows
```

# A new way to group



```
index_by(airbnb, two_hourly = floor_date(last_modified, "2 hour"))
```

last\_modified  
<dtm>

1	2017-07-11	15:09:38
2	2017-07-11	15:09:56
3	2017-07-11	15:18:00
4	2017-07-11	15:49:21
5	2017-07-11	15:51:35
6	2017-07-11	16:31:55
7	2017-07-11	17:19:11
8	2017-07-11	18:08:36
9	2017-07-11	18:18:08
10	2017-07-11	20:01:30



two\_hourly  
<dtm>

1	2017-07-11	14:00:00
2	2017-07-11	14:00:00
3	2017-07-11	14:00:00
4	2017-07-11	14:00:00
5	2017-07-11	14:00:00
6	2017-07-11	16:00:00
7	2017-07-11	16:00:00
8	2017-07-11	18:00:00
9	2017-07-11	18:00:00
10	2017-07-11	20:00:00

# A new way to group



```
airbnb %>%  
  index_by(  
    two_hourly = floor_date(last_modified, "2 hour")  
  ) %>%  
  summarise(median_price = median(price))
```

```
# A tsibble: 8 x 2 [2h]
```

	two_hourly <dtm>	median_price <dbl>
1	2017-07-11 14:00:00 [14-16)	55
2	2017-07-11 16:00:00 [16-18)	100
3	2017-07-11 18:00:00 [18-20)	199
4	2017-07-11 20:00:00 [20-22)	450
5	2017-07-11 22:00:00 [22-00)	152
6	2017-07-12 00:00:00 [00-02)	285
7	2017-07-12 02:00:00 [02-04)	882
8	2017-07-12 04:00:00 [04-06)	40

# A new way to group



```
airbnb %>%  
  index_by(  
    two_hourly = ceiling_date(last_modified, "2 hour")  
  ) %>%  
  summarise(median_price = median(price))
```

```
# A tsibble: 8 x 2 [2h]
```

	two_hourly <dtm>	median_price <dbl>
1	2017-07-11 16:00:00 [14-16)	55
2	2017-07-11 18:00:00 [16-18)	100
3	2017-07-11 20:00:00 [18-20)	199
4	2017-07-11 22:00:00 [20-22)	450
5	2017-07-11 00:00:00 [22-00)	152
6	2017-07-12 02:00:00 [00-02)	285
7	2017-07-12 04:00:00 [02-04)	882
8	2017-07-12 06:00:00 [04-06)	40

# The possibilities are endless



```
# Development versions of both
library(ggmap)
library(gganimate)

airbnb_plot <- airbnb %>%

  # Index by hour
  index_by(hourly = floor_date(last_modified, "hour")) %>%

  # Throw out a few outliers
  filter(
    between(price, quantile(price, .05), quantile(price, .95))
  ) %>%

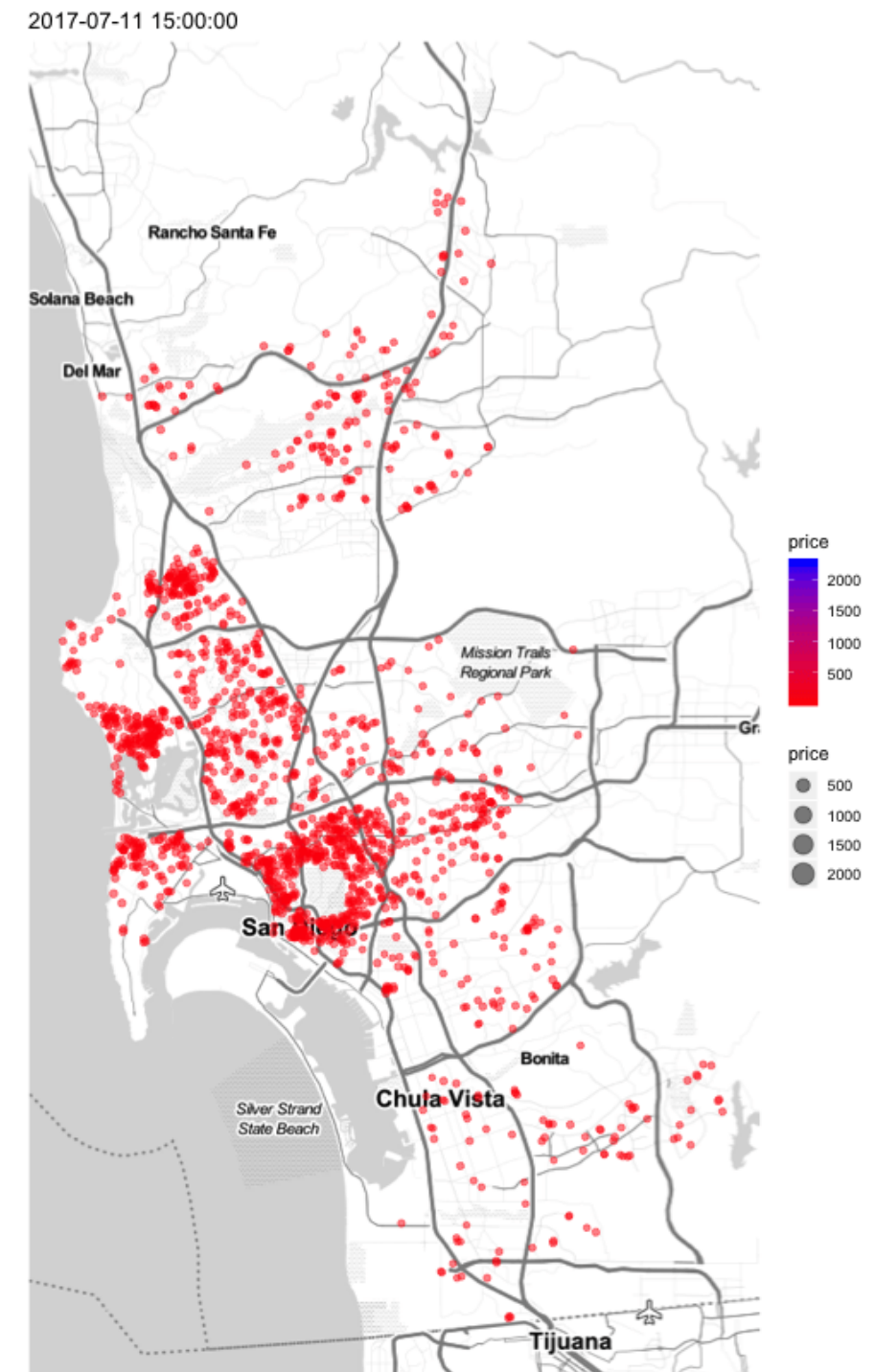
  # Map and animate
  qmplot(longitude, latitude, data = ., geom = "blank") +

  geom_point(
    aes(color = price, size = price),
    alpha = .5
  ) +

  scale_color_continuous(low = "red", high = "blue") +

  transition_manual(hourly) +
  labs(title = "{current_frame}")

animate(airbnb_plot)
```



# The possibilities are endless



```
# Development versions of both
library(ggmap)
library(gganimate)

airbnb_plot <- airbnb %>%

  # Index by hour
  index_by(hourly = floor_date(last_modified, "hour")) %>%

  # Throw out a few outliers
  filter(
    between(price, quantile(price, .05), quantile(price, .95))
  ) %>%

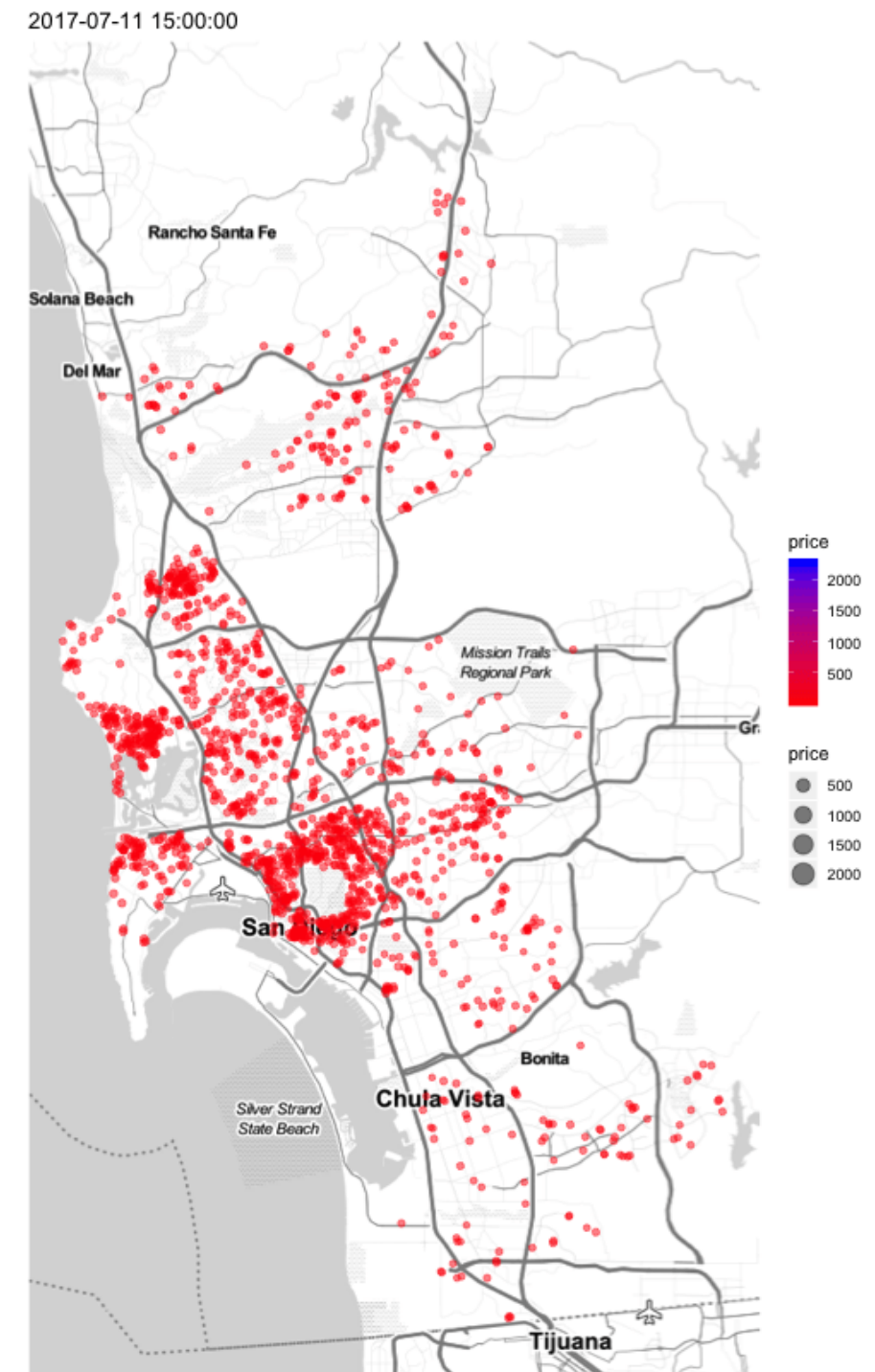
  # Map and animate
  qmplot(longitude, latitude, data = ., geom = "blank") +

  geom_point(
    aes(color = price, size = price),
    alpha = .5
  ) +

  scale_color_continuous(low = "red", high = "blue") +

  transition_manual(hourly) +
  labs(title = "{current_frame}")

animate(airbnb_plot)
```



# Extra functionality



*Multi-class support*

**Date**

**Posixct**

**yearmonth**

**yearquarter**

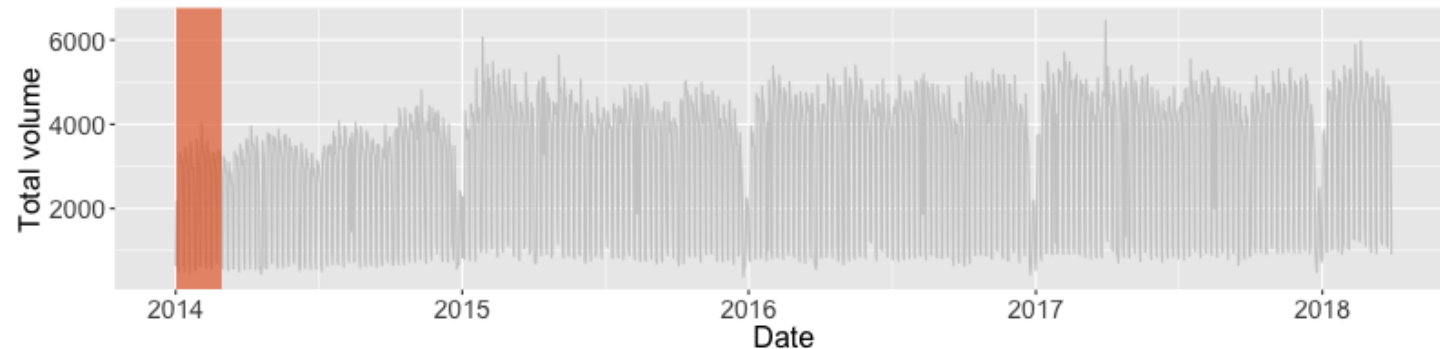
**hms**



# A family of window functions



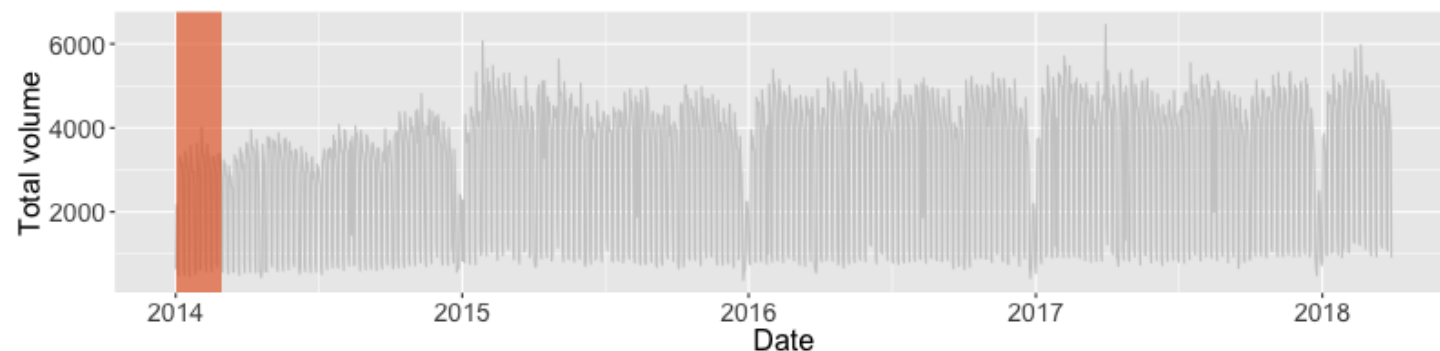
`slide()`, `slide2()`, `pslide()`



1. purrr-like syntax

- `~ .x + .y`

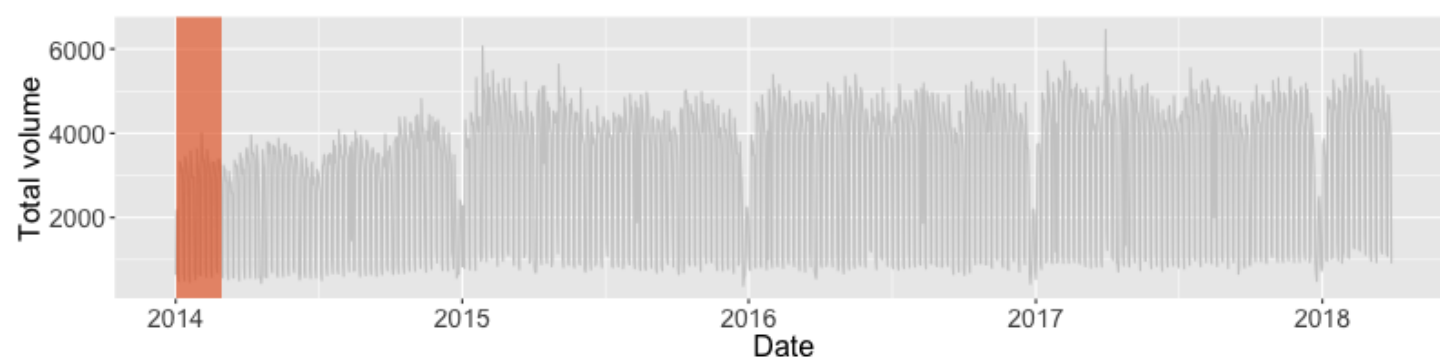
`tile()`, `tile2()`, `ptile()`



2. Type stable variants

- default = list
- `*_dbl()`
- `*_int()`
- `*_lgl()`
- ...

`stretch()`, `stretch2()`, `pstretch()`

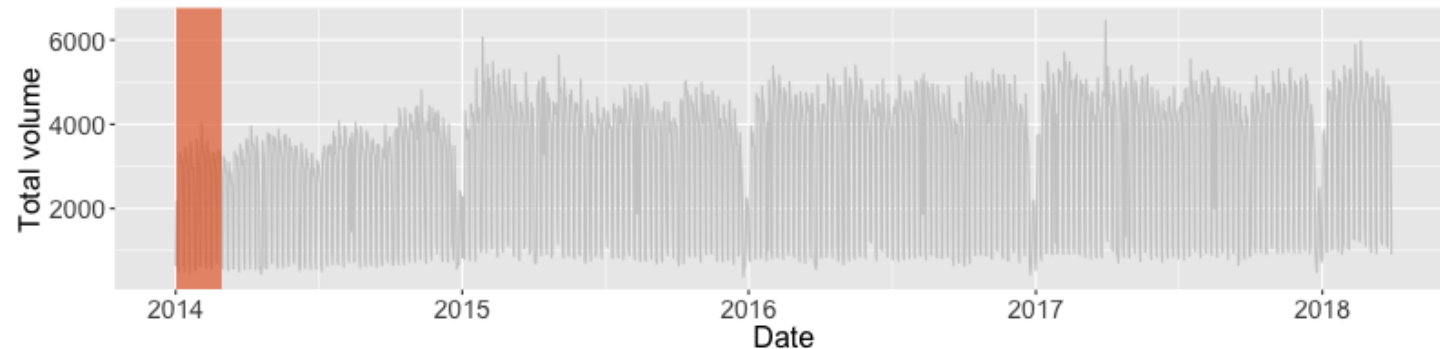




# A family of window functions



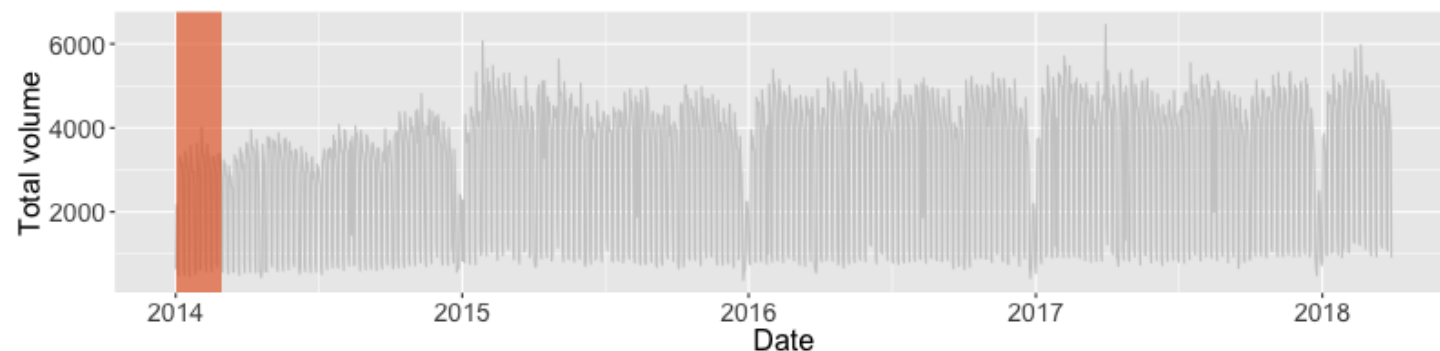
`slide()`, `slide2()`, `pslide()`



1. purrr-like syntax

- `~ .x + .y`

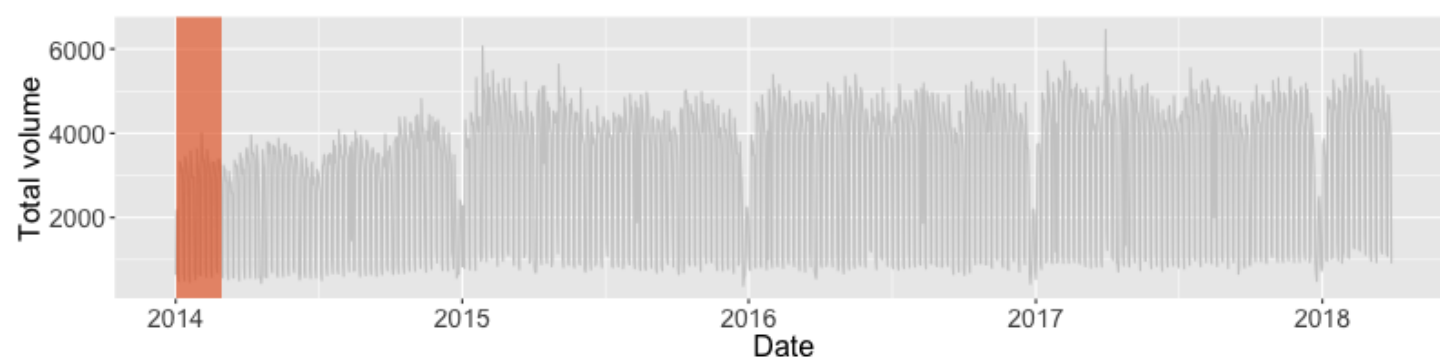
`tile()`, `tile2()`, `ptile()`



2. Type stable variants

- default = list
- `*_dbl()`
- `*_int()`
- `*_lgl()`
- ...

`stretch()`, `stretch2()`, `pstretch()`



# Sliiiiiide to the left



## **slide()**

*Sliding window calculations without overlapping observations*



# Sliiiiiide to the left



## **slide()**

*Sliding window calculations without overlapping observations*





# Sliiiiide to the left



```
> FB
# A tibble: 1,008 x 3
  date          adjusted volume
  <date>         <dbl>    <dbl>
1 2013-01-02      28.0  69846400
2 2013-01-03      27.8  63140600
3 2013-01-04      28.8  72715400
4 2013-01-07      29.4  83781800
5 2013-01-08      29.1  45871300
6 2013-01-09      30.6 104787700
7 2013-01-10      31.3  95316400
8 2013-01-11      31.7  89598000
9 2013-01-14      31.0  98892800
10 2013-01-15      30.1 173242600
# ... with 998 more rows
```

# Rolling averages



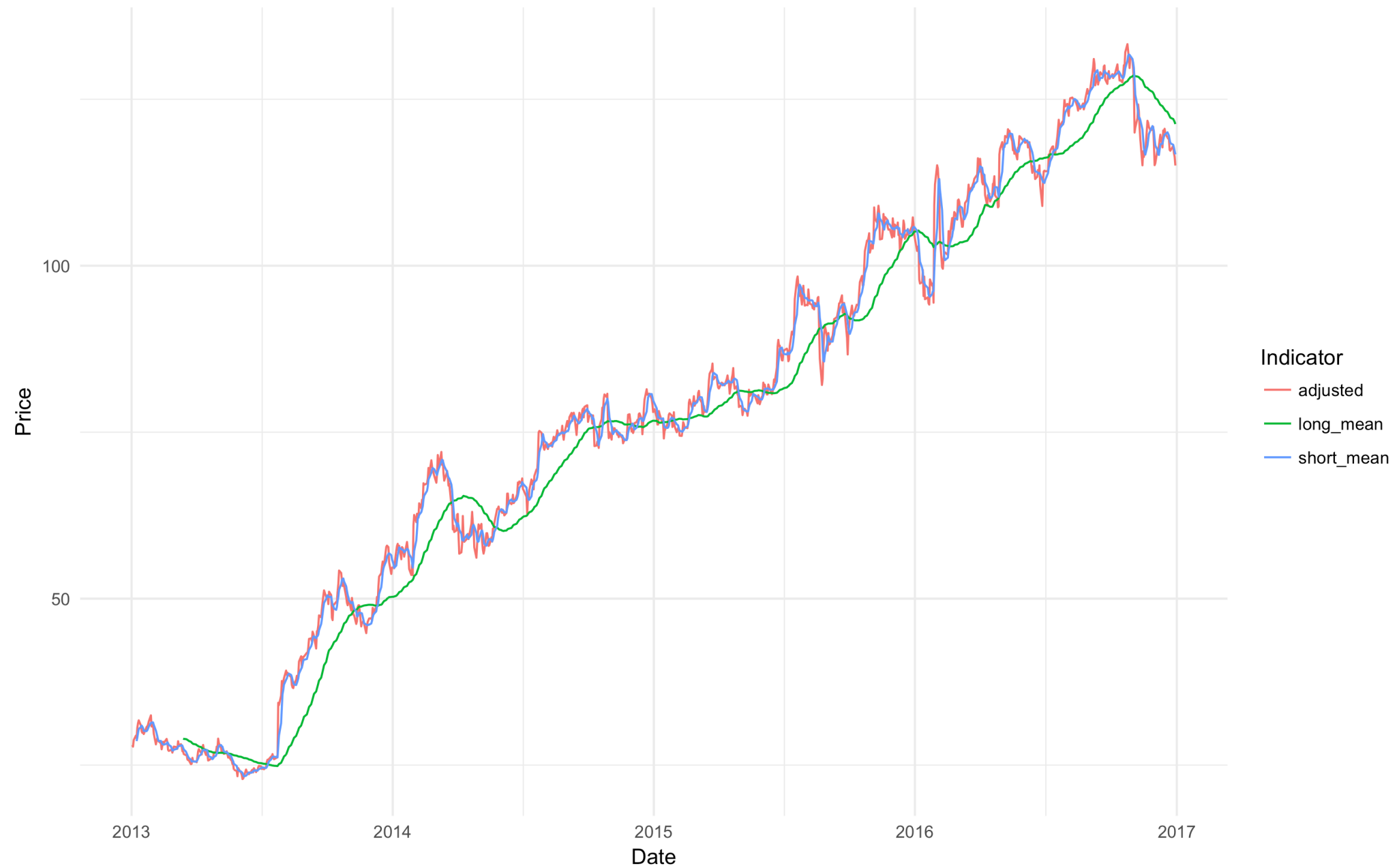
```
mutate(FB,  
  short_mean = slide_dbl(adjusted, ~mean(.x, na.rm = TRUE), .size = 5),  
  long_mean  = slide_dbl(adjusted, ~mean(.x, na.rm = TRUE), .size = 50),  
)
```

```
# A tibble: 1,008 x 4  
  date          adjusted short_mean long_mean  
  <date>         <dbl>    <dbl>    <dbl>  
1 2013-01-02      28.0      NA      NA  
2 2013-01-03      27.8      NA      NA  
3 2013-01-04      28.8      NA      NA  
4 2013-01-07      29.4      NA      NA  
5 2013-01-08      29.1     28.6     NA  
6 2013-01-09      30.6     29.1     NA  
7 2013-01-10      31.3     29.8     NA  
8 2013-01-11      31.7     30.4     NA  
9 2013-01-14      31.0     30.7     NA  
10 2013-01-15      30.1     30.9     NA
```

# Rolling averages



FB Adjusted stock price with long/short term moving averages



# Rolling linear models



```
FB_model ← FB %>%  
  mutate(  
    lag_volume = lag(volume),  
    model = slide2(  
      .x = lag_volume,  
      .y = adjusted,  
      .f = ~ lm(.y ~ .x),  
      .size = 5)  
  )
```

# Rolling linear models



Rolling equivalent  
of `map2()`.

```
FB_model ← FB %>%  
  mutate(  
    lag_volume = lag(volume),  
    model = slide2(  
      .x = lag_volume,  
      .y = adjusted,  
      .f = ~ lm(.y ~ .x),  
      .size = 5)  
  )
```



# Rolling linear models



```
FB_model ← FB %>%  
  mutate(  
    lag_volume = lag(volume),  
    model = slide2(  
      .x = lag_volume,  
      .y = adjusted,  
      .f = ~ lm(.y ~ .x),  
      .size = 5)  
  )
```

Rolling equivalent  
of map2( ).

.f can be  
anything.

# Rolling linear models



```
FB_model ← FB %>%  
  mutate(  
    lag_volume = lag(volume),  
    model = slide2(  
      .x = lag_volume,  
      .y = adjusted,  
      .f = ~lm(.y ~ .x),  
      .size = 5)  
  )
```

```
# A tibble: 1,008 x 5  
  date      adjusted volume lag_volume model  
  <date>      <dbl>   <dbl>    <dbl>  <list>  
1 2013-01-02    28.0 69846400      NA <lgl [1]>  
2 2013-01-03    27.8 63140600 69846400 <lgl [1]>  
3 2013-01-04    28.8 72715400 63140600 <lgl [1]>  
4 2013-01-07    29.4 83781800 72715400 <lgl [1]>  
5 2013-01-08    29.1 45871300 83781800 <S3: lm>  
6 2013-01-09    30.6 104787700 45871300 <S3: lm>  
7 2013-01-10    31.3 95316400 104787700 <S3: lm>  
8 2013-01-11    31.7 89598000 95316400 <S3: lm>  
9 2013-01-14    31.0 98892800 89598000 <S3: lm>  
10 2013-01-15    30.1 173242600 98892800 <S3: lm>  
# ... with 998 more rows
```

# Rolling linear models



```
FB_model ← FB %>%  
  mutate(  
    lag_volume = lag(volume),  
    model = slide2(  
      .x = lag_volume,  
      .y = adjusted,  
      .f = ~lm(.y ~ .x),  
      .size = 5)  
  )
```

```
# A tibble: 1,008 x 5  
  date      adjusted volume lag_volume model  
  <date>      <dbl>   <dbl>    <dbl>  <list>  
1 2013-01-02    28.0 69846400      NA <lgl [1]>  
2 2013-01-03    27.8 63140600 69846400 <lgl [1]>  
3 2013-01-04    28.8 72715400 63140600 <lgl [1]>  
4 2013-01-07    29.4 83781800 72715400 <lgl [1]>  
5 2013-01-08    29.1 45871300 83781800 <S3: lm>  
6 2013-01-09    30.6 104787700 45871300 <S3: lm>  
7 2013-01-10    31.3 95316400 104787700 <S3: lm>  
8 2013-01-11    31.7 89598000 95316400 <S3: lm>  
9 2013-01-14    31.0 98892800 89598000 <S3: lm>  
10 2013-01-15    30.1 173242600 98892800 <S3: lm>  
# ... with 998 more rows
```

First 4  
are NA



Wouldn't it be nice to have a **tibble**  
with **time-index support**,  
**fully leveraging** the tools of the tidyverse?

Built on top of  
tibbles

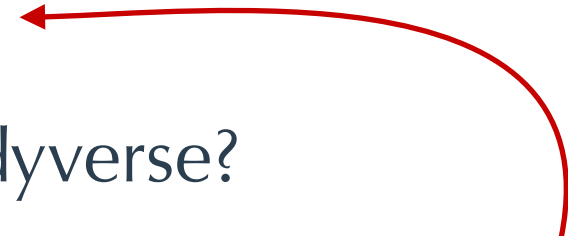


Wouldn't it be nice to have a **tibble**  
with **time-index support**,  
**fully leveraging** the tools of the tidyverse?

Built on top of  
tibbles

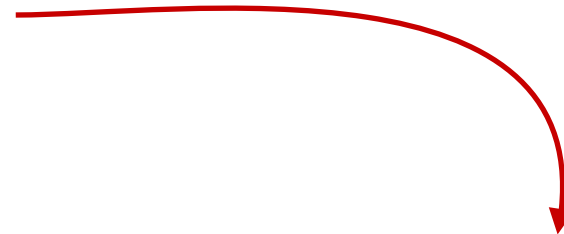


Wouldn't it be nice to have a **tibble**  
with **time-index support**,  
**fully leveraging** the tools of the tidyverse?



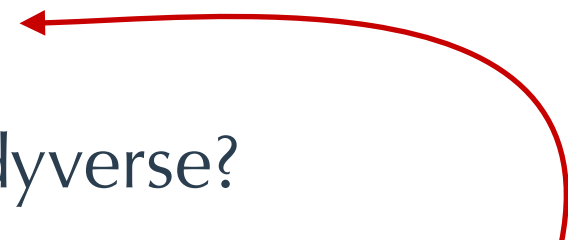
Learns about the  
index at creation

Built on top of  
tibbles



Wouldn't it be nice to have a **tibble**

with **time-index support**,



**fully leveraging** the tools of the tidyverse?

Learns about the  
index at creation



Seamless  
integration with  
the tidyverse

*We now have*

~~Wouldn't it be nice to have a **tibble**~~

with **time-index support**,

**fully leveraging** the tools of the tidyverse!



Future plans  
(aka my hopes and dreams)

# Facebook, Amazon, Netflix, Google

```
FANG_time ← FANG %>%  
  group_by(symbol) %>%  
  as_tsibble(  
    key = id(symbol),  
    index = date  
  )
```

```
slice(FANG_time, 1:2)
```

```
# A tsibble: 4,032 x 8 [1D]  
# Key:      symbol [4]  
# Groups:   symbol [4]  
  symbol    date      adjusted  
  <chr>    <date>      <dbl>  
1 AMZN    2013-01-02    257  
2 AMZN    2013-01-03    258  
3 FB      2013-01-02    28.0  
4 FB      2013-01-03    27.8  
5 GOOG    2013-01-02    361  
6 GOOG    2013-01-03    361  
7 NFLX    2013-01-02    13.1  
8 NFLX    2013-01-03    13.8
```

# Calculate returns

```
FANG_time %>%
```

```
  index_by(yearly = floor_date(date, "year")) %>%
```

```
  calculate_return(adjusted)
```

```
# A tibble: 16 x 7 [1Y]
```

```
# Key:      symbol [4]
```

```
# Groups:   symbol [4]
```

	symbol <chr>	date <date>	adjusted <dbl>	yearly <date>	adjusted_return <dbl>
1	FB	2013-12-31	54.7	2013-01-01	0.952
2	FB	2014-12-31	78.0	2014-01-01	0.428
3	FB	2015-12-31	105.	2015-01-01	0.341
4	FB	2016-12-30	115.	2016-01-01	0.0993
5	AMZN	2013-12-31	399.	2013-01-01	0.550
6	AMZN	2014-12-31	310.	2014-01-01	-0.222
7	AMZN	2015-12-31	676.	2015-01-01	1.18
8	AMZN	2016-12-30	750.	2016-01-01	0.109
9	NFLX	2013-12-31	52.6	2013-01-01	3.00
10	NFLX	2014-12-31	48.8	2014-01-01	-0.0721

# Calculate returns

```
FANG_time %>%
```

```
  index_by(yearly = floor_date(date, "year")) %>%
```

```
  calculate_return(adjusted) %>%
```

```
  mutate(drawdown = drawdown(adjusted_return),
```

```
         cum_ret   = cumulative_return(adjusted_return))
```

```
# A tibble: 16 x 7 [1Y]
```

```
# Key:      symbol [4]
```

```
# Groups:   symbol [4]
```

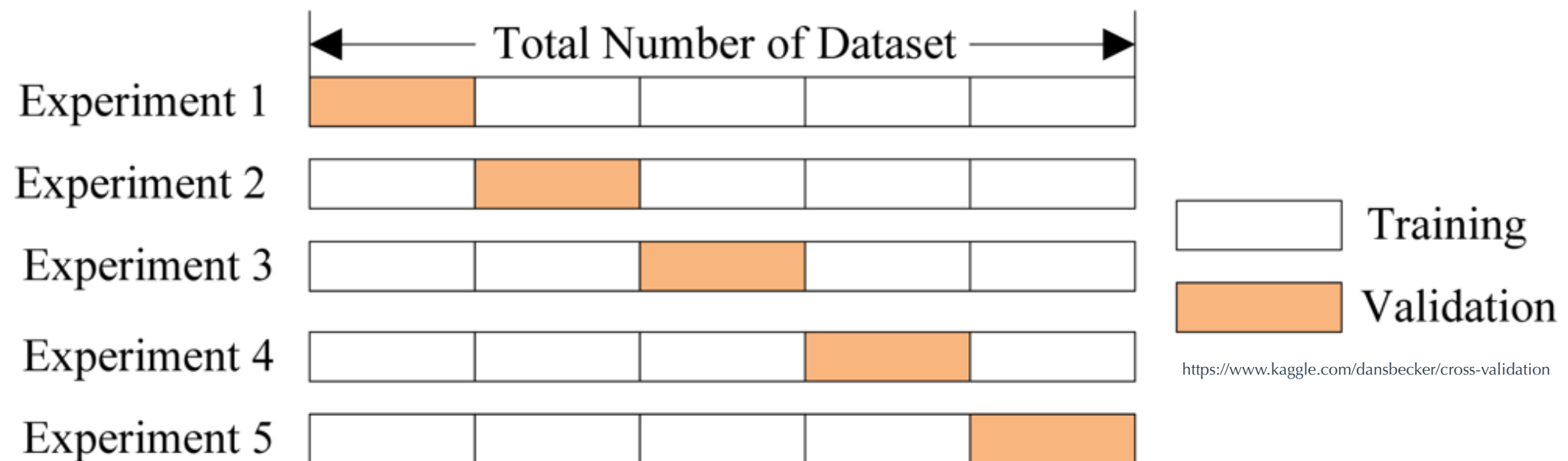
	symbol <chr>	date <date>	adjusted <dbl>	yearly <date>	adjusted_return <dbl>	drawdown <dbl>	cum_ret <dbl>
1	FB	2013-12-31	54.7	2013-01-01	0.952	0	0.952
2	FB	2014-12-31	78.0	2014-01-01	0.428	0	1.79
3	FB	2015-12-31	105.	2015-01-01	0.341	0	2.74
4	FB	2016-12-30	115.	2016-01-01	0.0993	0	3.11
5	AMZN	2013-12-31	399.	2013-01-01	0.550	0	0.550
6	AMZN	2014-12-31	310.	2014-01-01	-0.222	-0.222	0.206
7	AMZN	2015-12-31	676.	2015-01-01	1.18	0	1.63
8	AMZN	2016-12-30	750.	2016-01-01	0.109	0	1.91
9	NFLX	2013-12-31	52.6	2013-01-01	3.00	0	3.00
10	NFLX	2014-12-31	48.8	2014-01-01	-0.0721	-0.0721	2.71

**Switching gears**

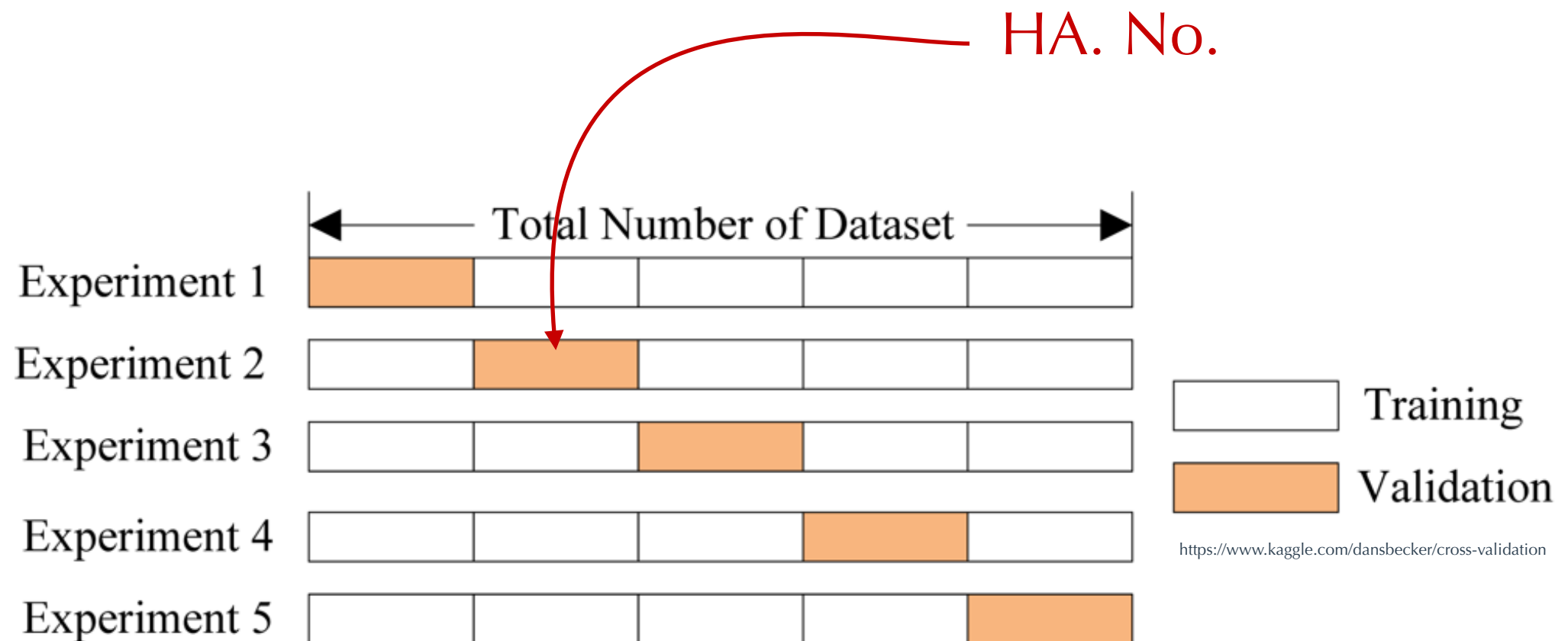
## **Problem:**

You want to perform **cross-validation** in  
R with **time series**.

# Cross-validation

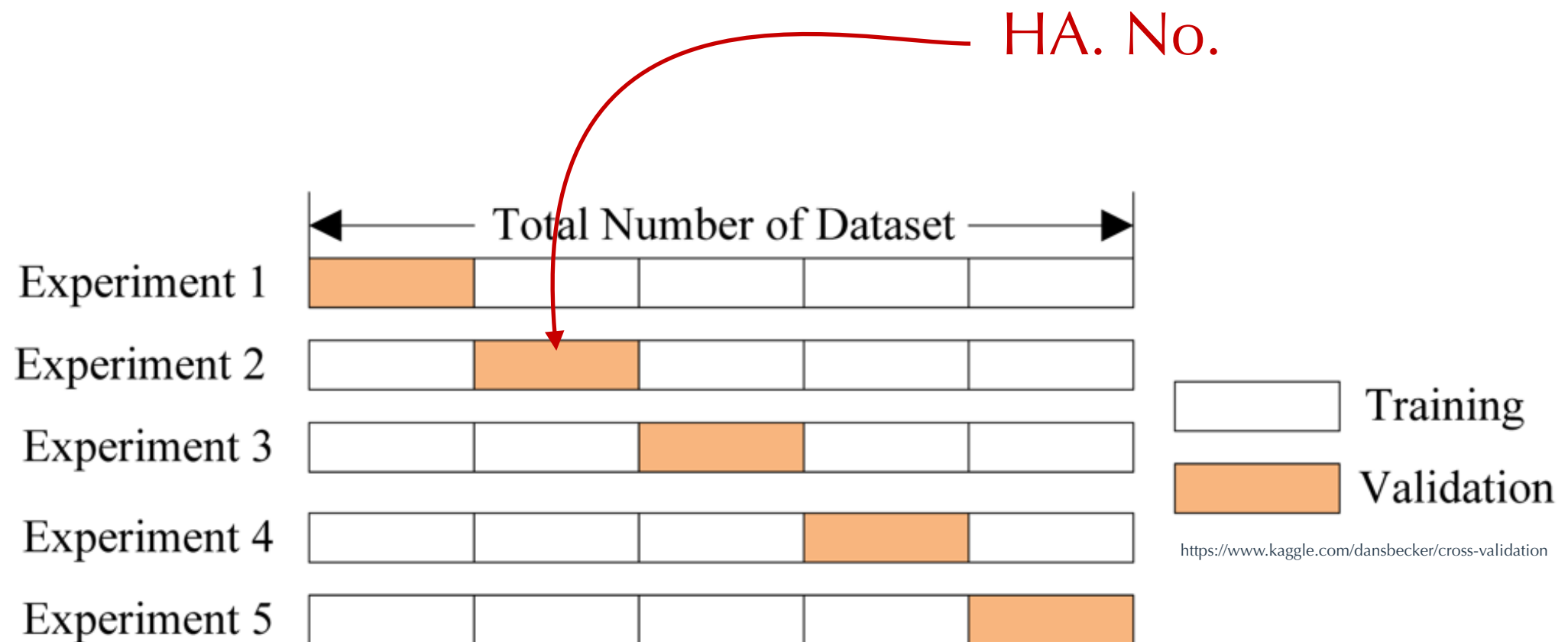


# Cross-validation





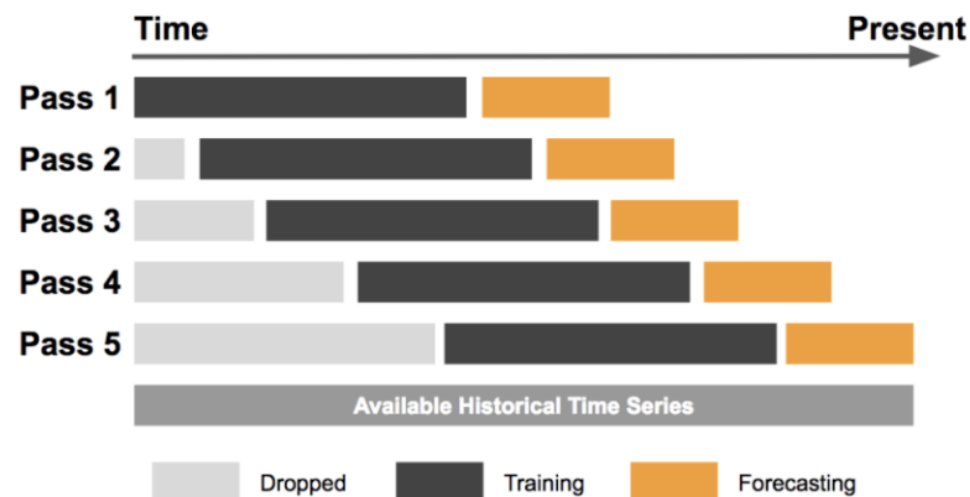
# Cross-validation



This makes sense!

# Cross-validation for ~~the shit we do~~ time series

## Sliding Window

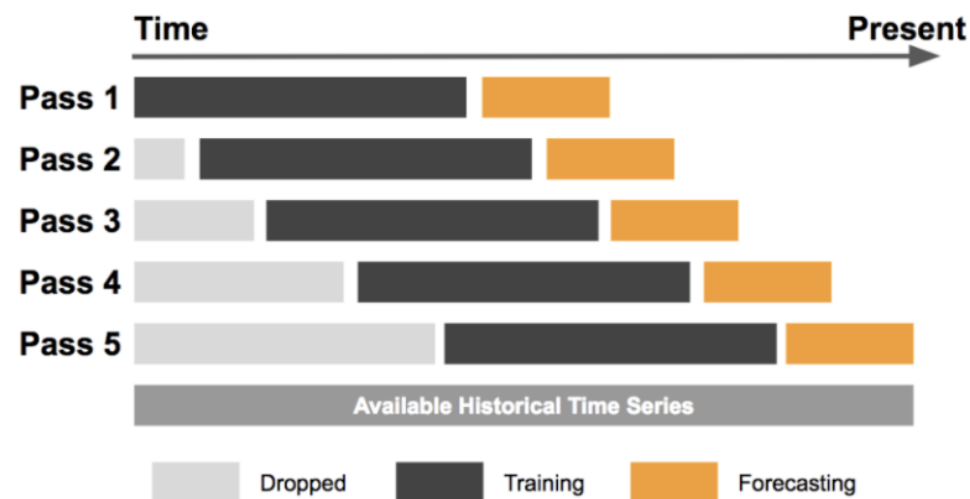


## Expanding Window



# Cross-validation for ~~the shit we do~~ time series

## Sliding Window



Training set is of  
fixed size

## Expanding Window



Training set size  
varies

# Solution: `rsample`

Classes and functions to  
create and summarize different  
types of resampling objects.

# Solution: `rsample`

Classes and functions to  
create and summarize different  
types of resampling objects.

**Bootstraps**

**V-Fold  
Cross-  
Validation**

**Sliding &  
Expanding  
Window**

# rolling\_origin()

```
rolling_origin(FB_adjusted,  
  initial = 500,  
  assess = 20,  
  cumulative = FALSE  
)
```

`initial` - The number of rows to start with.

`assess` - The number of rows to holdout for assessment.

`cumulative` - Sliding or Expanding?

```
> FB_adjusted  
# A tibble: 1,008 x 2  
  date      adjusted  
  <date>      <dbl>  
1 2013-01-02      28.0  
2 2013-01-03      27.8  
3 2013-01-04      28.8  
4 2013-01-07      29.4  
5 2013-01-08      29.1  
6 2013-01-09      30.6  
7 2013-01-10      31.3  
8 2013-01-11      31.7  
9 2013-01-14      31.0  
10 2013-01-15      30.1  
# ... with 998 more rows
```

# Extracting analysis / assessment sets

```
FB_splits ← rolling_origin(FB_adjusted, initial = 500,  
                           assess = 20, cumulative = FALSE)
```

```
# Rolling origin forecast resampling
```

```
# A tibble: 489 x 2
```

	splits	id
	<list>	<chr>

1	<S3: rsplit>	Slice001
---	--------------	----------

2	<S3: rsplit>	Slice002
---	--------------	----------

3	<S3: rsplit>	Slice003
---	--------------	----------

4	<S3: rsplit>	Slice004
---	--------------	----------

5	<S3: rsplit>	Slice005
---	--------------	----------

6	<S3: rsplit>	Slice006
---	--------------	----------

7	<S3: rsplit>	Slice007
---	--------------	----------

8	<S3: rsplit>	Slice008
---	--------------	----------

9	<S3: rsplit>	Slice009
---	--------------	----------

10	<S3: rsplit>	Slice010
----	--------------	----------

```
# ... with 479 more rows
```

# Extracting analysis / assessment sets

```
FB_splits ← rolling_origin(FB_adjusted, initial = 500,  
                           assess = 20, cumulative = FALSE)
```

```
# Rolling origin forecast resampling
```

```
# A tibble: 489 x 2
```

```
  splits      id  
  <list>    <chr>
```

```
1 <S3: rsplit> Slice001
```

```
2 <S3: rsplit> Slice002
```

```
3 <S3: rsplit> Slice003
```

```
4 <S3: rsplit> Slice004
```

```
5 <S3: rsplit> Slice005
```

```
6 <S3: rsplit> Slice006
```

```
7 <S3: rsplit> Slice007
```

```
8 <S3: rsplit> Slice008
```

```
9 <S3: rsplit> Slice009
```

```
10 <S3: rsplit> Slice010
```

```
# ... with 479 more rows
```

```
<train/test/total>
```

```
<500/20/1008>
```



# Extracting analysis / assessment sets

```
FB_splits ← rolling_origin(FB_adjusted, initial = 500,  
                             assess = 20, cumulative = FALSE)
```

```
# Rolling origin forecast resampling
```

```
# A tibble: 489 x 2
```

```
  splits      id  
  <list>    <chr>  
1 <S3: rsplit> Slice001  
2 <S3: rsplit> Slice002  
3 <S3: rsplit> Slice003  
4 <S3: rsplit> Slice004  
5 <S3: rsplit> Slice005  
6 <S3: rsplit> Slice006  
7 <S3: rsplit> Slice007  
8 <S3: rsplit> Slice008  
9 <S3: rsplit> Slice009  
10 <S3: rsplit> Slice010  
# ... with 479 more rows
```

<train/test/total>

<500/20/1008>

analysis()

```
# A tibble: 500 x 2
```

```
  date      adjusted  
  <date>    <dbl>  
1 2013-01-02      28  
2 2013-01-03     27.8  
3 2013-01-04     28.8
```

...

# Extracting analysis / assessment sets

```
FB_splits ← rolling_origin(FB_adjusted, initial = 500,  
                             assess = 20, cumulative = FALSE)
```

```
# Rolling origin forecast resampling
```

```
# A tibble: 489 x 2
```

	splits	id
	<list>	<chr>
1	<S3: rsplit>	Slice001
2	<S3: rsplit>	Slice002
3	<S3: rsplit>	Slice003
4	<S3: rsplit>	Slice004
5	<S3: rsplit>	Slice005
6	<S3: rsplit>	Slice006
7	<S3: rsplit>	Slice007
8	<S3: rsplit>	Slice008
9	<S3: rsplit>	Slice009
10	<S3: rsplit>	Slice010

```
# ... with 479 more rows
```

<train/test/total>

<500/20/1008>

analysis()

```
# A tibble: 500 x 2  
  date      adjusted  
  <date>    <dbl>  
1 2013-01-02      28  
2 2013-01-03     27.8  
3 2013-01-04     28.8  
...
```

assessment()

```
# A tibble: 20 x 2  
  date      adjusted  
  <date>    <dbl>  
1 2014-12-26     80.8  
2 2014-12-29     80.0  
3 2014-12-30     79.2  
...
```

# Workflow for fitting many models

```
library(purrr)
library(forecast)
```

```
fit_arima ← function(split) {
  # tibble with date and adjusted cols
  analysis_set ← analysis(split)

  # fit arima (really just AR1)
  Arima(
    y = analysis_set$adjusted,
    order = c(1, 0, 0)
  )
}
```

```
FB_splits %>%
  mutate(
    model = map(
      .x = splits,
      .f = ~fit_arima(.x)
    )
  )
```

```
# Rolling origin forecast resampling
# A tibble: 489 x 3
  splits      id      model
* <list>      <chr>    <list>
1 <S3: rsplit> Slice001 <S3: ARIMA>
2 <S3: rsplit> Slice002 <S3: ARIMA>
3 <S3: rsplit> Slice003 <S3: ARIMA>
4 <S3: rsplit> Slice004 <S3: ARIMA>
5 <S3: rsplit> Slice005 <S3: ARIMA>
6 <S3: rsplit> Slice006 <S3: ARIMA>
7 <S3: rsplit> Slice007 <S3: ARIMA>
8 <S3: rsplit> Slice008 <S3: ARIMA>
9 <S3: rsplit> Slice009 <S3: ARIMA>
10 <S3: rsplit> Slice010 <S3: ARIMA>
# ... with 479 more rows
```

# Then what?

- Predict on the assessment set
- Visualize performance
- Calculate in / out of sample performance metrics
- Calculate confidence intervals around metrics because of the resamples

## **Problem:**

You want to perform **cross-validation** in  
R with **time series**

**Problem:**

You want to perform **cross-validation** in  
R with **time series**

...

## **Problem:**

You want to perform **cross-validation** in  
R with **time series**

...

faster.

# Solution: `furrr`

Apply `purrr`'s mapping  
functions **in parallel**



# Example

```
library(purrr)
```

```
# Sleep 3 times - Sequentially
```

```
map(c(2, 2, 2), ~Sys.sleep(.x))
```

```
#> 6.08 sec elapsed
```

# Example

```
library(purrr)
```

```
# Sleep 3 times - Sequentially
```

```
map(c(2, 2, 2), ~Sys.sleep(.x))
```

```
#> 6.08 sec elapsed
```

```
library(furrr)  
plan(multiprocess)
```

```
# Sleep 3 times - In parallel!
```

```
future_map(c(2, 2, 2), ~Sys.sleep(.x))
```

```
#> 2.212 sec elapsed
```

# Example

```
library(purrr)
```

```
library(furrr)  
plan(multiprocess)
```

Set a parallel  
“plan”



```
# Sleep 3 times - Sequentially
```

```
map(c(2, 2, 2), ~Sys.sleep(.x))
```

```
#> 6.08 sec elapsed
```

```
# Sleep 3 times - In parallel!
```

```
future_map(c(2, 2, 2), ~Sys.sleep(.x))
```

```
#> 2.212 sec elapsed
```

rsample

+

furrr

=



# Fit resamples in parallel

```
FB_splits %>%  
  mutate(  
    model = map(  
      .x = splits,  
      .f = ~fit_arima(.x)  
    )  
  )
```

#> 8.113 sec elapsed

```
FB_splits %>%  
  mutate(  
    model = future_map(  
      .x = splits,  
      .f = ~fit_arima(.x)  
    )  
  )
```

#> 4.229 sec elapsed

**Demo**

# Thank you!

*Davis Vaughan*

 @dvaughan32

 DavisVaughan

davis@rstudio.com

<https://github.com/DavisVaughan/slides>