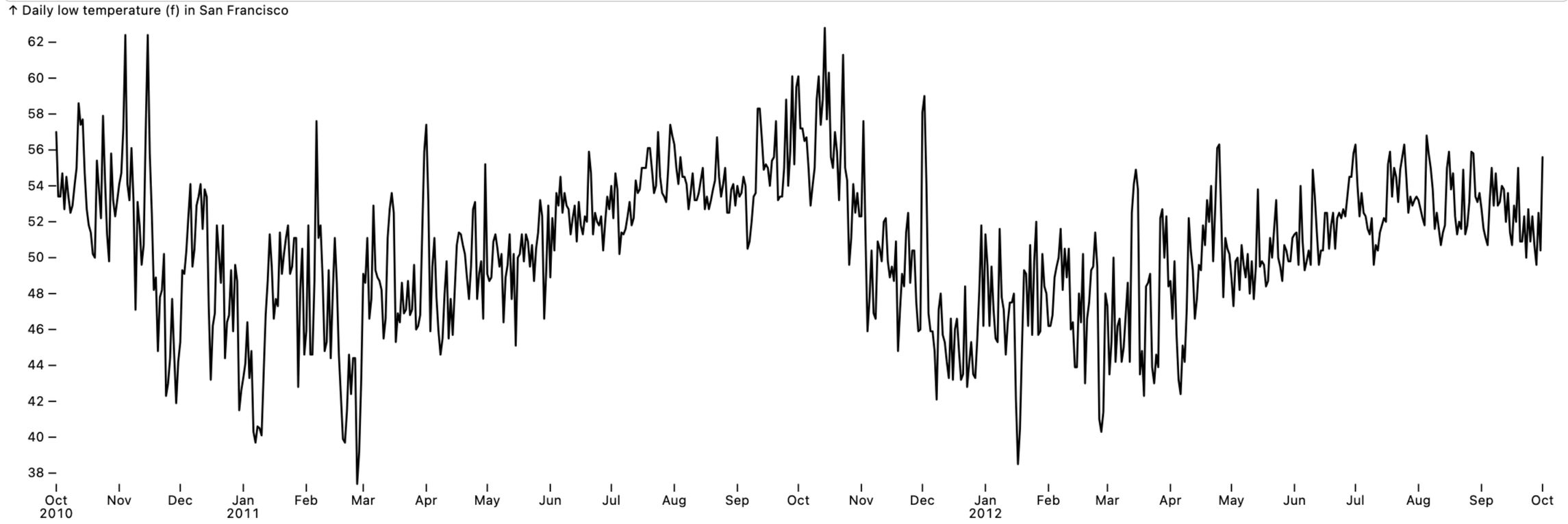
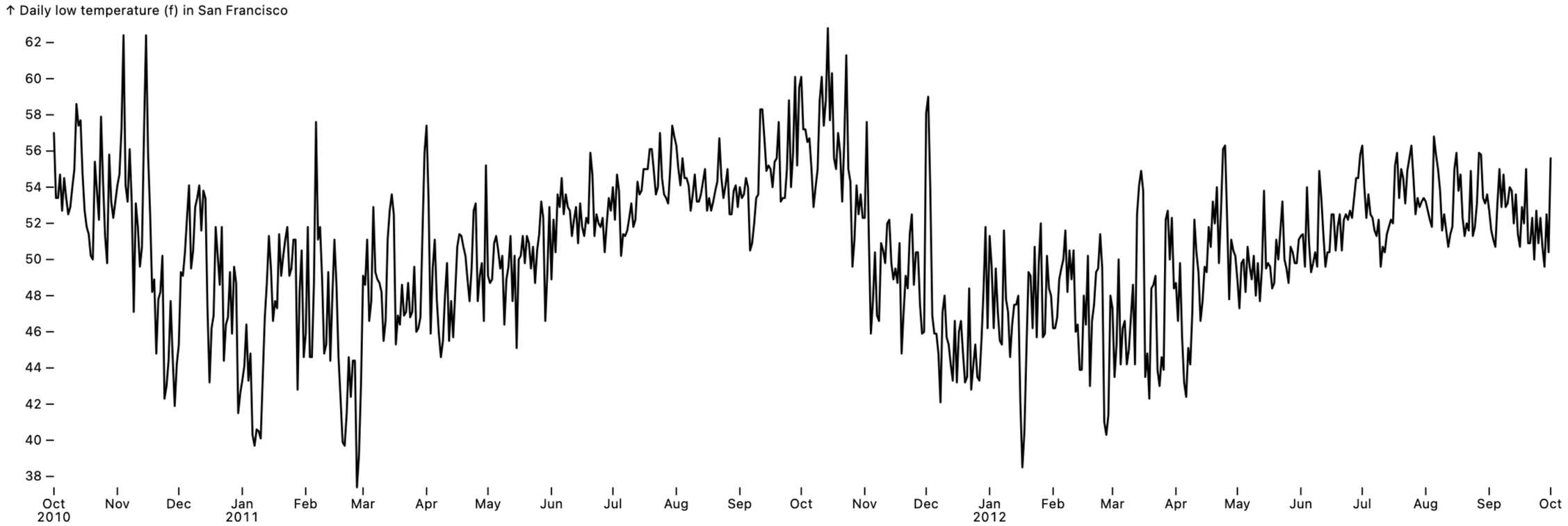
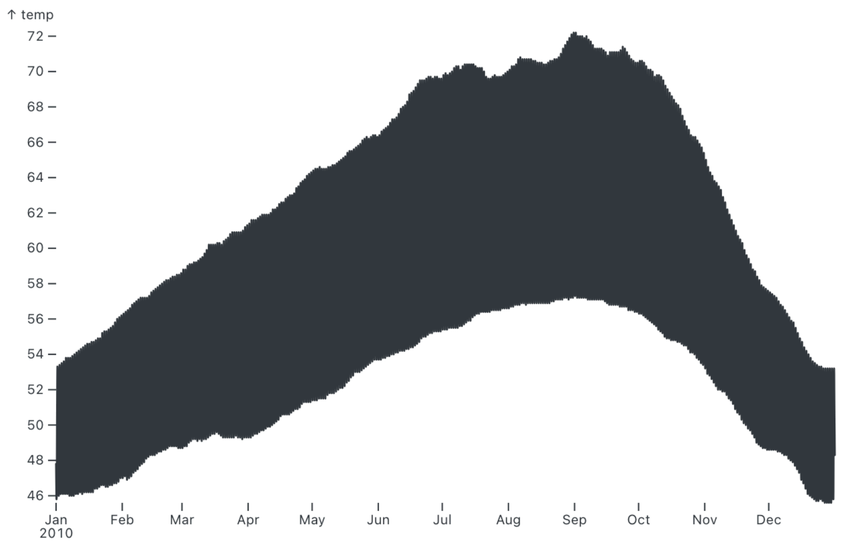
Timeseries visualization

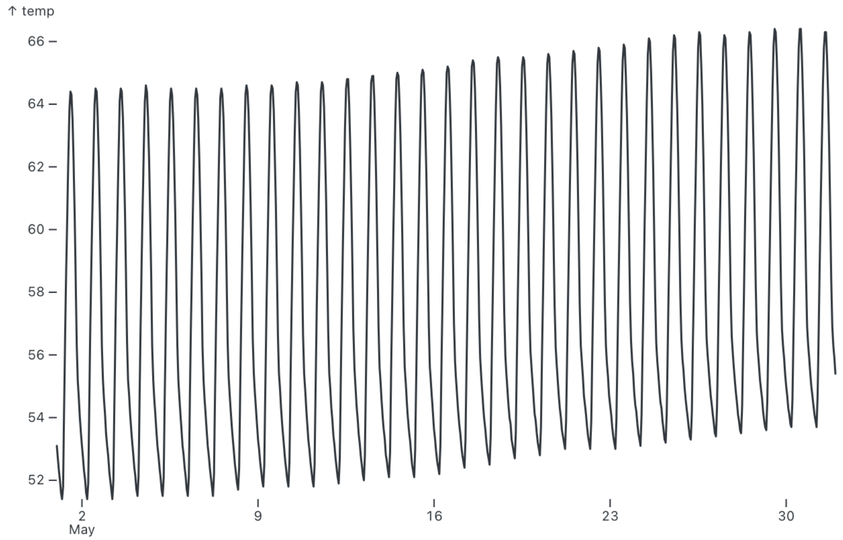
What properties do we want to highlight in a time series?



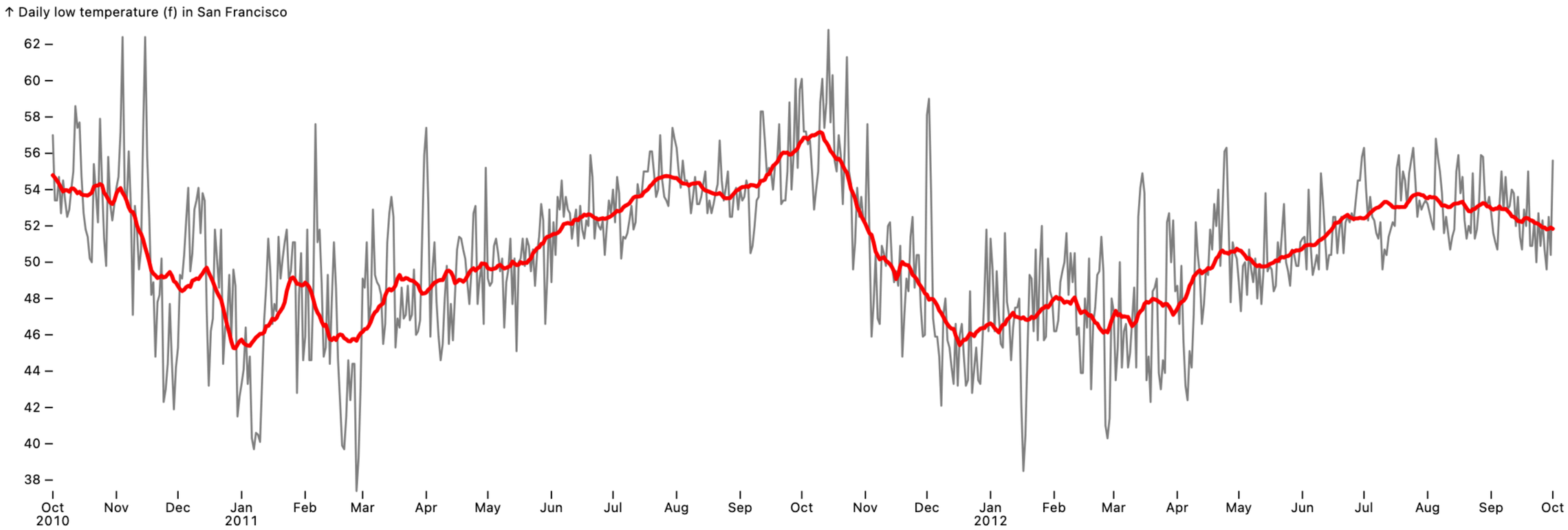
How do we highlight trends?




↑ temp
72
70
68
66
64
62
60
58
56
54
52
50
48
46
Jan
2010
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec

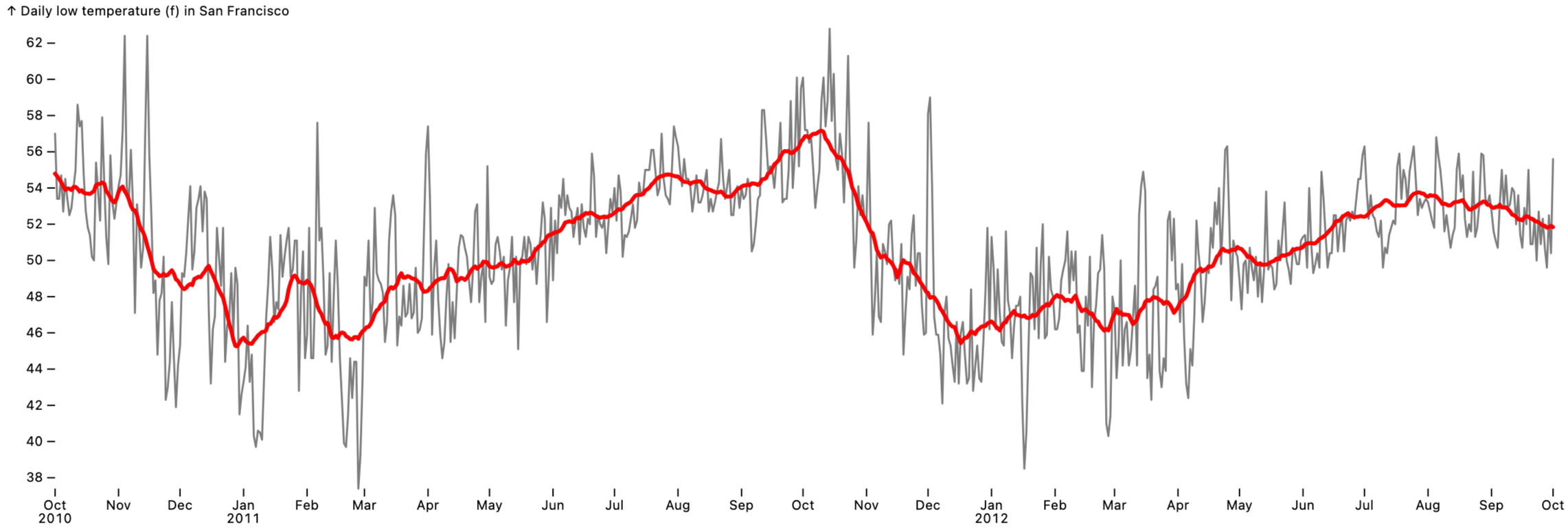

↑ temp
66
64
62
60
58
56
54
52
2
May
9
16
23
30

Moving Average



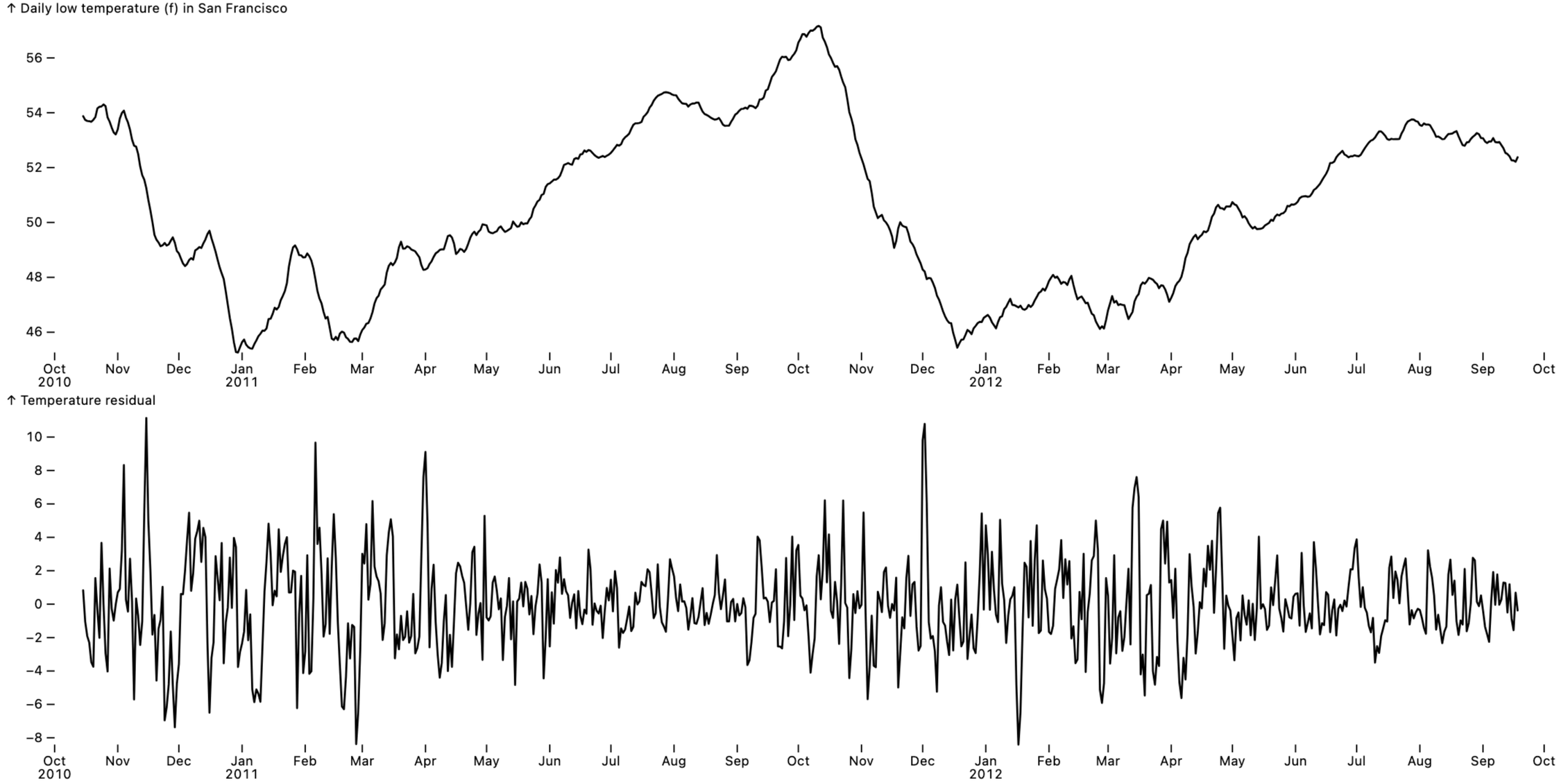
Moving Average (Pandas)

```
1  sf['MA'] = sf['low'].rolling(window=28, center=True).mean()
```



Timeseries Decomposition

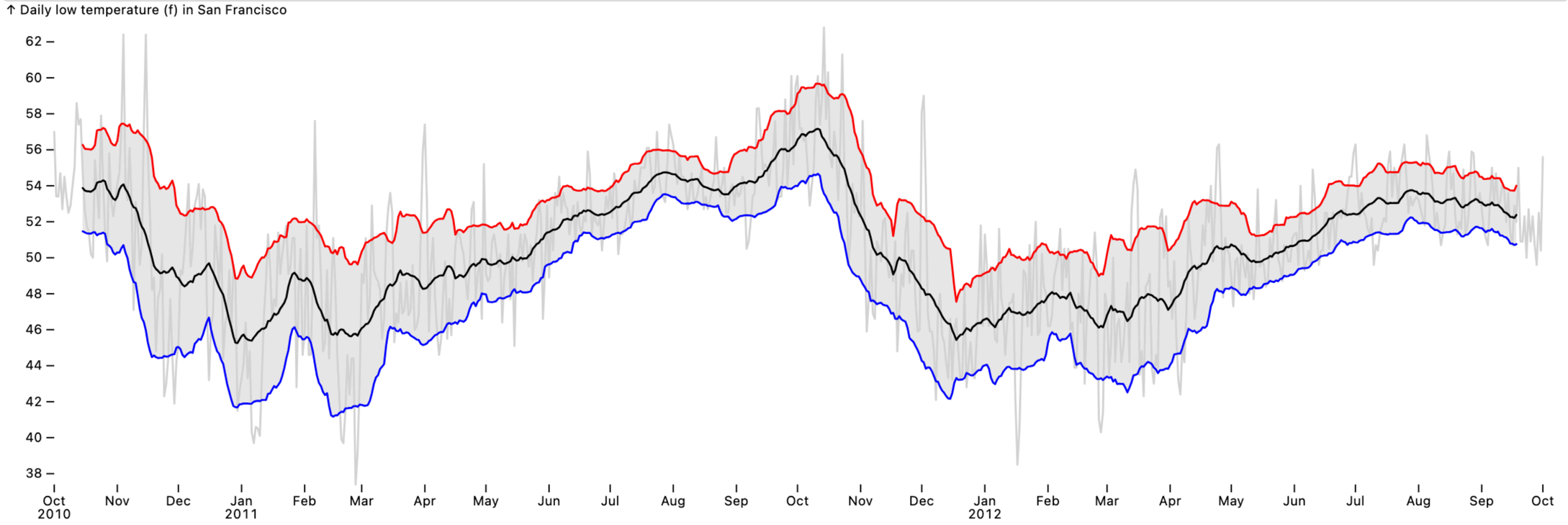
```
1  sf['residual'] = sf['low'] - sf['MA']
```



Bands

```
sf['std'] = sf['low'].rolling(window=28, center=True).std()
sf['lower_band'] = sf['MA'] - sf['std']
sf['upper_band'] = sf['MA'] + sf['std']
```

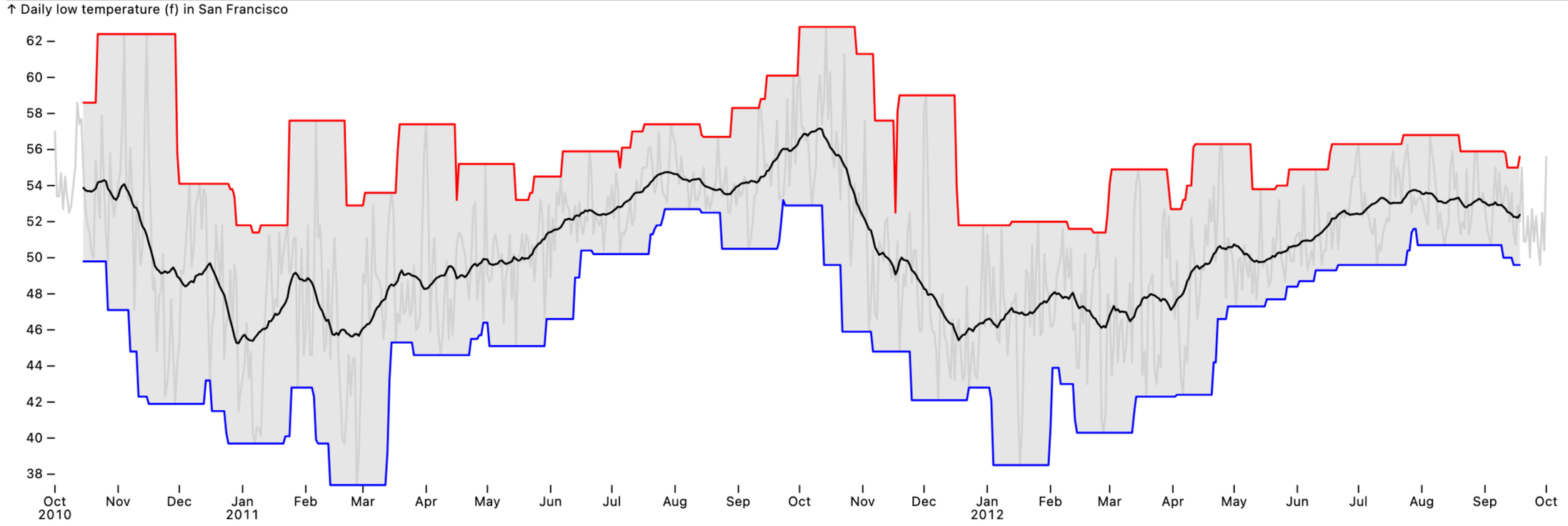
```
Plot.plot({
    x: {type: 'utc'}, y: {label: 'Daily low temperature (f) in San Francisco'}, width: 1200, height: 400,
    marks: [
        Plot.lineY(sf, {x: 'date', y: 'low', stroke: 'lightgrey'}),
        Plot.areaY(sf, {x: 'date', y1: 'lower_band', y2: 'upper_band', fill: 'lightgrey', fillOpacity: 0.5}),
        Plot.lineY(sf, {x: 'date', y: 'lower_band', stroke: 'blue'}),
        Plot.lineY(sf, {x: 'date', y: 'upper_band', stroke: 'red'}),
        Plot.lineY(sf, {x: 'date', y: 'MA', stroke: 'black'}),
    ]
})
```



Bands

```
sf['min'] = sf['low'].rolling(window=28, center=True).min()
sf['max'] = sf['low'].rolling(window=28, center=True).max()
sf['median'] = sf['low'].rolling(window=28, center=True).median()
```

```
Plot.plot({
    x: {type: 'utc'}, y: {label: 'Daily low temperature (f) in San Francisco'}, width: 1200, height: 400,
    marks: [
        Plot.lineY(sf, {x: 'date', y: 'low', stroke: 'lightgrey'}),
        Plot.areaY(sf, {x: 'date', y1: 'min', y2: 'max', fill: 'lightgrey', fillOpacity: 0.5}),
        Plot.lineY(sf, {x: 'date', y: 'min', stroke: 'blue'}),
        Plot.lineY(sf, {x: 'date', y: 'max', stroke: 'red'}),
        Plot.lineY(sf, {x: 'date', y: 'MA', stroke: 'black'}),
    ]
})
```

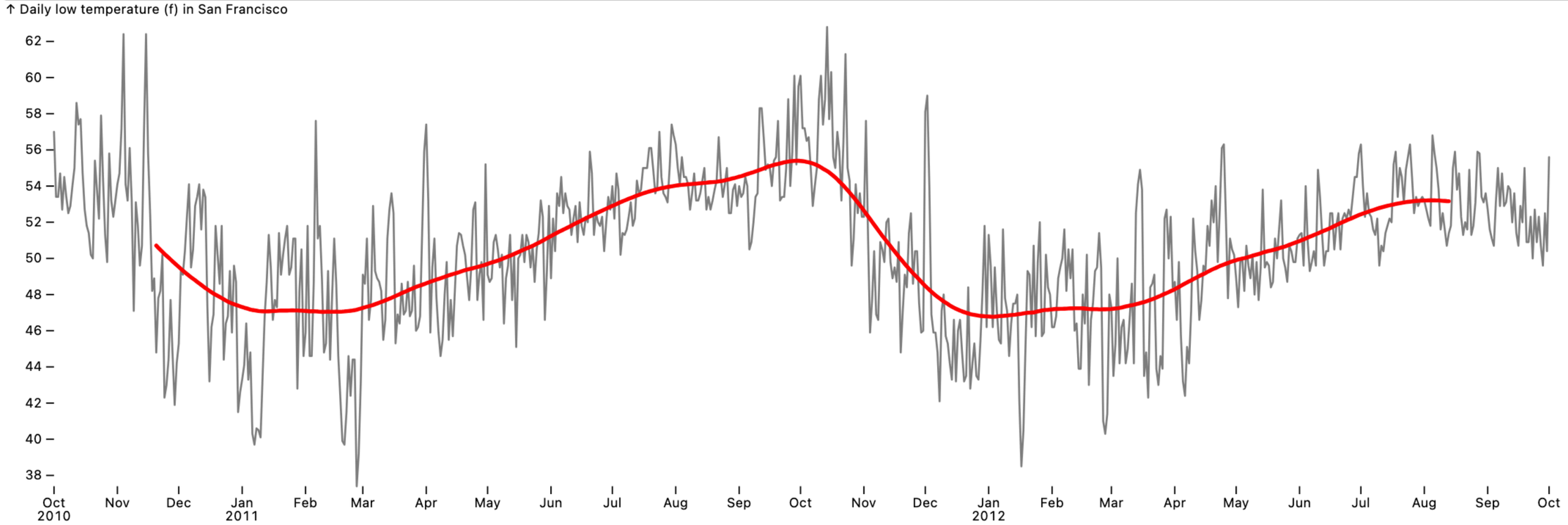


Candlestick chart

Weighted Moving average

```
1  sf['WMA'] = sf['low'].rolling(window=100, center=True, win_type='gaussian').mean(std=21)
```

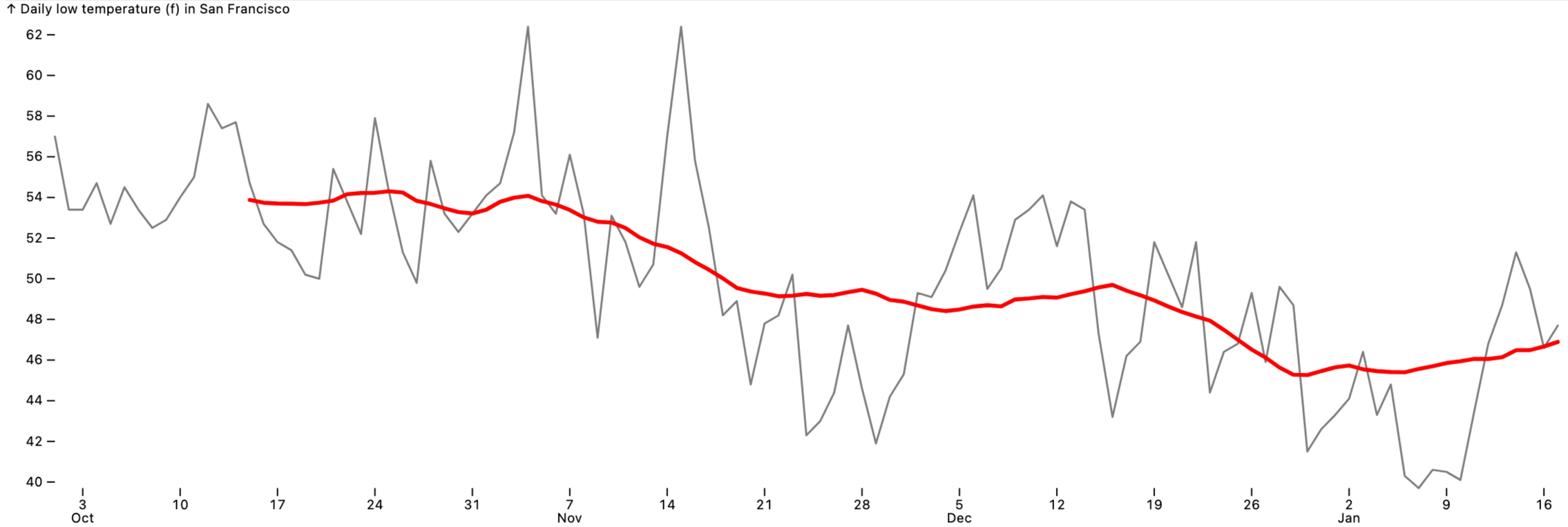
```
Plot.plot({
    x: {type: 'utc'}, y: {label: 'Daily low temperature (f) in San Francisco'}, width: 1200, height: 400,
    marks: [
        Plot.lineY(sf, {x: 'date', y: 'low', stroke: 'grey'}),
        Plot.lineY(sf, {x: 'date', y: 'WMA', stroke: 'red', strokeWidth: 3}),
    ]
})
```

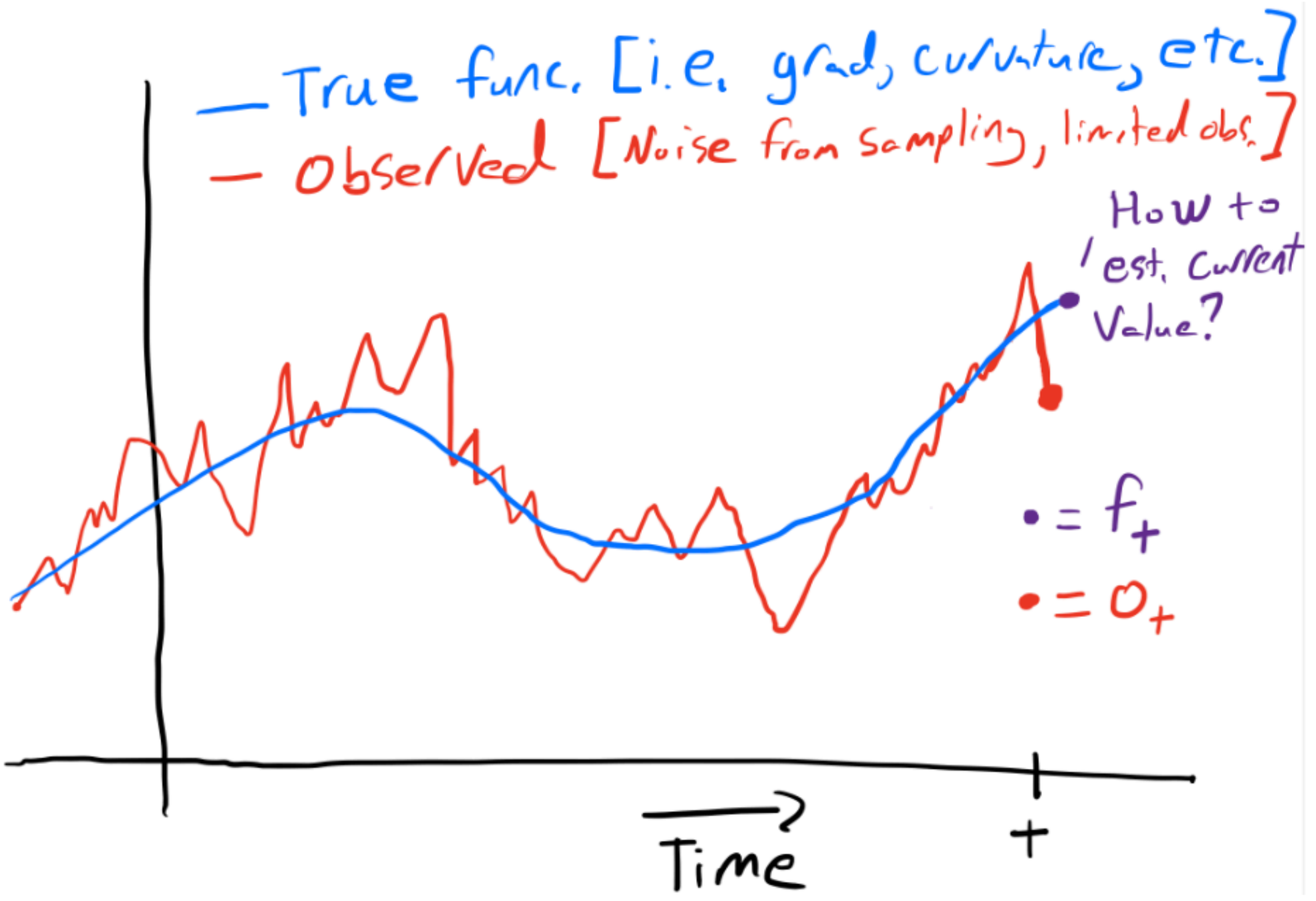


Boundary effects

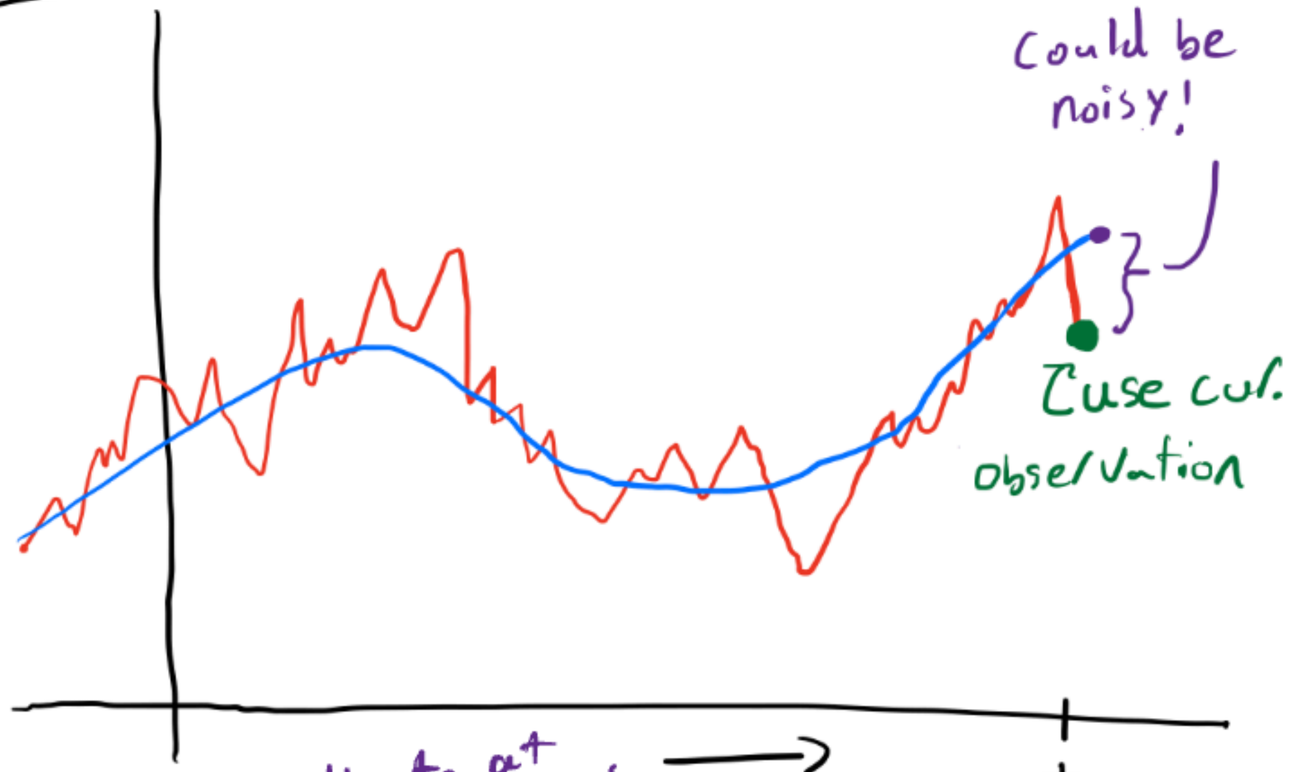
```
sf_small = sf.query('date < "2011-01-18"')
sf_small['MA'] = sf_small['low'].rolling(window=14, center=False).mean()
```

```
Plot.plot({
    x: {type: 'utc'}, y: {label: 'Daily low temperature (f) in San Francisco'}, width: 1200, height: 400,
    marks: [
        Plot.lineY(sf_small, {x: 'date', y: 'low', stroke: 'grey'}),
        Plot.lineY(sf_small,  {x: 'date', y: 'MA', stroke: 'red', strokeWidth: 3}),
    ]
})
```




— True func. [i.e. grad, curvature, etc.]
— Observed [Noise from sampling, limited obs.]
How to
est. current
value?
• = f_t
• = O_t
Time
t

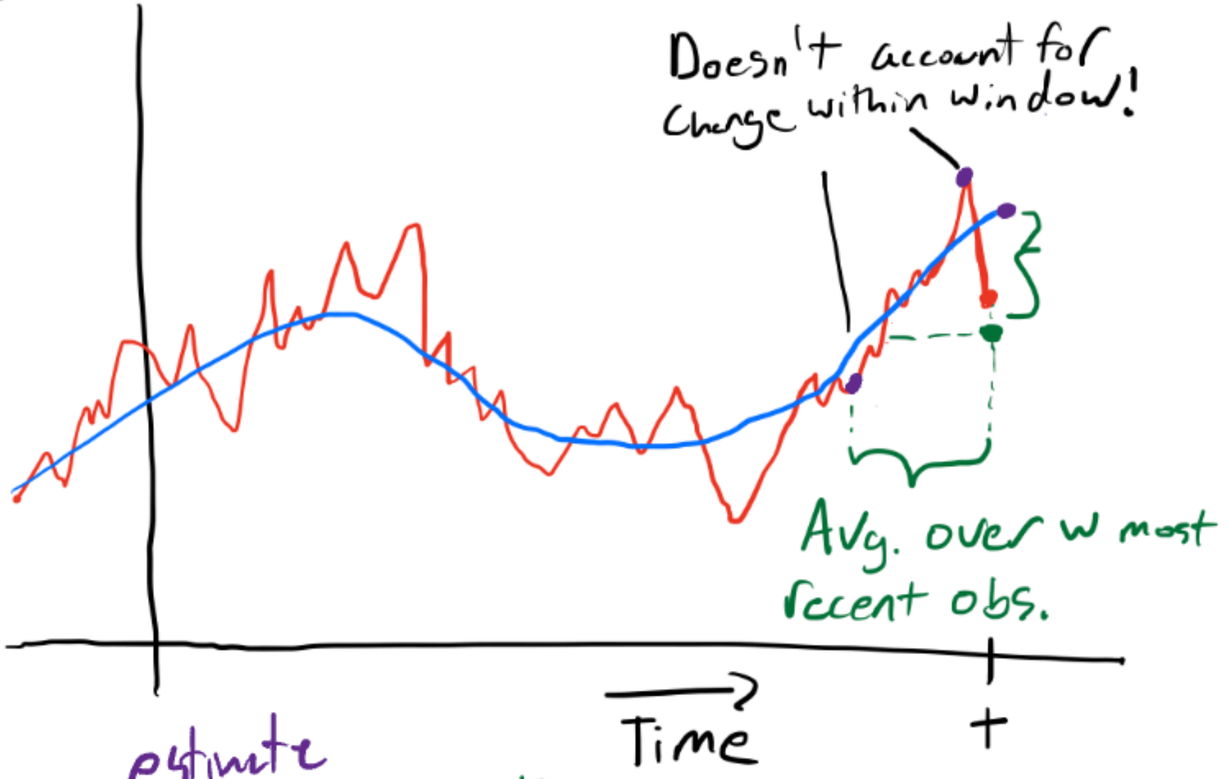
Idea 1



$$f_t \approx O_t$$

O_t: observation at time t

Idea 2 [Box avg.]

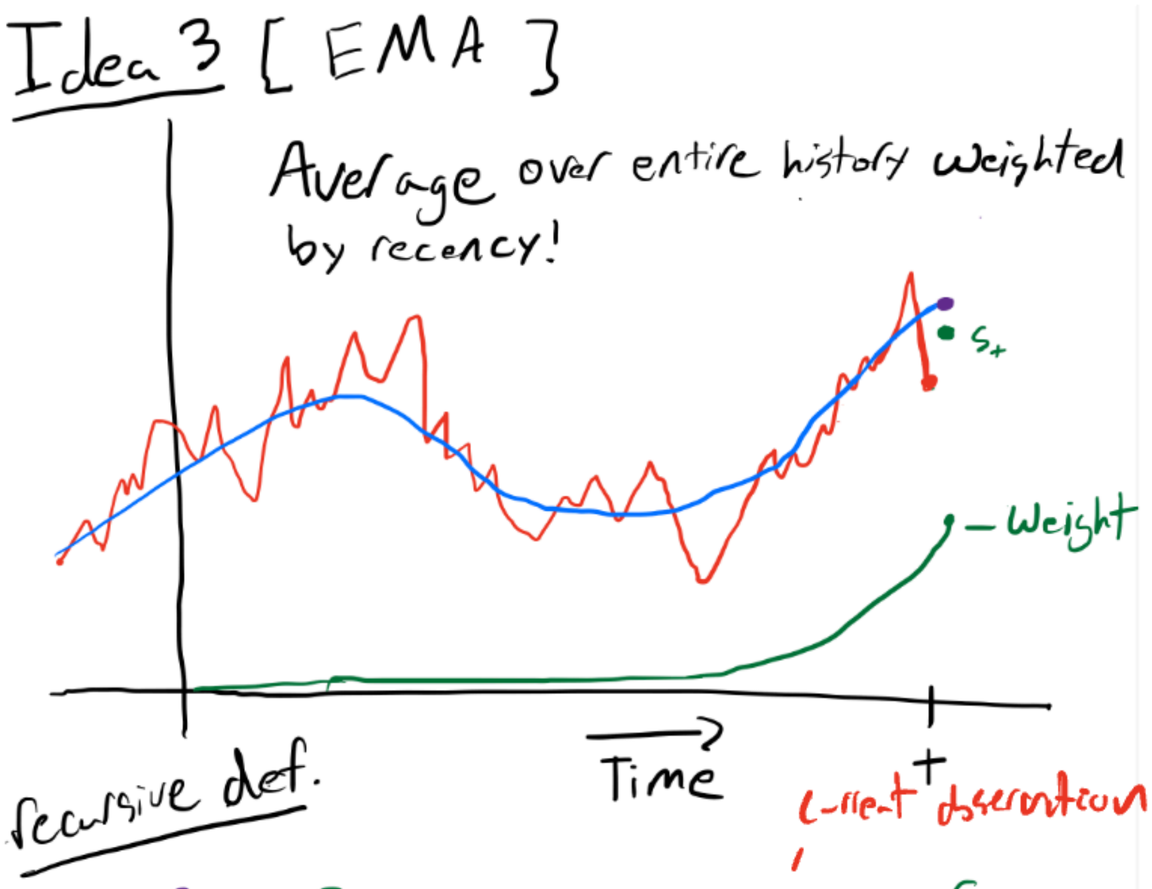


estimate

$$f_t \approx \frac{1}{w} \sum_{i=t-w}^{t} o_t$$

↳ Average obs. over a window of size w

Exponential moving average (EMA)



$f_t \approx S_t$

f_t: estimate at time t

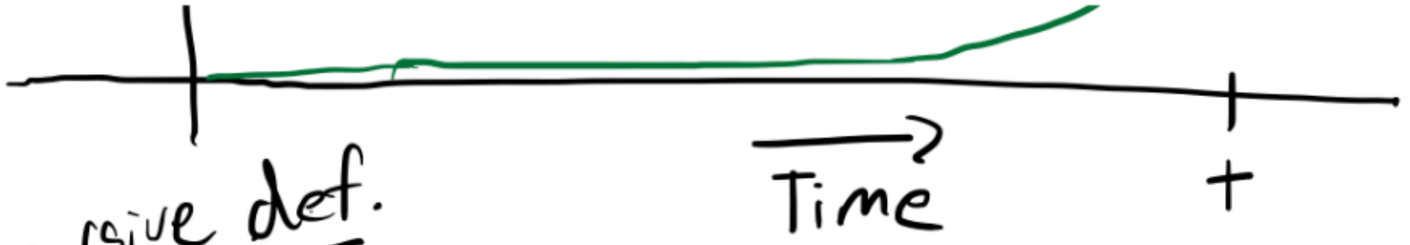
$S_t = (1-\alpha) O_t + \alpha S_{t-1}$

O_t: current observation

S_{t-1}: prev. at $t-1$

$\alpha \in [0, 1]$

Exponential moving average (EMA)



momentum

Recursive def.

$$f_t \approx S_t \qquad S_t = (1-\alpha) O_t + \alpha S_{t-1}$$

$$\rightarrow f_t \approx (1-\alpha) O_t + (1-\alpha)\alpha O_{t-1} + (1-\alpha)\alpha^2 O_{t-2} + \dots$$

Ex. $\alpha = 0.9$

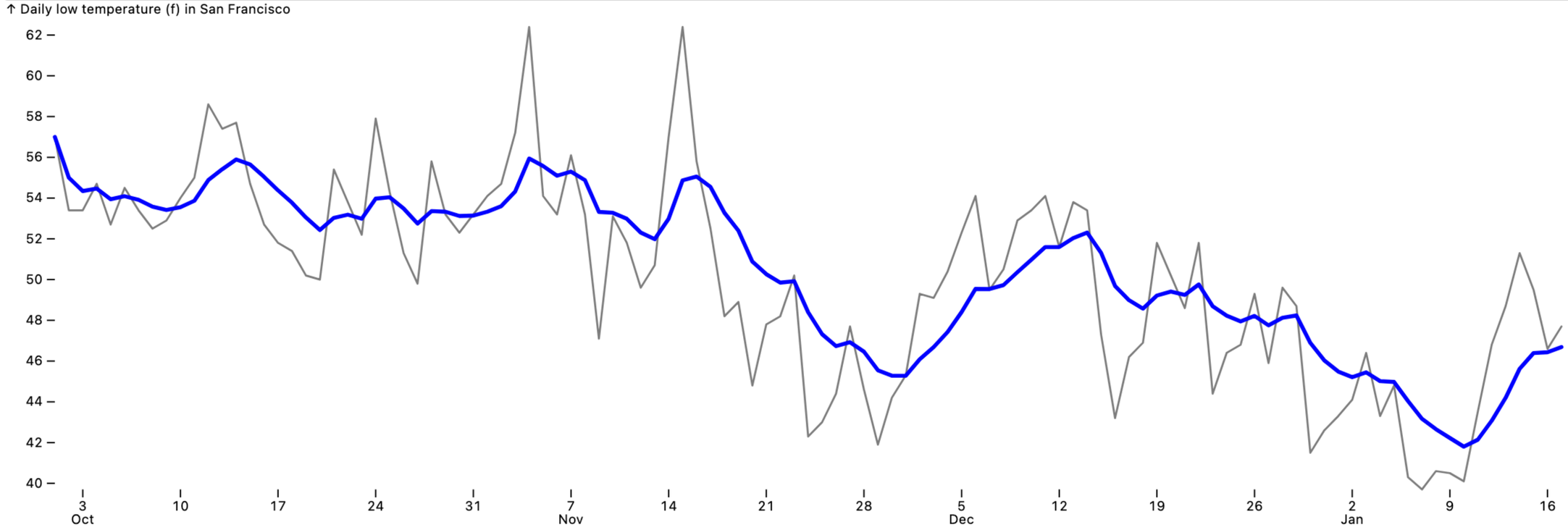
$$f_t \approx 0.1 O_t + 0.09 O_{t-1} + 0.081 O_{t-2} + 0.0729 O_{t-3} + \dots$$

$$\approx 0 + \square O_{t-100}$$

Exponential Moving average

```
sf_small = sf.query('date < "2011-01-18"')
sf_small['EMA'] = sf_small['low'].ewm(alpha=0.2).mean()
```

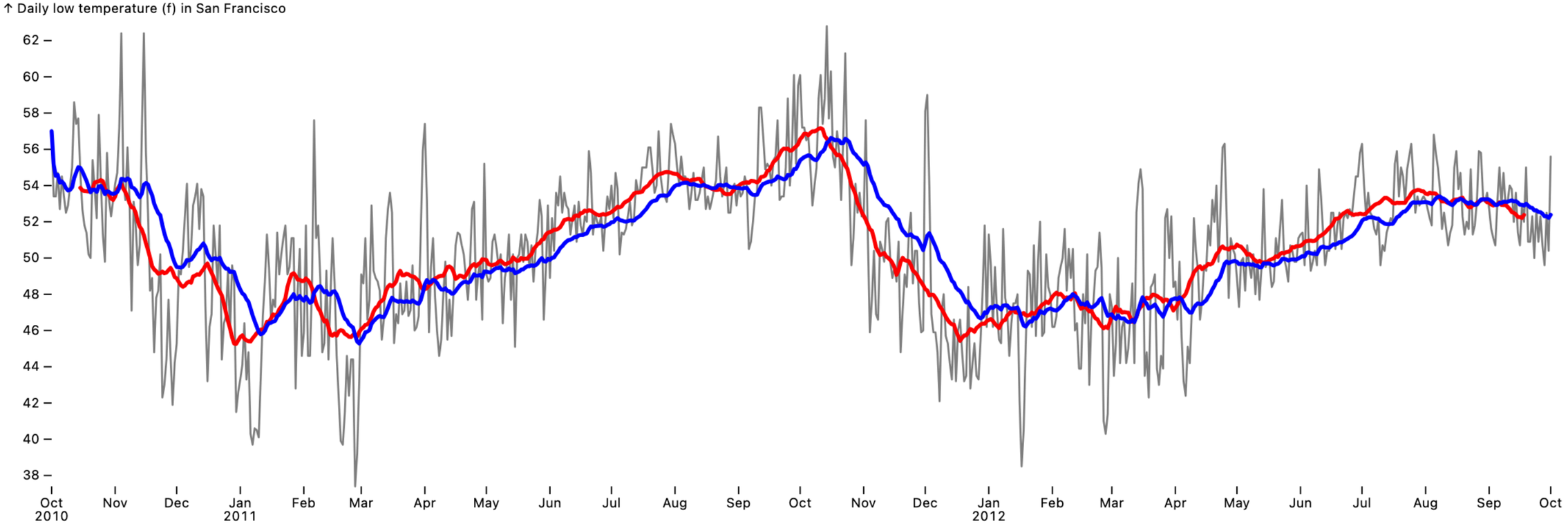
```
Plot.plot({
    x: {type: 'utc'}, y: {label: 'Daily low temperature (f) in San Francisco'}, width: 1200, height: 400,
    marks: [
        Plot.lineY(sf_small, {x: 'date', y: 'low', stroke: 'grey'}),
        Plot.lineY(sf_small,  {x: 'date', y: 'EMA', stroke: 'blue', strokeWidth: 3}),
    ]
})
```



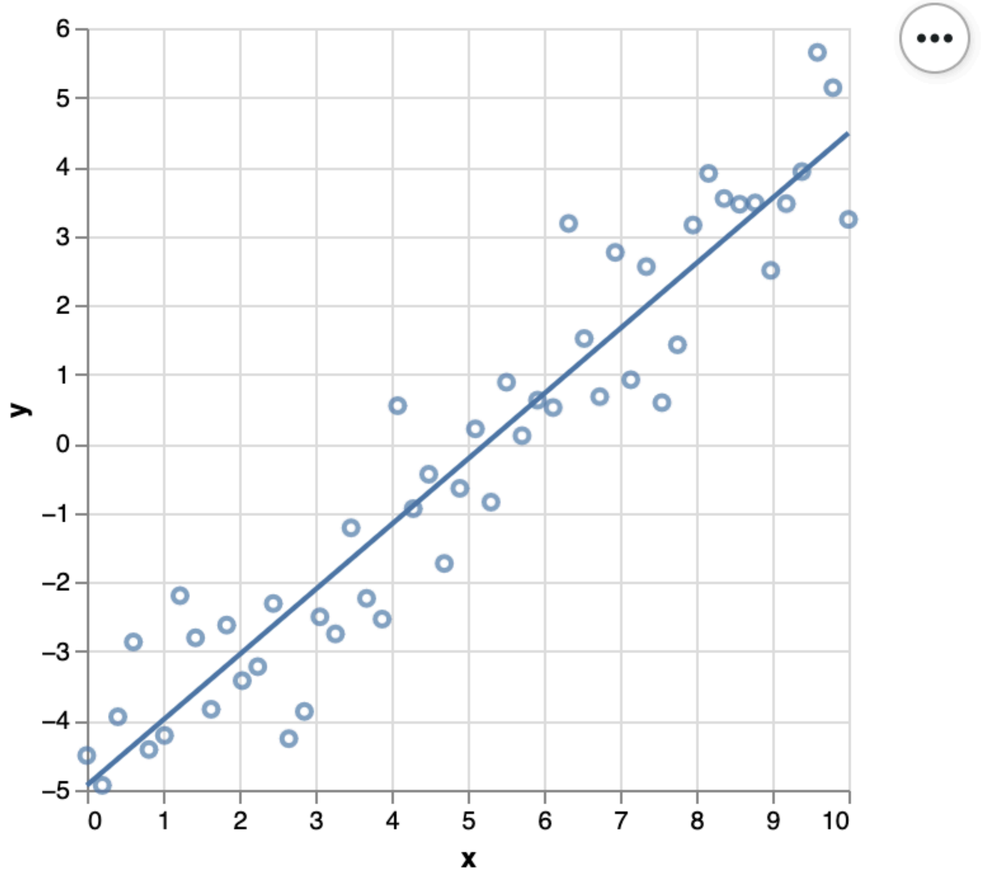
Exponential Moving average

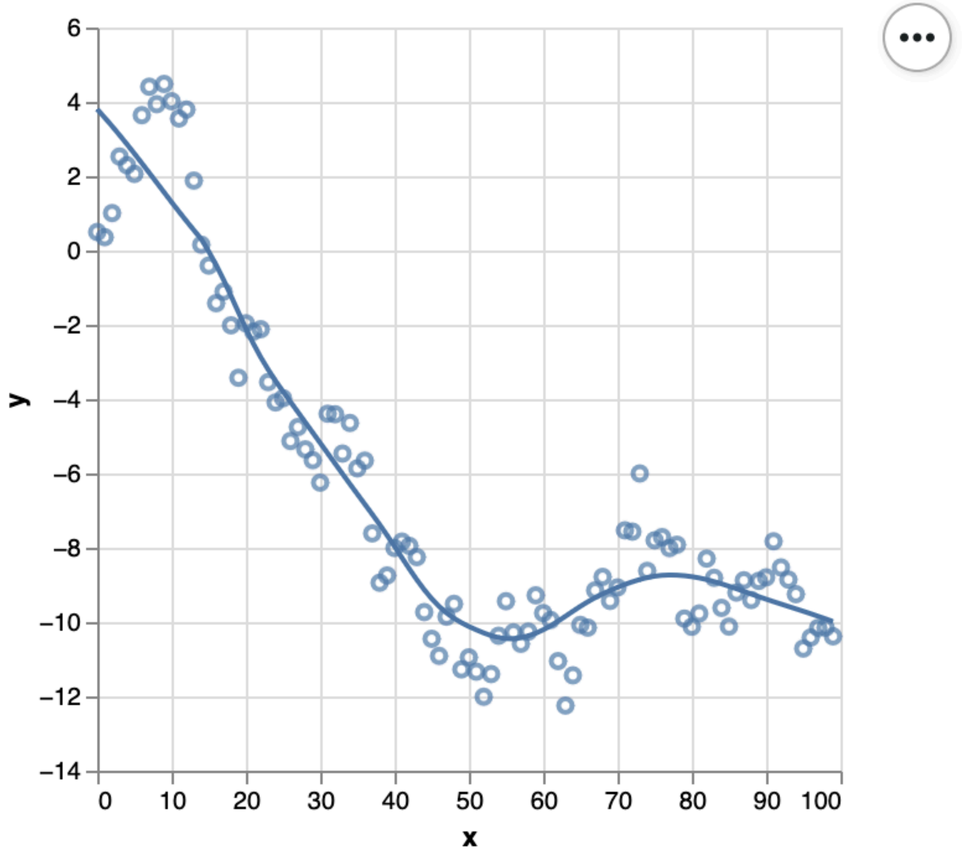
```
1  sf['EMA'] = sf['low'].ewm(alpha=0.05).mean()
```

```
Plot.plot({
    x: {type: 'utc'}, y: {label: 'Daily low temperature (f) in San Francisco'}, width: 1200, height: 400,
    marks: [
        Plot.lineY(sf, {x: 'date', y: 'low', stroke: 'grey'}),
        Plot.lineY(sf,  {x: 'date', y: 'MA', stroke: 'red', strokeWidth: 3}),
        Plot.lineY(sf,  {x: 'date', y: 'EMA', stroke: 'blue', strokeWidth: 3}),
    ]
})
```



Linear and LO(W)ESS Regression

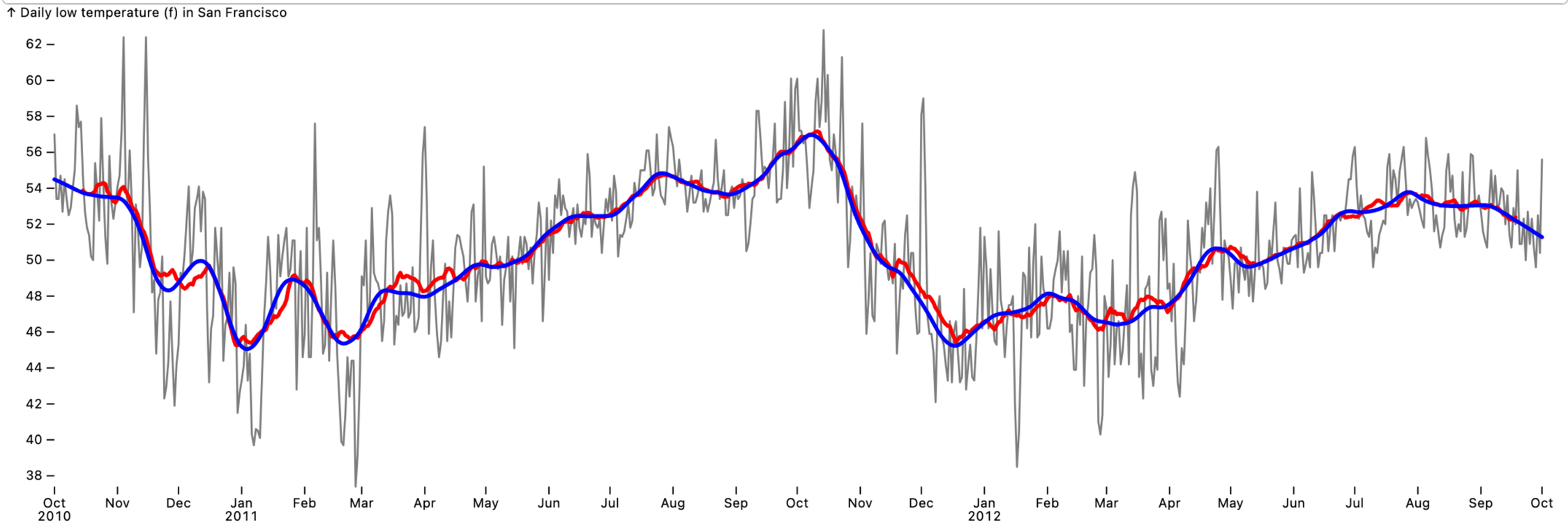




LOWESS Smoothing

```
from statsmodels.nonparametric.smoothers_lowess import lowess
sf['lowess'] = lowess(sf['low'], np.arange(len(sf['low'])), return_sorted=False, it=1, frac=0.05)
```

```
Plot.plot({
    x: {type: 'utc'}, y: {label: 'Daily low temperature (f) in San Francisco'}, width: 1200, height: 400,
    marks: [
        Plot.lineY(sf, {x: 'date', y: 'low', stroke: 'grey'}),
        Plot.lineY(sf,  {x: 'date', y: 'MA', stroke: 'red', strokeWidth: 3}),
        Plot.lineY(sf,  {x: 'date', y: 'lowess', stroke: 'blue', strokeWidth: 3}),
    ]
})
```



How are dates represented?

As text:

```
"2024-10-28 11:15:23.45 UTC-07:00"
```

How are dates represented?

In Python

```
1 import time
2 time.time()
```

```
1730918860.749311
```

How are dates represented?

In Pandas

```
1  pd.Timestamp(time.time(), unit='s')
```

```
Timestamp('2024-11-06 18:47:40.754076004')
```

Verify epoch

```
1  pd.Timestamp(0, unit='s')
```

```
Timestamp('1970-01-01 00:00:00')
```

How are dates represented?

Time *deltas*

```
1  pd.Timestamp(time.time(), unit='s') - pd.Timestamp(0, unit='s')
```

```
Timedelta('20033 days 18:47:40.763437033')
```

Delta math

```
1  delta = pd.Timestamp(time.time(), unit='s') - pd.Timestamp(0, unit='s')
2  pd.Timestamp(time.time(), unit='s') + delta
```

```
Timestamp('2079-09-13 13:35:21.536878825')
```

```
1  pd.Timestamp(time.time(), unit='s') - delta
```

```
Timestamp('1970-01-01 00:00:00.005089998')
```

```
1  delta * 5
```

```
Timedelta('100168 days 21:58:23.842070105')
```

Timestamp math ***Not allowed***

```
1  pd.Timestamp(time.time(), unit='s') + pd.Timestamp(time.time(), unit='s')
```

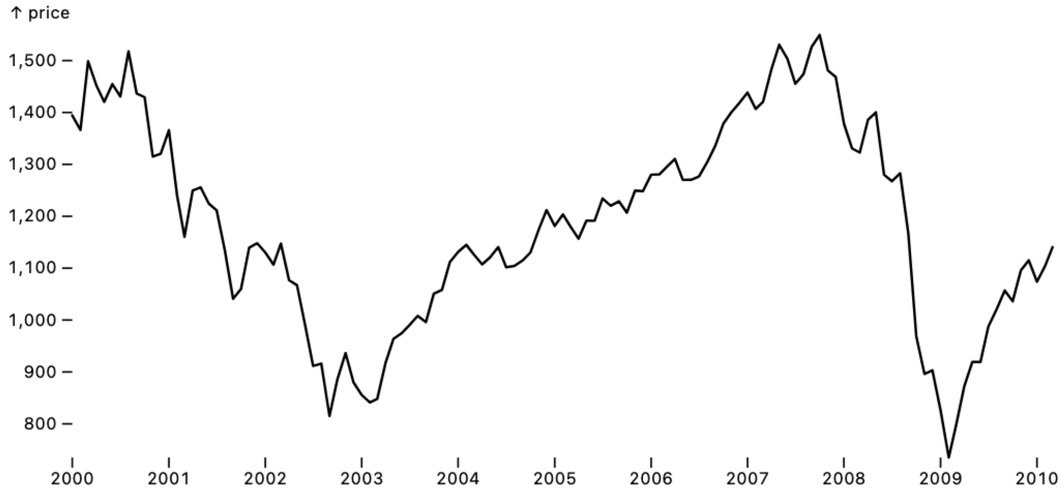
Resampling

Consider 2 data sets:

S&P 500 Index

- Collected *monthly*

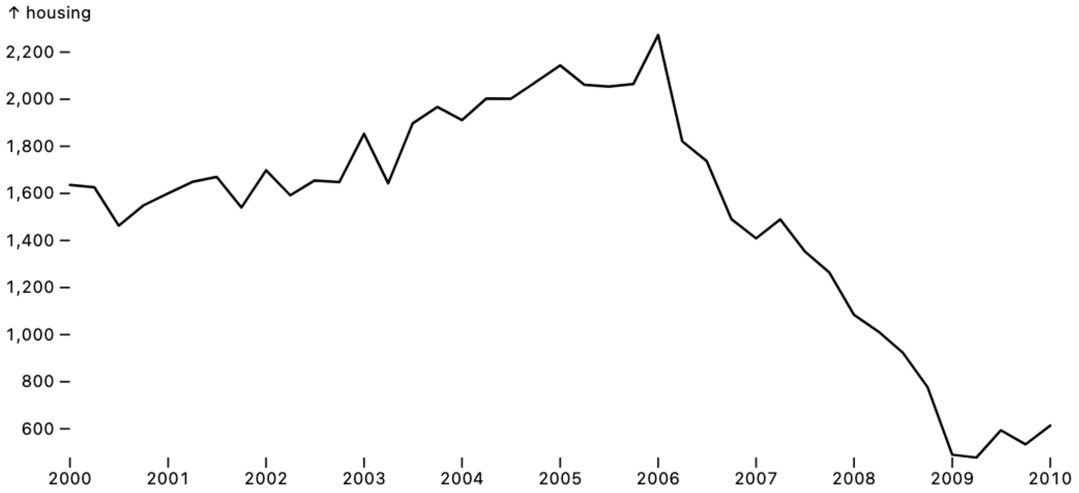
	date	price
0	2000-01-01	1394.46
1	2000-02-01	1366.42
2	2000-03-01	1498.58
3	2000-04-01	1452.43
4	2000-05-01	1420.60



U.S. New Housing construction

- Collected *quarterly*

	date	housing
0	2000-01-01	1636.0
1	2000-04-01	1626.0
2	2000-07-01	1463.0
3	2000-10-01	1549.0
4	2001-01-01	1600.0



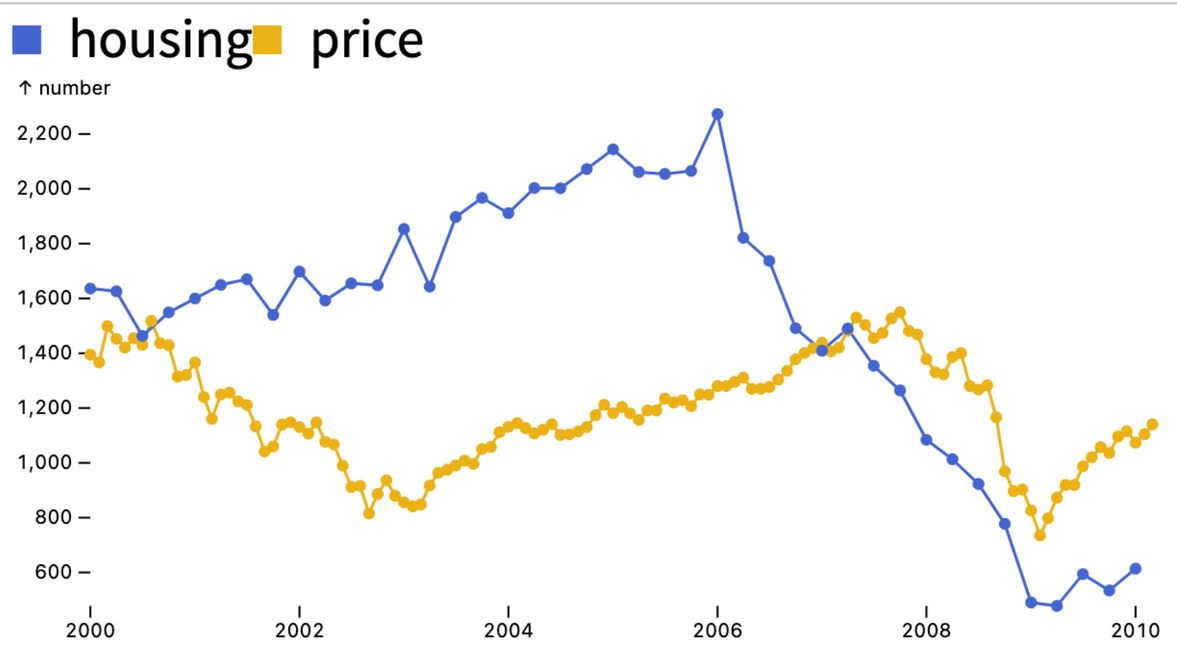
Resampling

Joining The datasets

- Housing variable is *missing* for many months.

```
1 data = spx.merge(housing, on='date', how='outer')
2 data.head(15)
```

	date	price	housing
0	2000-01-01	1394.46	1636.0
1	2000-02-01	1366.42	NaN
2	2000-03-01	1498.58	NaN
3	2000-04-01	1452.43	1626.0
4	2000-05-01	1420.60	NaN
5	2000-06-01	1454.60	NaN
6	2000-07-01	1430.83	1463.0
7	2000-08-01	1517.68	NaN
8	2000-09-01	1436.51	NaN
9	2000-10-01	1429.40	1549.0
10	2000-11-01	1314.95	NaN
11	2000-12-01	1320.28	NaN
12	2001-01-01	1366.01	1600.0
13	2001-02-01	1239.94	NaN
14	2001-03-01	1160.33	NaN



Resampling

2 options:

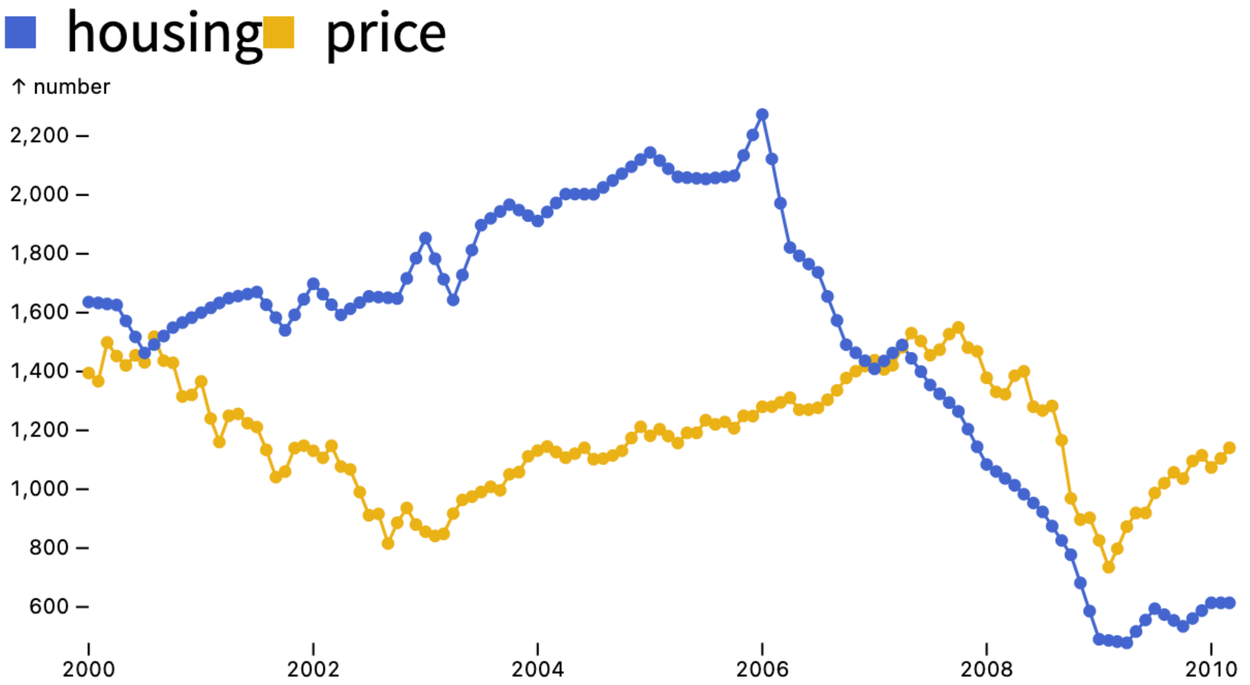
- *Upsample* `housing` to match the frequency of S&P 500 `price` (use *interpolation*)
- *Downsample* `price` to match the frequency of `housing` (how?)

Resampling

Upsample `housing` to match the frequency of S&P 500 `price` (use *interpolation*)

```
1 data = spx.merge(housing, on='date', how='outer')
2 data = data.interpolate()
```

	date	price	housing
0	2000-01-01	1394.46	1636.000000
1	2000-02-01	1366.42	1632.666667
2	2000-03-01	1498.58	1629.333333
3	2000-04-01	1452.43	1626.000000
4	2000-05-01	1420.60	1571.666667
5	2000-06-01	1454.60	1517.333333
6	2000-07-01	1430.83	1463.000000
7	2000-08-01	1517.68	1491.666667
8	2000-09-01	1436.51	1520.333333
9	2000-10-01	1429.40	1549.000000
10	2000-11-01	1314.95	1566.000000
11	2000-12-01	1320.28	1583.000000
12	2001-01-01	1366.01	1600.000000
13	2001-02-01	1239.94	1616.333333
14	2001-03-01	1160.33	1632.666667



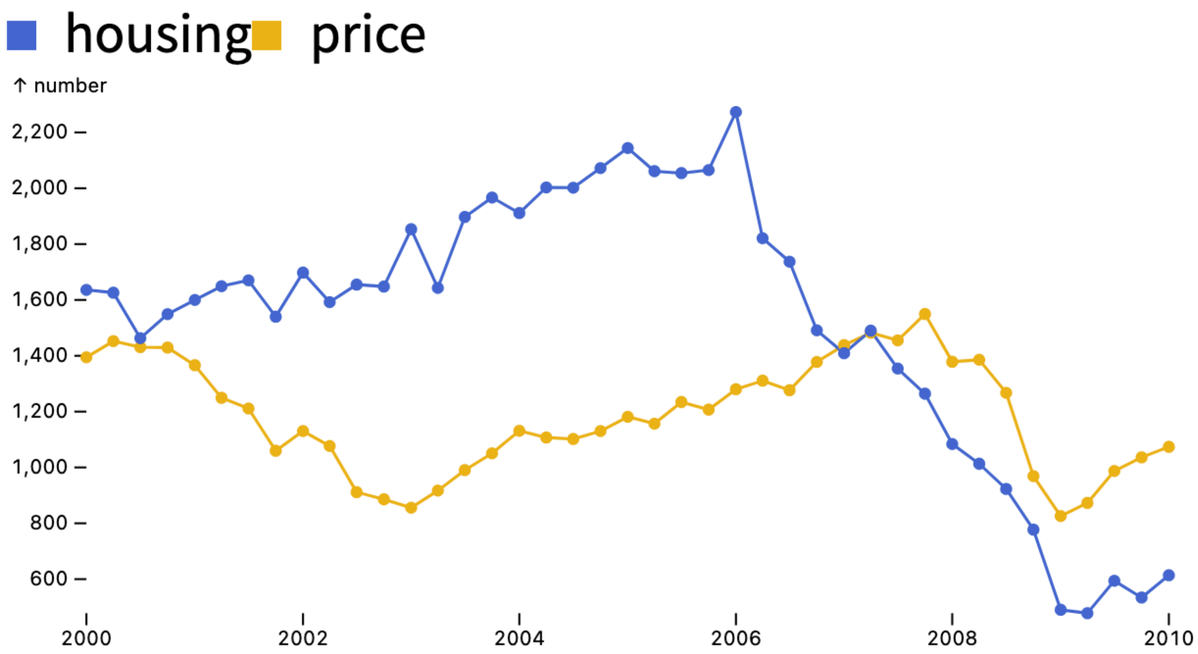
Resampling

Downsample `price` to match the frequency of `housing`

- Could just drop the non-matching housing observations

```
1 data = spx.merge(housing, on='date', how='outer')
2 data = data.dropna()
```

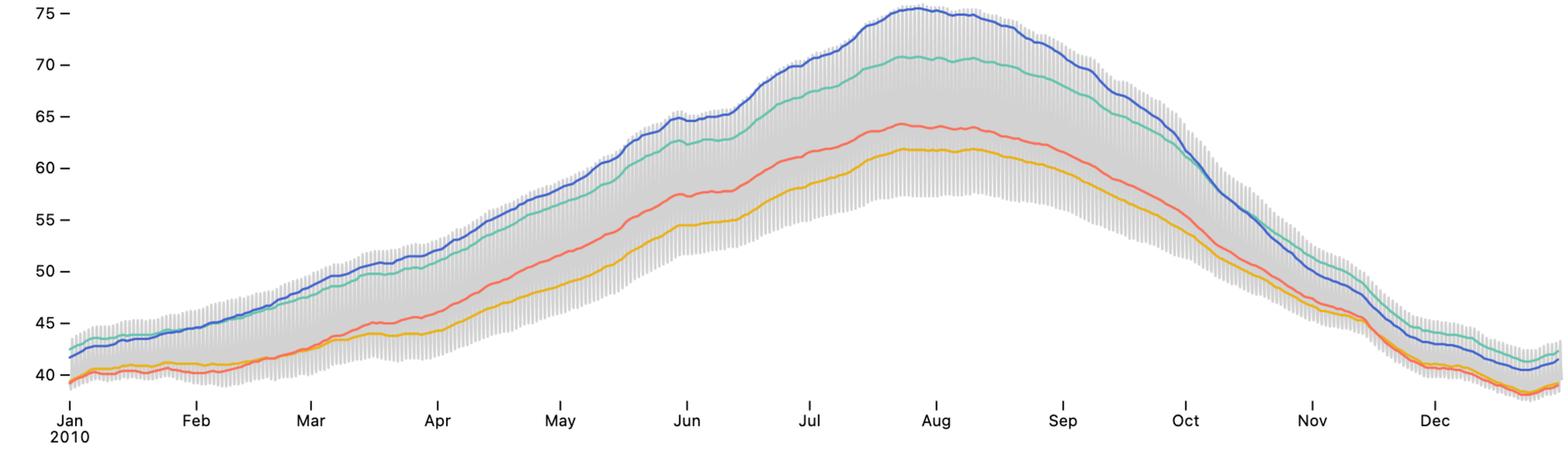
	date	price	housing
0	2000-01-01	1394.46	1636.0
3	2000-04-01	1452.43	1626.0
6	2000-07-01	1430.83	1463.0
9	2000-10-01	1429.40	1549.0
12	2001-01-01	1366.01	1600.0
15	2001-04-01	1249.46	1649.0
18	2001-07-01	1211.23	1670.0
21	2001-10-01	1059.78	1540.0
24	2002-01-01	1130.20	1698.0
27	2002-04-01	1076.92	1592.0
30	2002-07-01	911.62	1655.0
33	2002-10-01	885.76	1648.0
36	2003-01-01	855.70	1853.0
39	2003-04-01	916.92	1643.0
42	2003-07-01	990.31	1897.0



Resampling

Extreme variations in other datasets

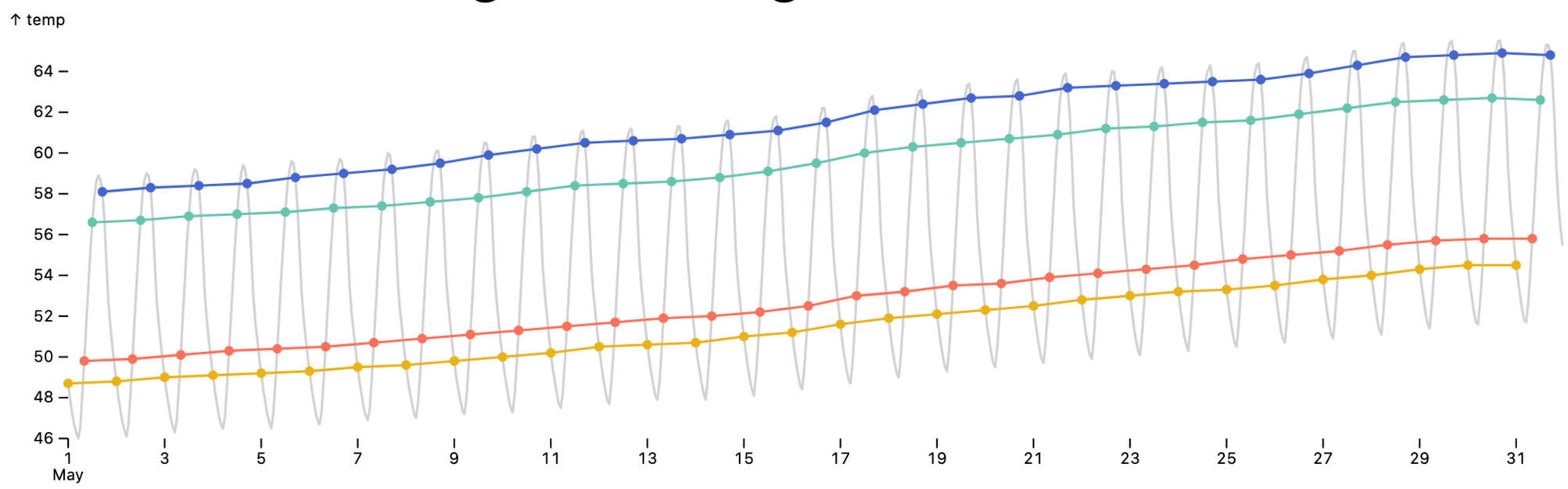
- Want to downsample Seattle temperature from hourly to daily
- Do we take values from 12am, 9am, 5pm…?



Resampling

Extreme variations in other datasets

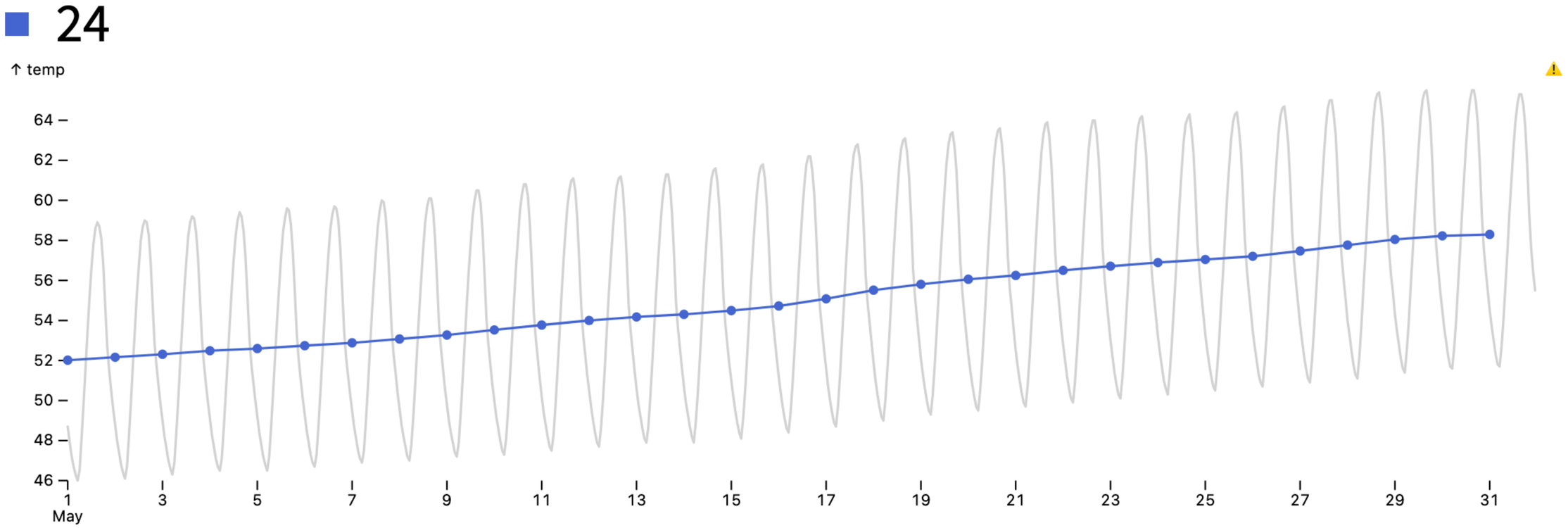
- Want to downsample Seattle temperature from hourly to daily
- Do we take values from 12am, 9am, 5pm…?



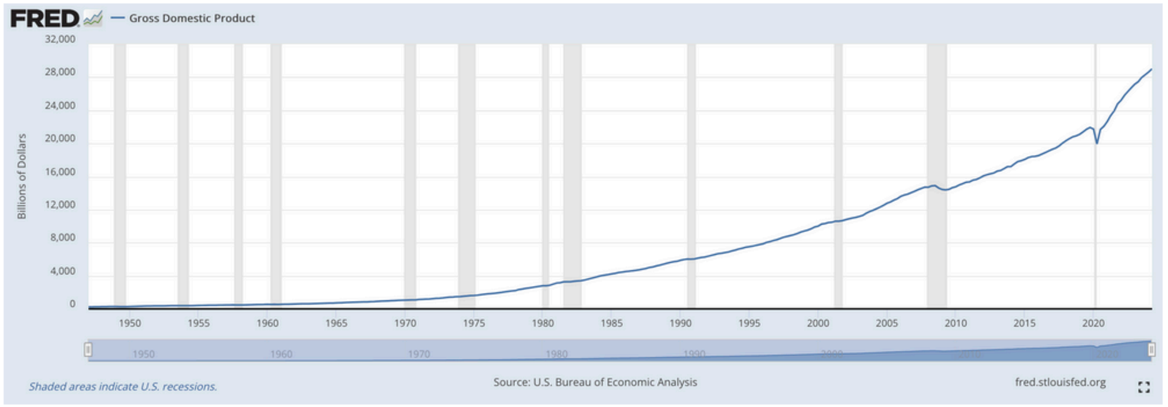
Resampling

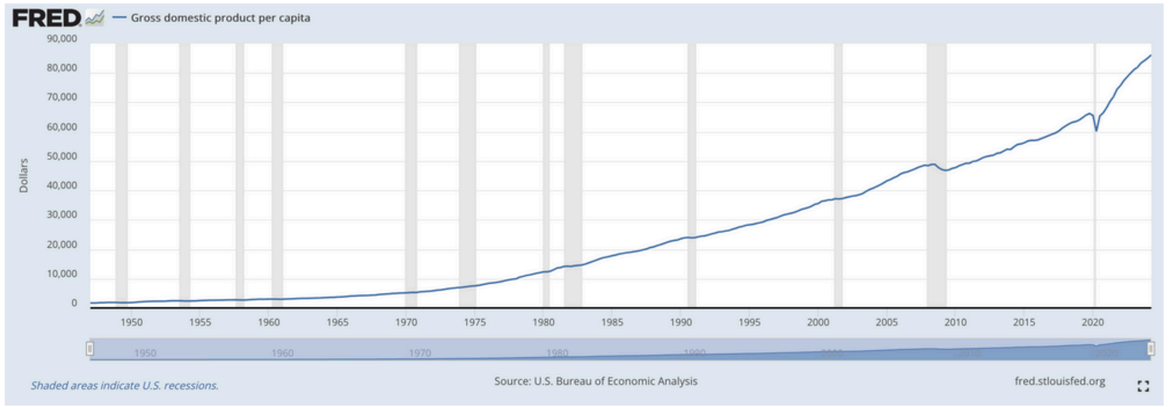
Solution: Smooth then resample

- Use a window size the same size you are downsampling!

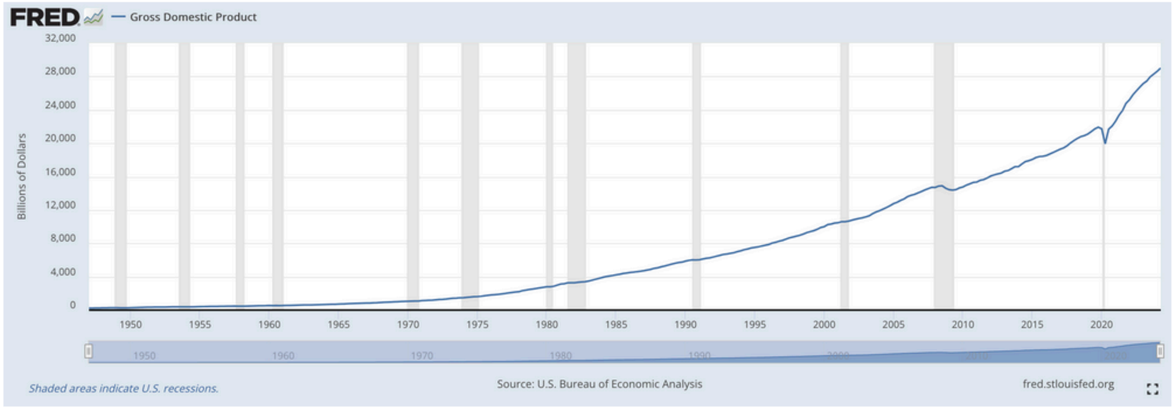


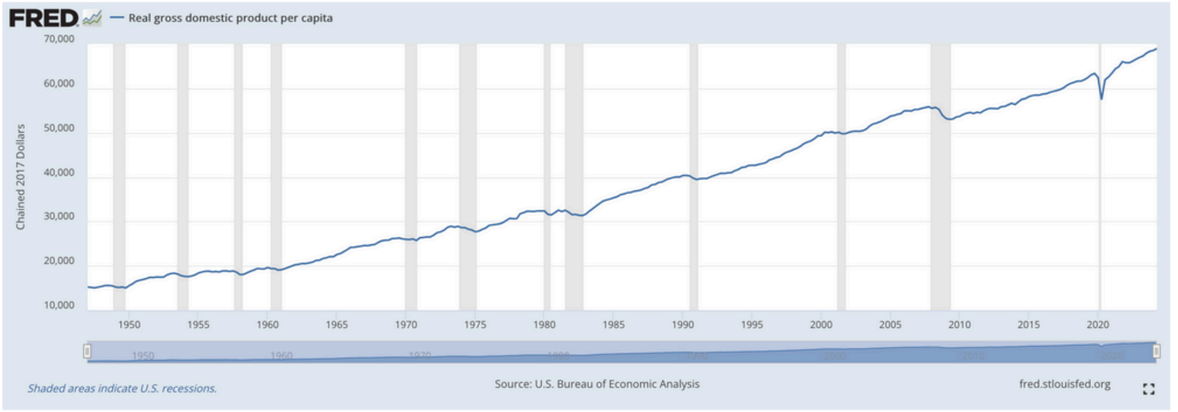
Population Adjustment



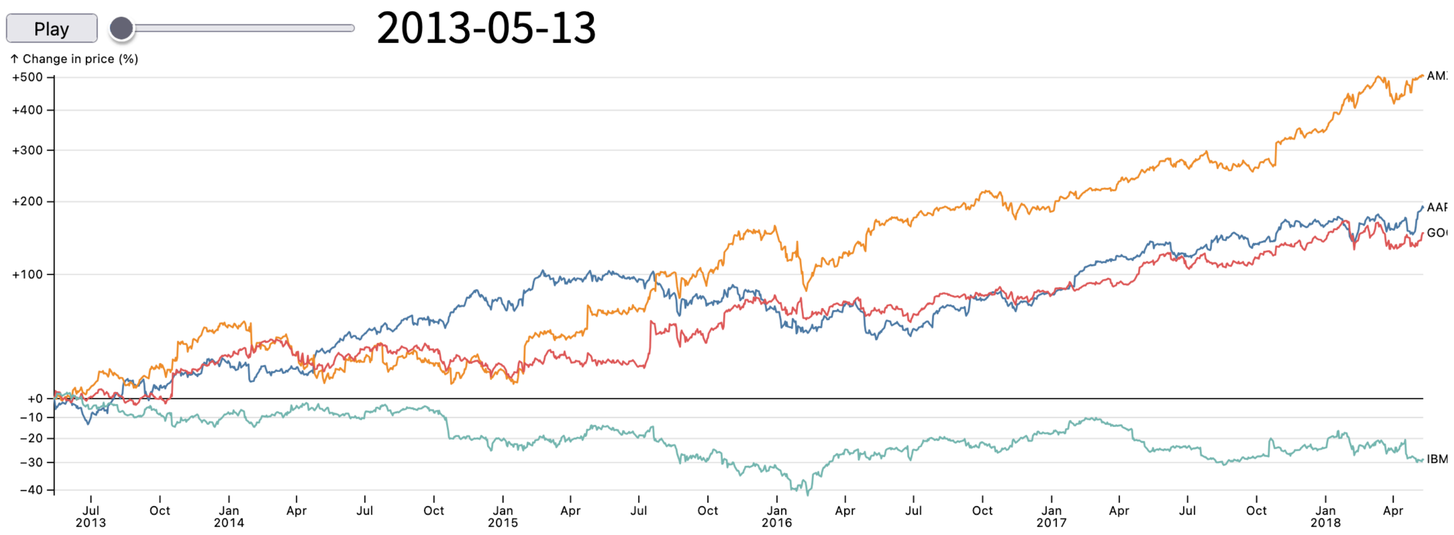


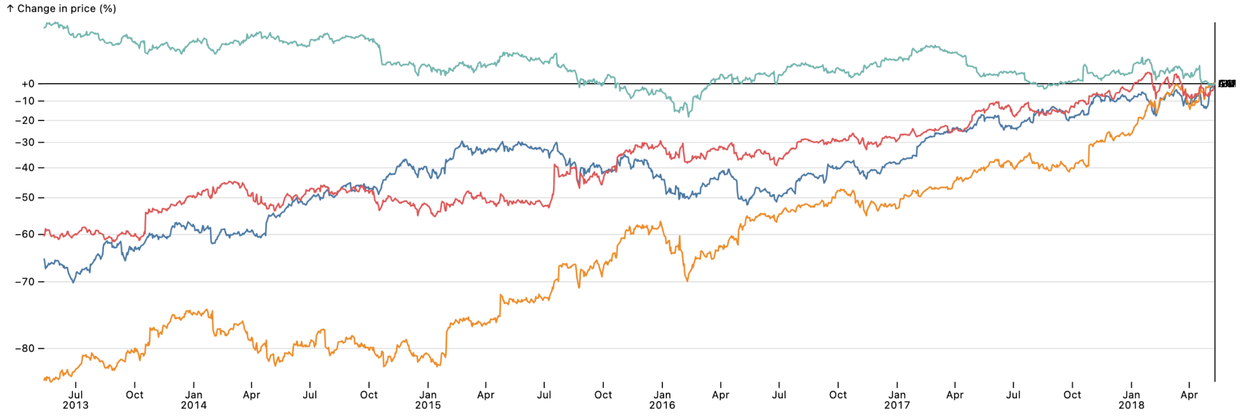
Inflation Adjustment

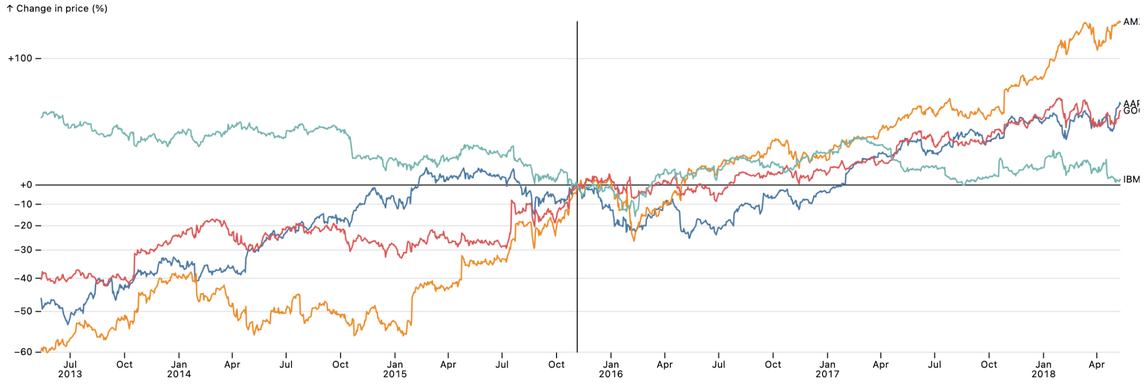




Normalize to time

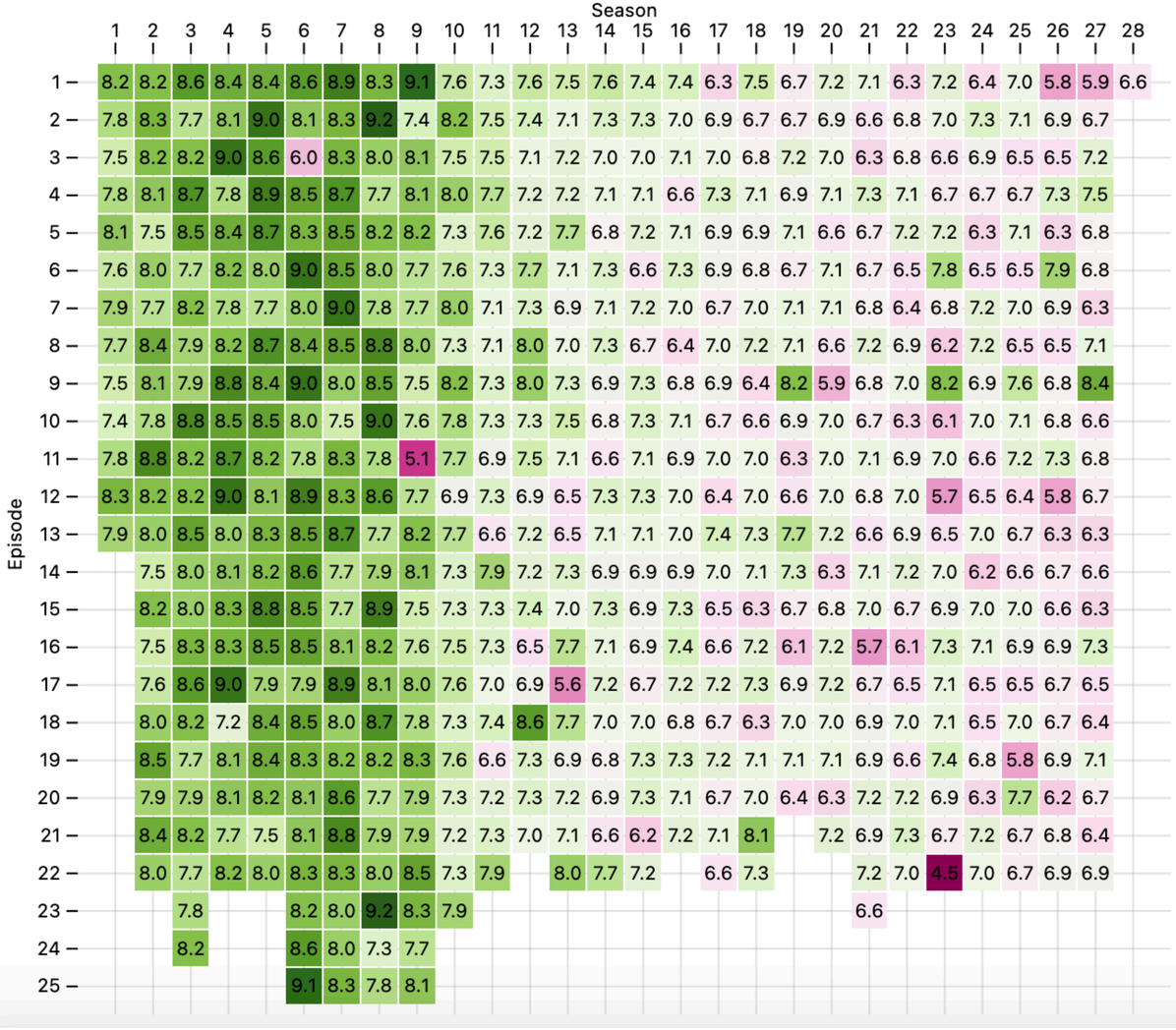






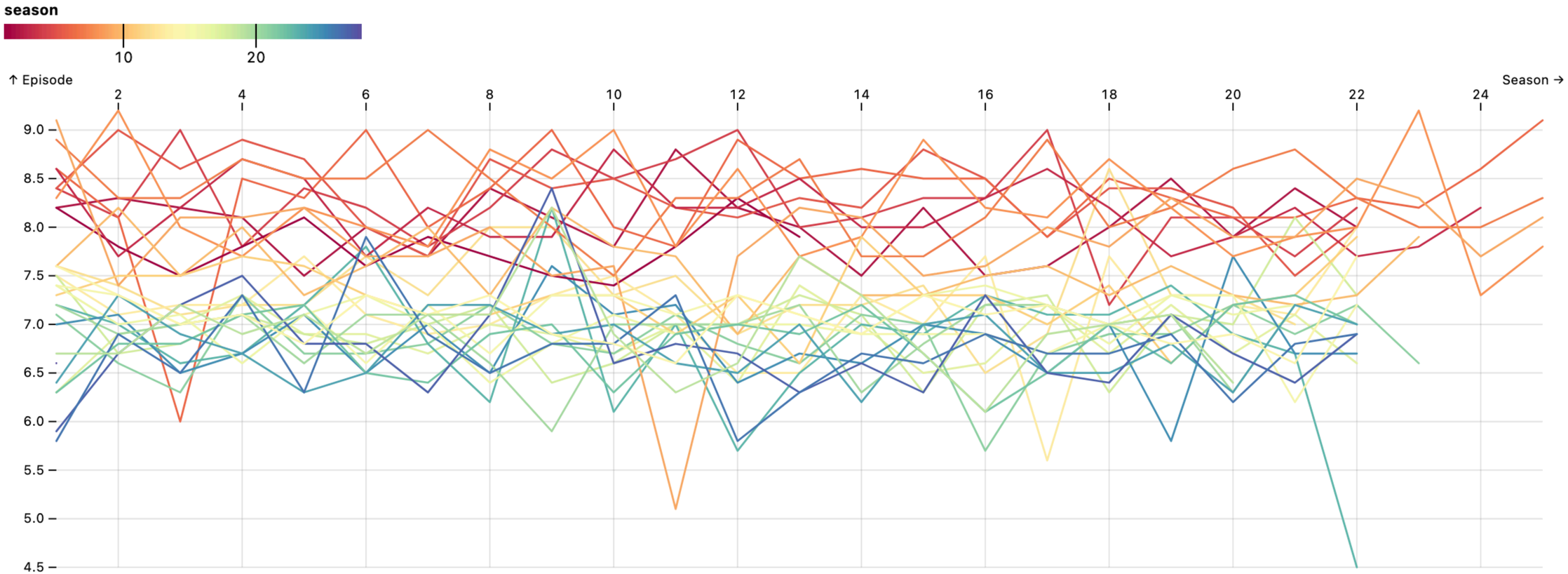
Seasonality

```
Plot.plot({
  padding: 0 ,grid: true, x: {axis: "top", label: "Season"}, y: {label: "Episode"},color: {type: "linear", scheme: "PiYG"},
  marks: [
    Plot.cell(simpsons, {x: "season", y: "number_in_season", fill: "imdb_rating", inset: 0.5}),
    Plot.text(simpsons, {x: "season", y: "number_in_season", text: (d) => d.imdb_rating?.toFixed(1), fill: "black", title: "title"})
  ]})
```

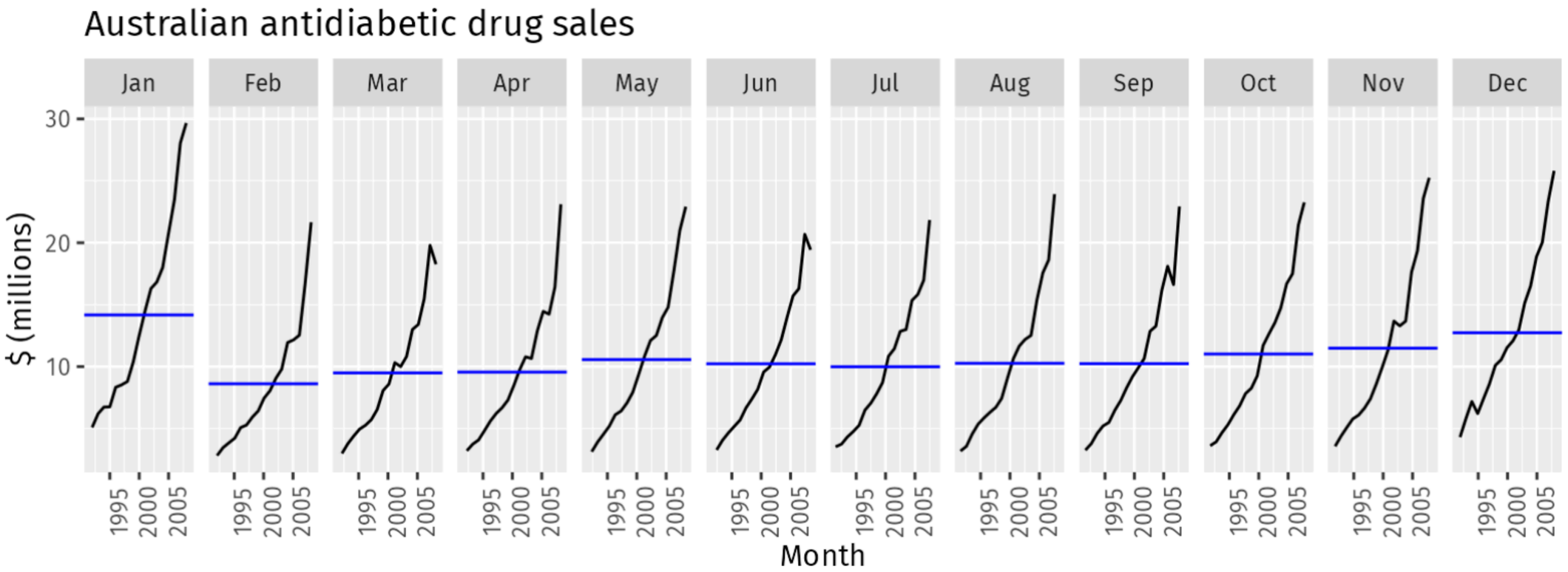


Seasonality

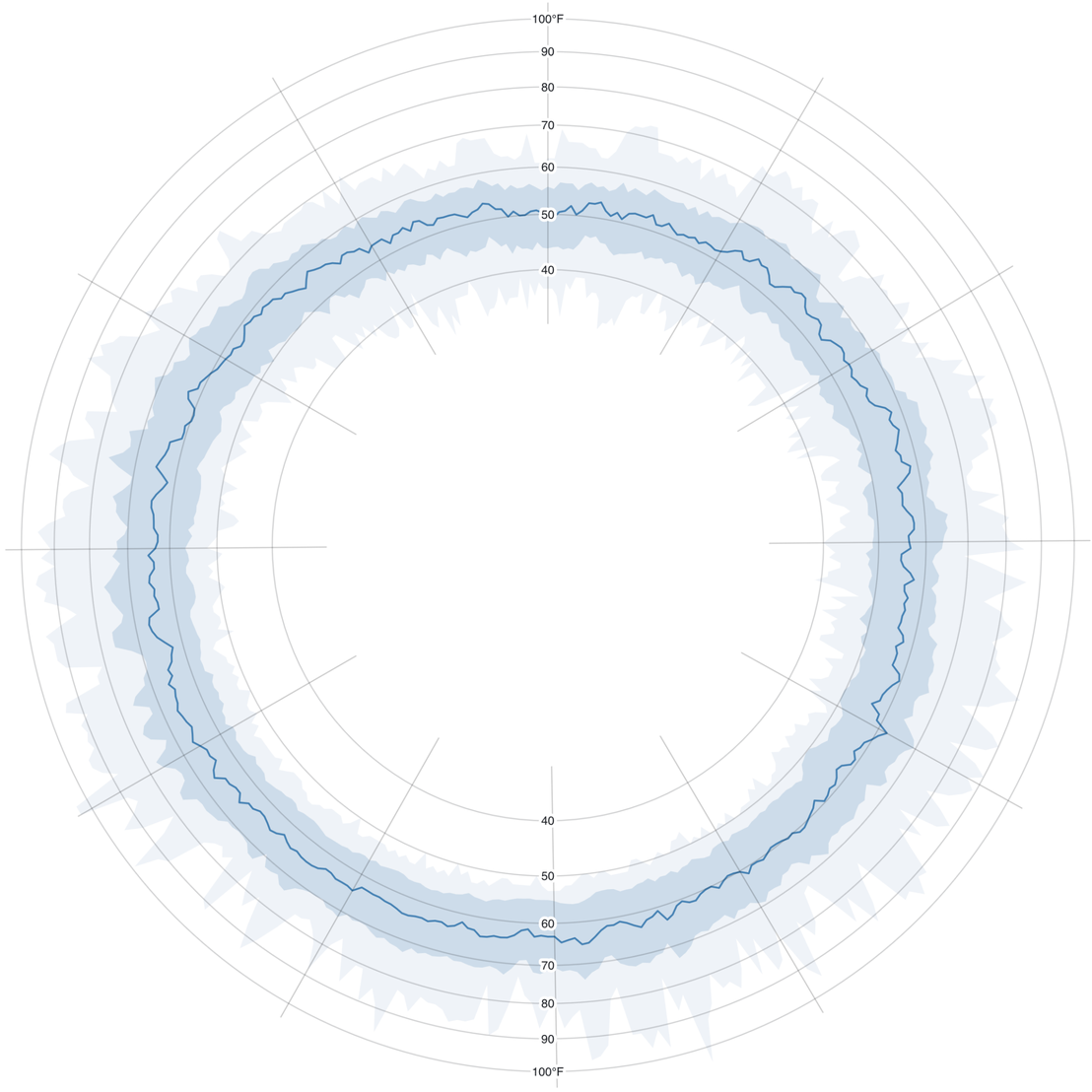
```
Plot.plot({
  padding: 0 ,grid: true, x: {axis: "top", label: "Season"}, y: {label: "Episode"},color: {type: "linear", scheme: "spectral", legend: true}, width: 1
  marks: [
    Plot.lineY(simpsons, Plot.sort("number_in_season", {stroke: "season", x: "number_in_season", y: "imdb_rating", inset: 0.5})),
  ]})
```



Seasonality

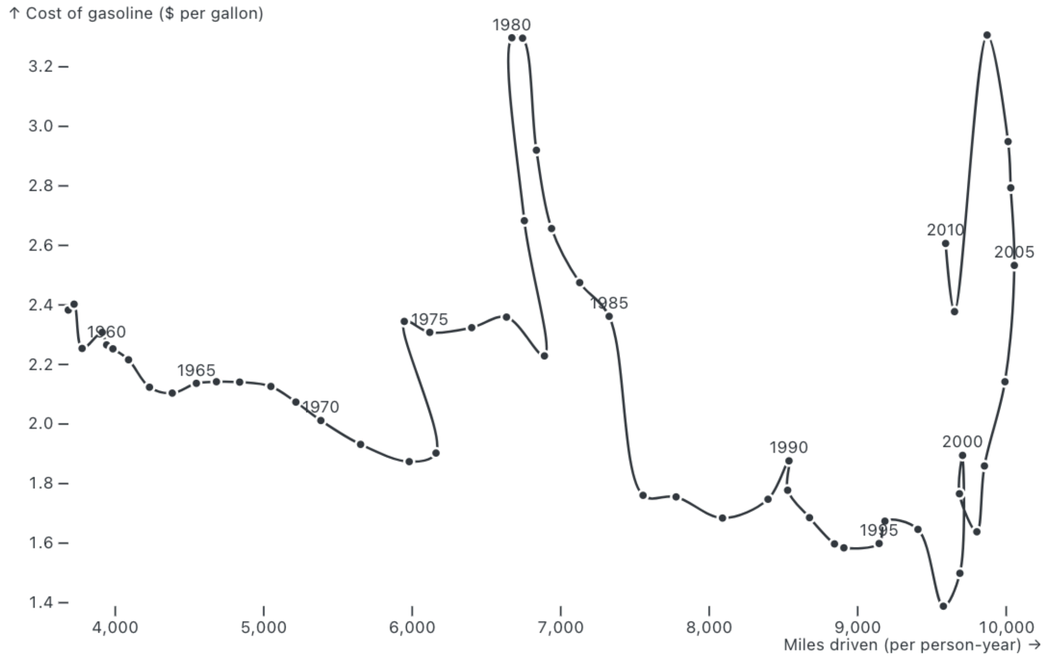


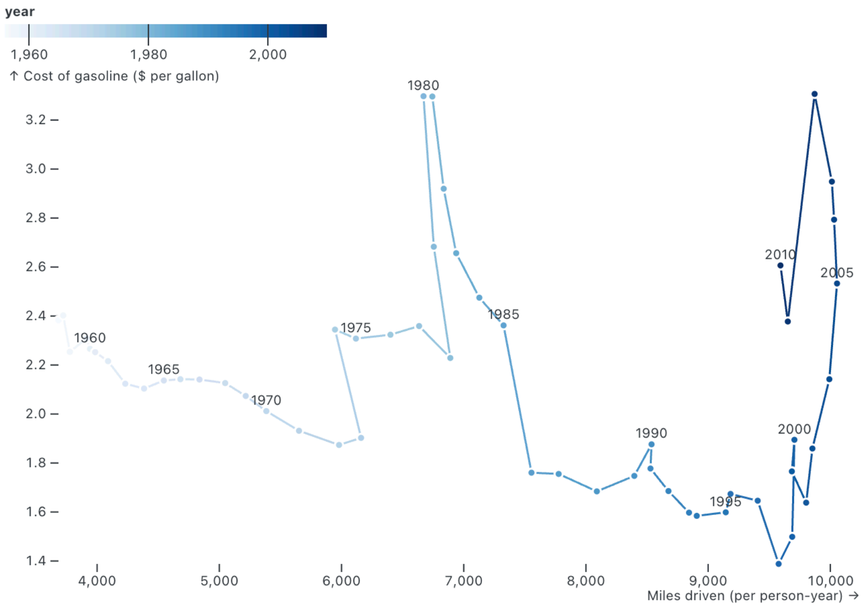
Seasonality



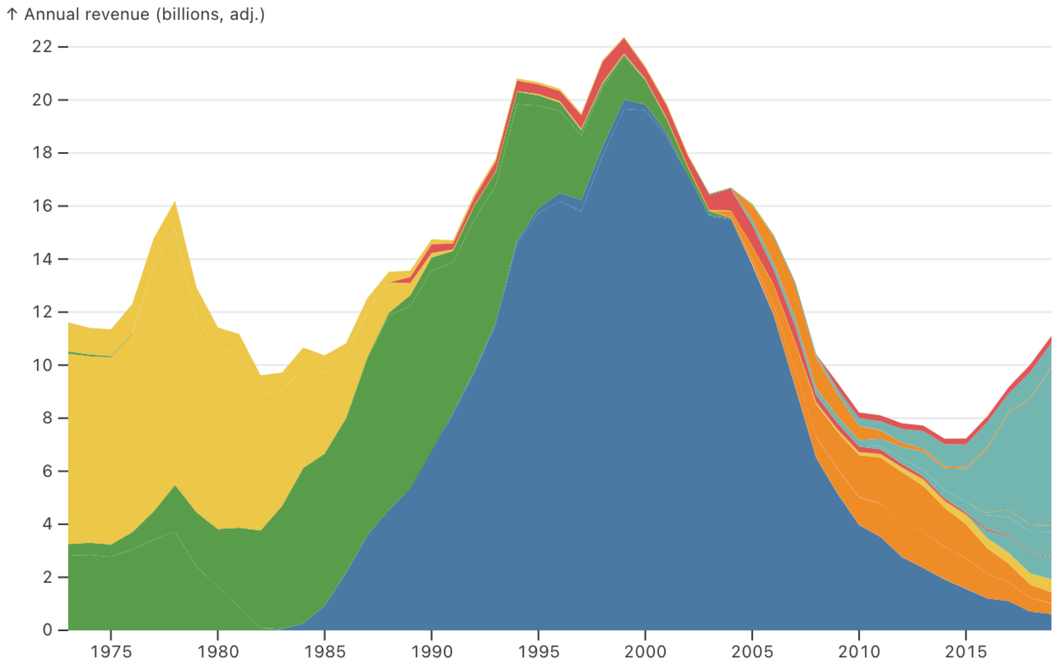
Time as a color axis

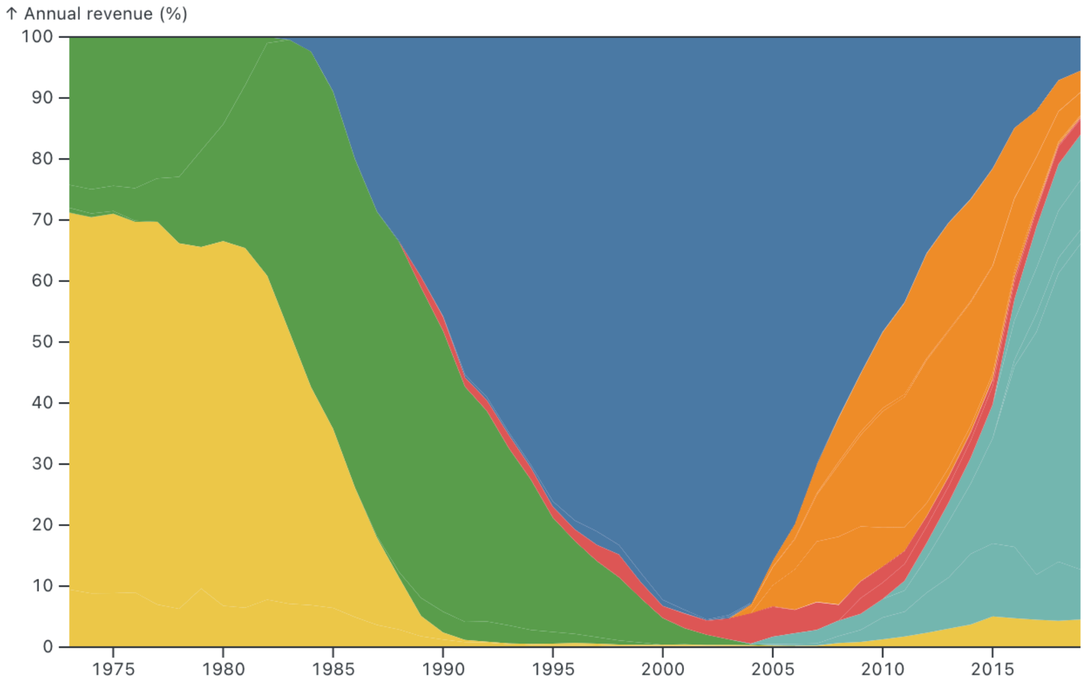
Time as a color axis



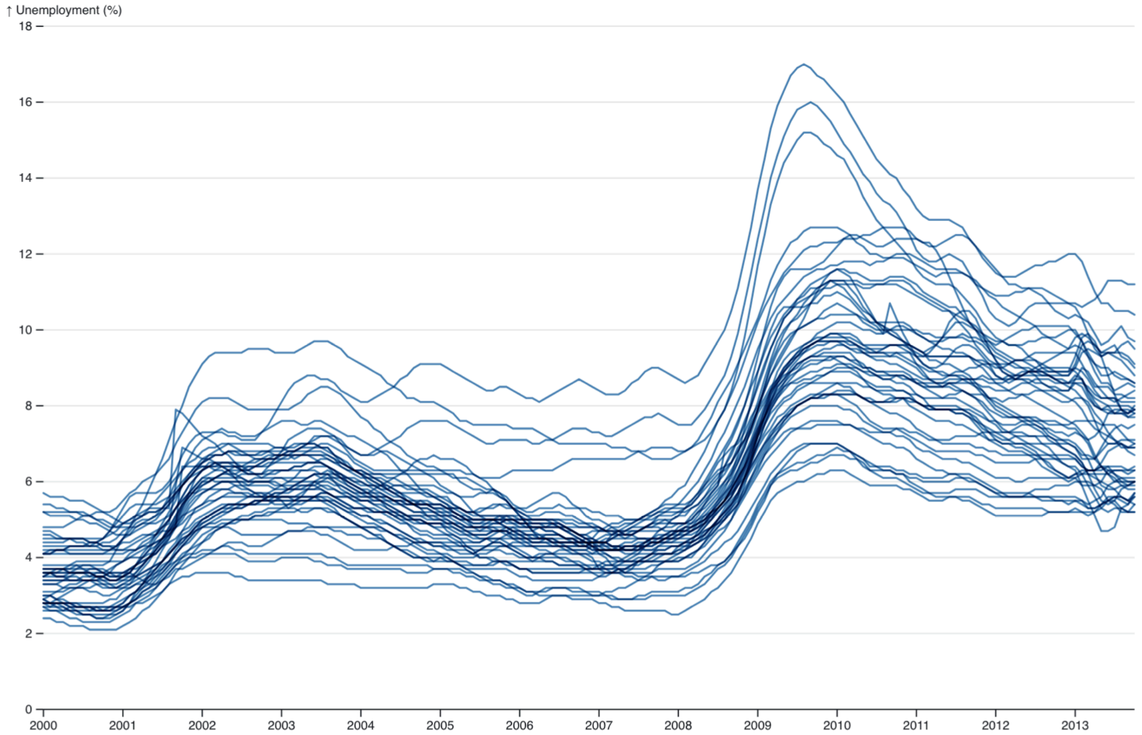


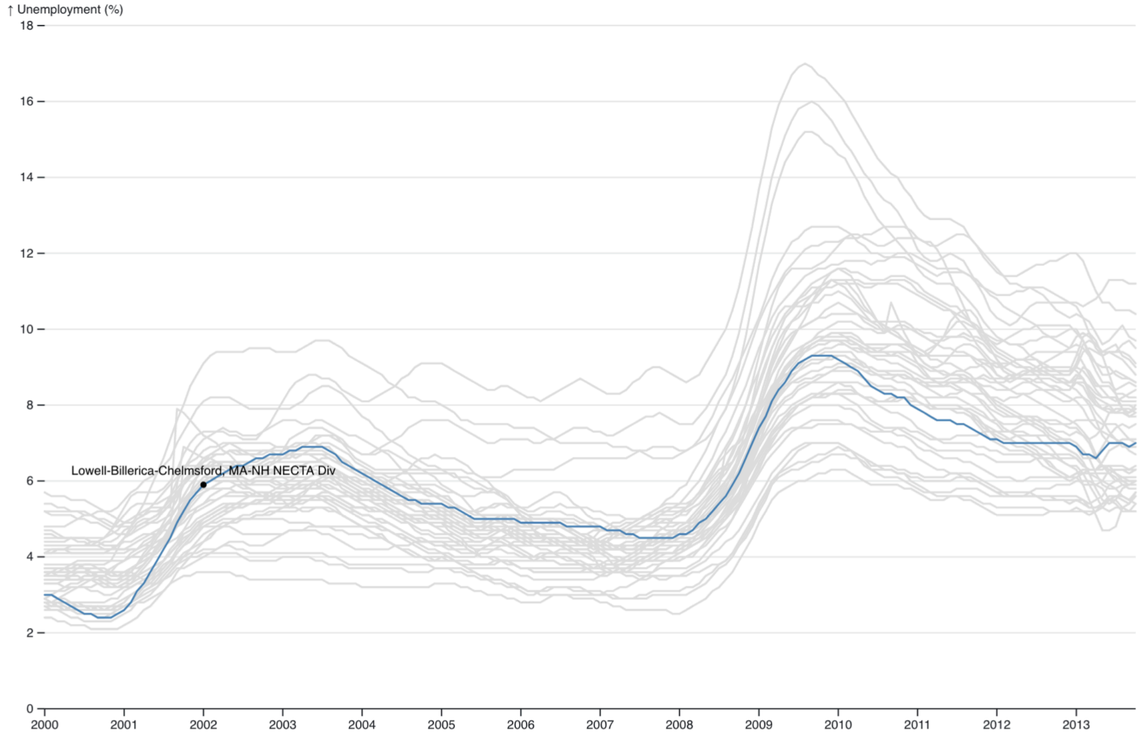
Stacked area and proportion plots





Multi-line plots with highlighting





```
▼ Array(12) [
  0: ▶ Object {date: 2019-01-01T08:00Z, injuries: 2788}
  1: ▶ Object {date: 2019-02-01T08:00Z, injuries: 2443}
  2: ▶ Object {date: 2019-03-01T08:00Z, injuries: 2856}
  3: ▶ Object {date: 2019-04-01T07:00Z, injuries: 2909}
  4: ▶ Object {date: 2019-05-01T07:00Z, injuries: 3485}
  5: ▶ Object {date: 2019-06-01T07:00Z, injuries: 3613}
  6: ▶ Object {date: 2019-07-01T07:00Z, injuries: 3494}
  7: ▶ Object {date: 2019-08-01T07:00Z, injuries: 3248}
  8: ▶ Object {date: 2019-09-01T07:00Z, injuries: 3131}
  9: ▶ Object {date: 2019-10-01T07:00Z, injuries: 3206}
  10: ▶ Object {date: 2019-11-01T07:00Z, injuries: 2890}
  11: ▶ Object {date: 2019-12-01T08:00Z, injuries: 3175}
]
```

```
injuriesByMonth
```

```
dateExtent = ▶ Array(2) [2019-01-01T08:00Z, 2019-12-01T08:00Z]
```

```
dateExtent = d3.extent(injuriesByMonth, d => d.date)
```

```
x = ƒ(n)
```

```
x = d3.scaleTime()
    .domain(dateExtent)
    .range([margin.left, width - margin.right])
```

```
maxInjuries = 3613
```

```
maxInjuries = d3.max(injuriesByMonth, d => d.injuries)
```

```
y = ƒ(n)
```

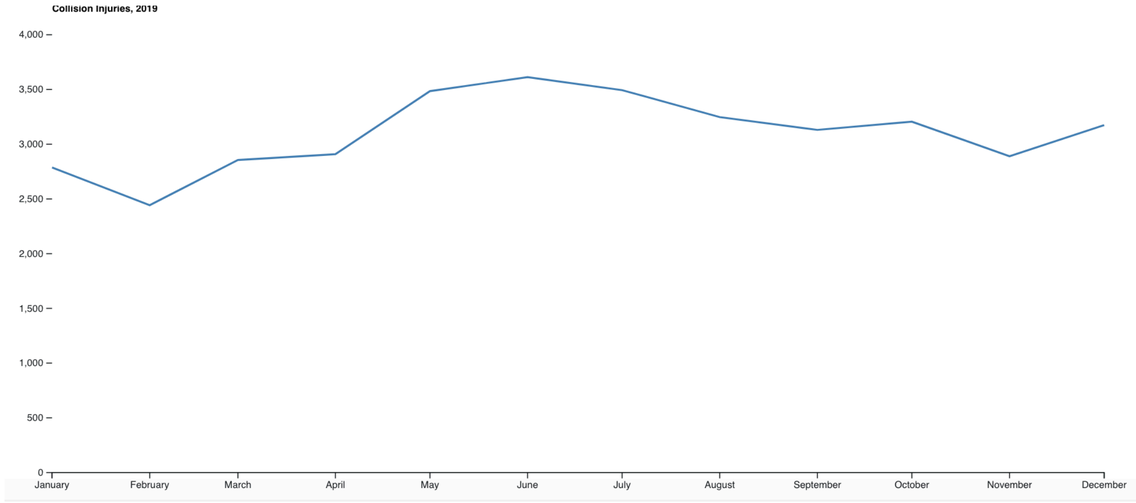
```
y = d3.scaleLinear()
    .domain([0, maxInjuries]).nice()
    .range([height - margin.bottom, margin.top]);
```

```
line = d3.line()
    .x(d => x(d.date))
    .y(d => y(d.injuries))
```

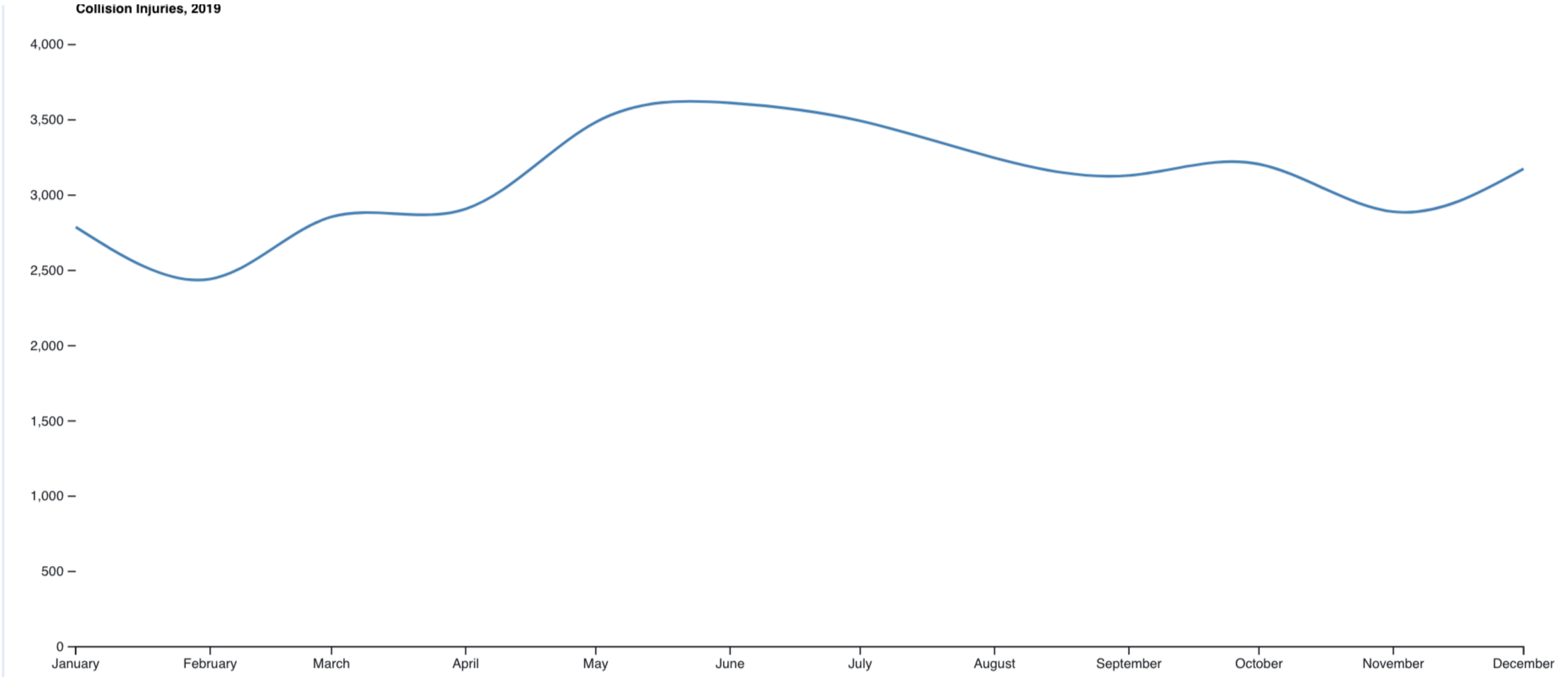
```
"M40,166.35L140.425,205.163L231.132,158.7L331.422,152.738L428.608,87.938L529.033,73.538L626.218,86.925L726.643,114.6L827.069,127.763L924.
```

```
line(injuriesByMonth)
```

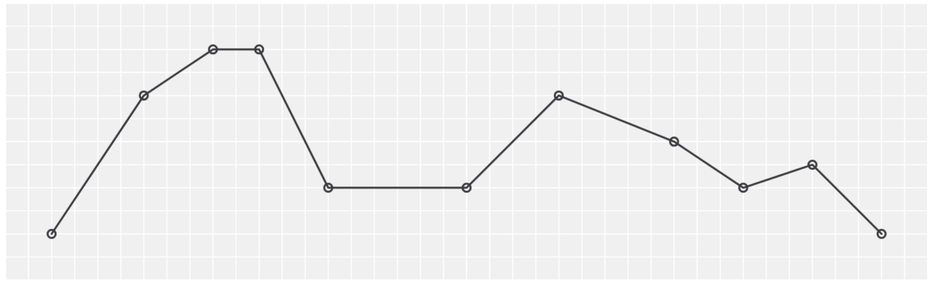
```
// Draw the line
// We are only drawing one line, so we can append one path
svg.append('path')
    .attr('stroke', 'steelblue')
    .attr('stroke-width', 2)
    .attr('fill', 'none')
    .attr('d', line(injuriesByMonth));
```



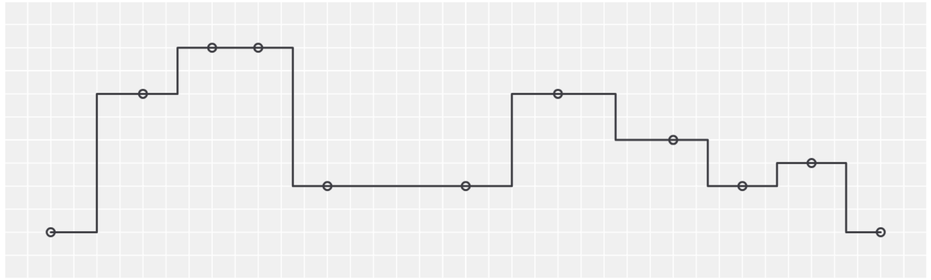
```
line = d3.line()
    .x(d => x(d.date))
    .y(d => y(d.injuries))
    .curve(d3.curveNatural)
```



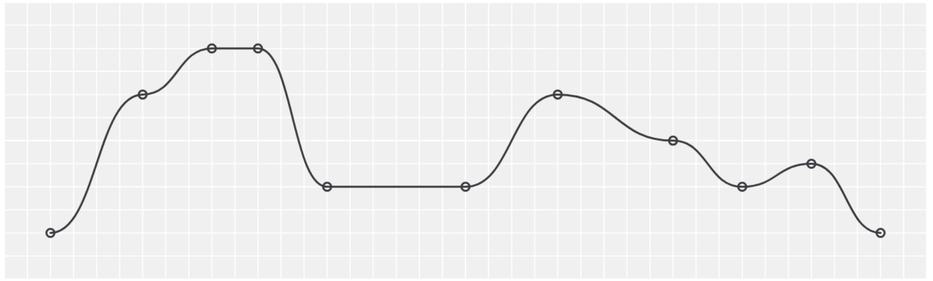
curveLinear(*context*)



curveStep(*context*)



curveBumpX(*context*)



curveNatural(*context*)