Marks Cheatsheet

Observable

Plots are comprised of visual **marks** representing your data.

Creating a mark

Pass in **data** and set the **visual channels**

CODE

Type of mark to draw

Data to display

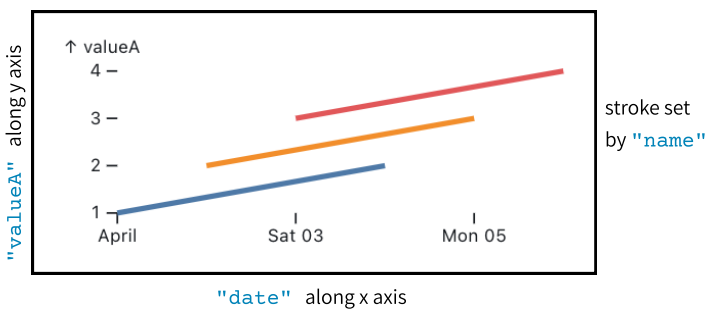
```
Plot.line(data, {
  x: "date",
  y: "valueA",
  stroke: "name"
})
```

x axis channel

y axis channel

color channel

OUTPUT



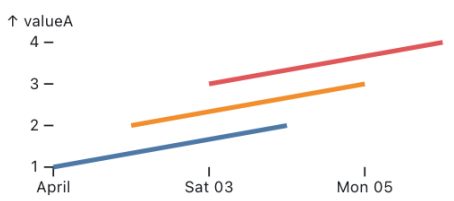
DATA

name	date	valueA	valueB
a	2021-04-01	1	4
b	2021-04-02	2	1
c	2021-04-03	3	3
a	2021-04-04	2	0

Individual Marks

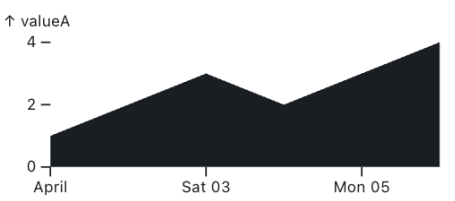
Specify the rendering of each mark by declaring relevant **options** to the mark

line



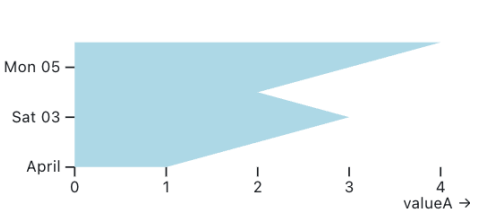

```
Plot.line(data, { x: "date", y: "valueA",
z: "name", strokeWidth: 3, stroke: "name" })
```

areaY




```
Plot.areaY(data, { x: "date", y: "valueA" })
```

areaX

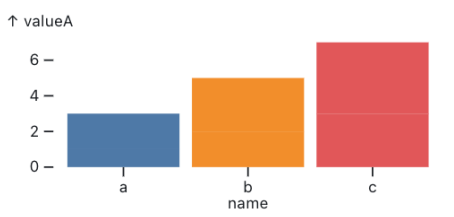



```
Plot.areaX(data, { x: "valueA", y: "date",
fill: "lightblue" })
```

dot

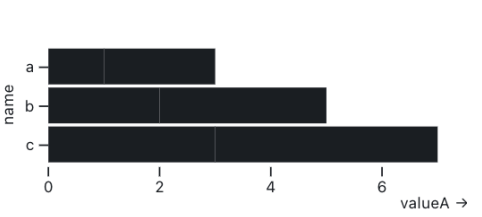

```
Plot.dot(data, { x: "name", y: "valueA",
r: "valueB" })
```

barY



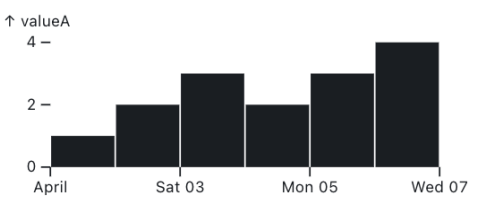

```
Plot.barY(data, { x: "name", y: "valueA",
fill: "name" })
```

barX



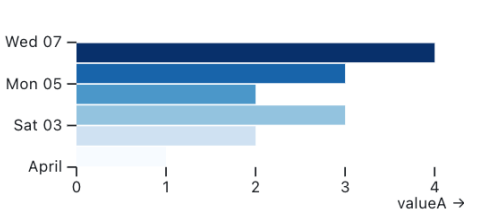

```
Plot.barX(data, { x: "valueA", y: "name" })
```

rectY



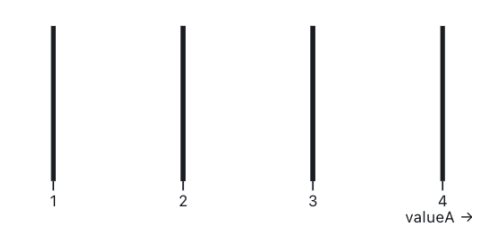

```
Plot.rectY(data, { x: "date", y: "valueA",
interval: d3.utcDay })
```

rectX



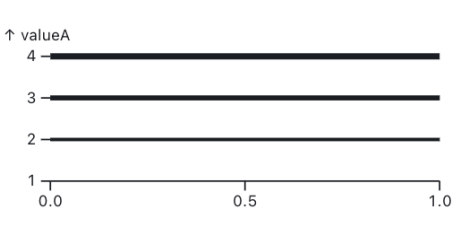

```
Plot.rectX(data, { x: "valueA", y: "date",
interval: d3.utcDay, fill: "date" })
```

ruleX




```
Plot.ruleX(data, { x: "valueA",
strokeWidth: 3 })
```

ruleY

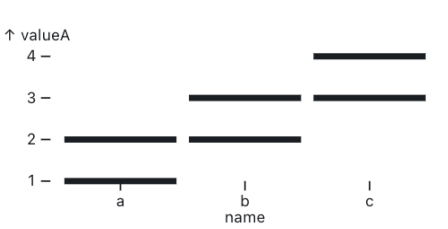



```
Plot.ruleY(data, { y: "valueA",
strokeWidth: "valueA" })
```

tickX


```
Plot.tickX(data, { x: "valueA", y: "name",
strokeWidth: 4, strokeOpacity: 0.5 })
```

tickY

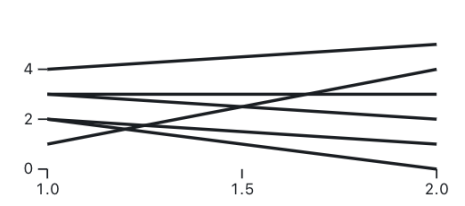



```
Plot.tickY(data, { y: "valueA", x: "name",
strokeWidth: 4 })
```

text

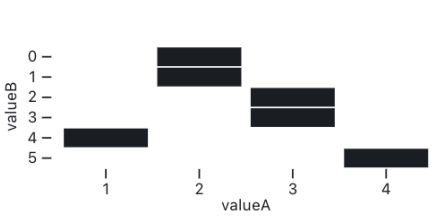

```
Plot.text(data, { x: "date", y: "valueA",
text: "name", fontSize: 25 })
```

link




```
Plot.link(data, { x1: 1, x2: 2, y1: "valueA"
y2: "valueB", strokeWidth: 2 })
```

cell

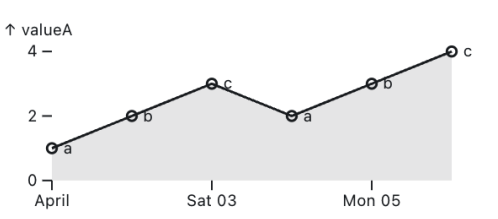



```
Plot.cell(data, { x: "valueA", y: "valueB" }
```

Multiple Marks

Layer multiple marks by passing an **array of marks** into your plot function.

```
Plot.plot({
  marks: [
    Plot.dot(data, { x: "date", y: "valueA" }),
    Plot.line(data, { x: "date", y: "valueA" }),
    Plot.areaY(data, { x: "date", y: "valueA"}),
    Plot.text(data, { x: "date", y: "valueA" })
  ]
})
```



Scales Cheatsheet

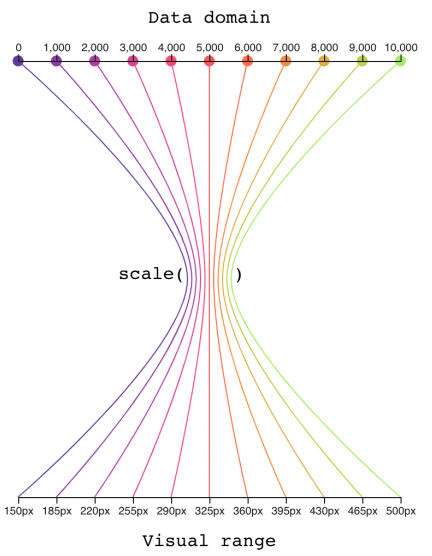
Observable

Scales project your data from an abstract **data domain** to a **visual range**

Working with scales

Configure the scale for each visual channel in your `Plot.plot()` function:

```
Plot.plot({
  // For any channel, configure the scale
  x: {
    type: "log",      // scale type
    ticks: 5,         // # of ticks
    tickFormat: ".2s" // formatting
  }
})
```



SCALE OPTIONS

`axis: "top"` 0.0 0.5 1.0

`domain: [0, .5]` 0.0 0.5

`grid: true` 0.0 0.5 1.0

`inset: 20` 0 1

`line: true` 0.0 0.5 1.0

`percent: true` 0 50 100 (%) →

`reverse: true` 1.0 0.5 0.0

LABELS + TICKS

`label: "My label"` 0.0 0.5 1.0 My label

`labelAnchor: "left"` 0.0 0.5 1.0 My label

`labelOffset: 35` 0.0 0.5 1.0 My label

`nice: true` 0.0 0.5 1.0

`tickPadding: 10` 0.0 0.5 1.0

`tickRotate: -90` 0.0 0.5 1.0

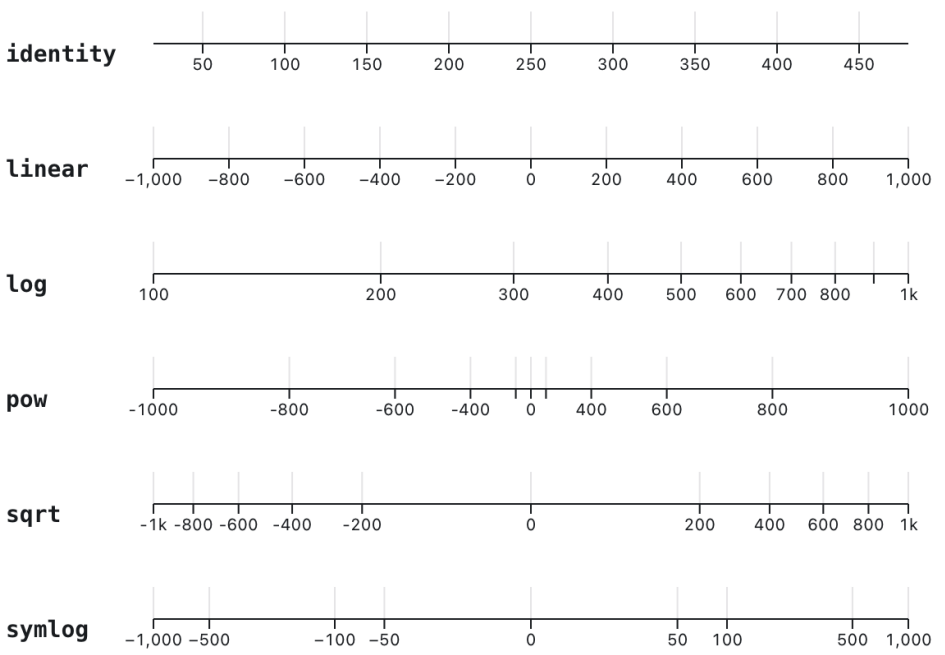
`tickSize: 10` 0.0 0.5 1.0

`ticks: 2` 0 1

Quantitative

Display continuous data by setting one of these **types**

```
Plot.plot( x: { type: “identity” })
```



Specify a `tickFormat:"[symbol][comma][precision][type]"`

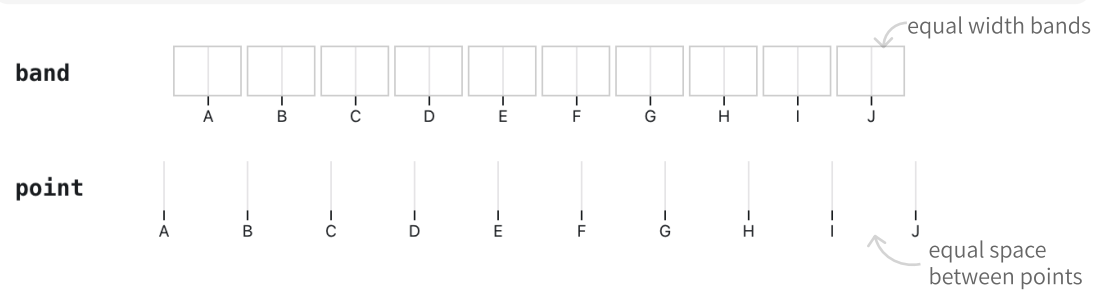
```
tickFormat: “0.2sf”
```

Syntax	Description	format(0.00013)	format(543005)
$	Currency symbol	$0.00013	$543005
s	International System of Units (SI).	130.000µ	543.005k
e	Exponent notation	1.300000e-4	5.430050e+5
f	Fixed point notation	0.000130	543005.000000
p	Percentage notation	0.0130000%	54300500%
,	Comma separated	0.00013	543,005
.2	Precision of 2 digits	0.00013	5.4e+5
.5	Precision of 5 digits	0.00013	5.4301e+5
.2s	Two significant digits, shown in SI.	130µ	540k
,.1f	Comma separated, one fixed value after the decimal place	0.0	543,005.0
,.1p	Comma separated, one digit, percentage type	0.01%	50,000,000%
$,.1	Currency syntax, Comma separated, one digit, percentage type	$0.0001	$5e+5

Categorical

Display categorical data by setting one of these **types**

```
Plot.plot( x: { type: “band” })
```



Customize your ticks using a function

```
tickFormat: d => `Group ${d}`
```

```
tickFormat: d => `Group ${d}`
```
the **value** of each tick

Group A Group B Group C

```
tickFormat: d => d.toLowerCase()
```

a b c

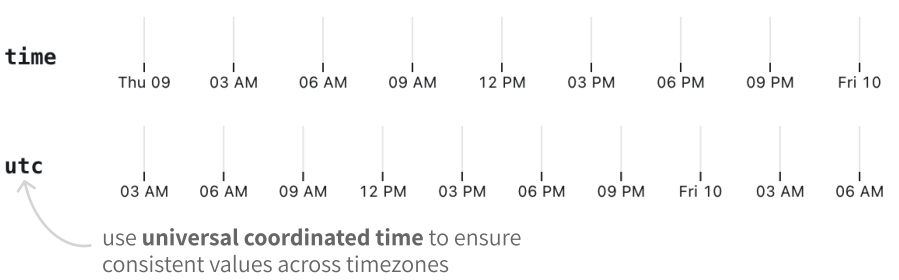
```
tickFormat: (d, i) => `${d}: ${i}`
```
the **index** of each tick

A: 0 B: 1 C: 2

Date

Display temporal data by setting one of these **types**

```
Plot.plot( x: { type: “utc” })
```



Compose a time formatter using the syntax below

```
tickFormat: d3.utcFormat(“%A %B %d, %Y”)
```

year	month	day	hour	minute
%y 21	%b Nov	%a Tue	%H 00	%M 00
%Y 2021	%B November	%A Tuesday	%I 12	
	%m 11	%d 16		

combine formats into a single string

"%A %B %d, %Y"	Friday January 01, 2021
"%b. %d, %Y"	Jan. 01, 2021
"%d-%m-%y"	01-01-21

Transforms Cheatsheet

Augment your data for plotting

Observable

PLOT version 0.3.2

Group into categories

`Plot.group, Plot.groupX, Plot.groupY, Plot.groupZ`

Compute a summary value for each group

```
Plot.groupY({ x: "mean" }, { x: "sales", y: "category" })
```

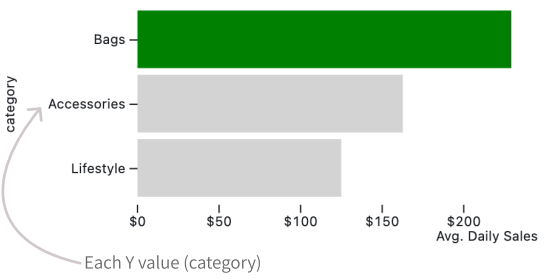
Compute the mean of the x

Compute for each y group

date	category	sales
Nov 20	Accessories	$334
Nov 20	Bags	$90
Nov 20	Lifestyle	$138
Nov 21	Accessories	$250
Nov 21	Bags	$276
Nov 21	Lifestyle	$64
Nov 22	Accessories	$72
Nov 22	Bags	$80
Nov 22	Lifestyle	$162

mean

category	sales
Bags	$229
Accessories	$163
Lifestyle	$125




```
Plot.barX(data, Plot.groupY({ x: "mean" }, { x: "sales", y: "category" }))
```

Create bars of your grouped data

Bin into intervals

`Plot.bin, Plot.binX, Plot.binY`

Group into intervals and compute summary values

```
Plot.binY({ x: "count"}, { y: "sales"})
```

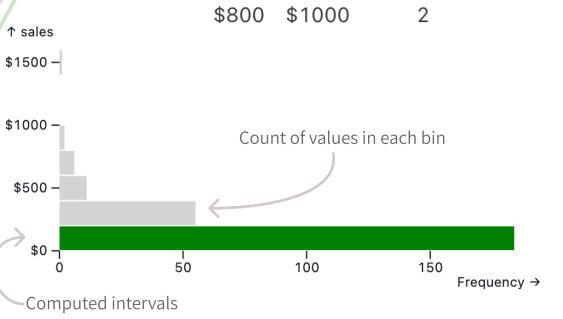
Count observations in each interval

Create intervals based on sales

date	category	sales
Nov 20	Accessories	$334
Nov 20	Bags	$90
Nov 20	Lifestyle	$138
Nov 21	Accessories	$250
Nov 21	Bags	$276
Nov 21	Lifestyle	$64
Nov 22	Accessories	$72
Nov 22	Bags	$80
Nov 22	Lifestyle	$162

count

y1	y2	count
$0	$200	184
$200	$400	55
$400	$600	11
$600	$800	6
$800	$1000	2




```
Plot.rectX(data, Plot.binY({ x: "count"}, { y: "sales"}))
```

Create a histogram of the sales

Tip: customize the intervals using **thresholds:** a number (10), cutpoints (`[0, 10, 50]`), or a function (`d3.utcDay`)

Window to smooth values

`Plot.window, Plot.windowX, Plot.windowY`

Compute moving averages across values

```
Plot.windowY({ reduce: "mean", k: 7, anchor: "start" })
```

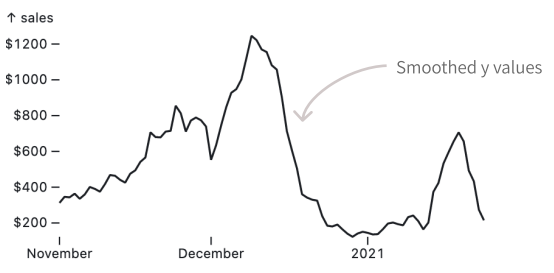
Window size

Compute the mean across values

Align the window at the interval's start

date	sales
Dec 01	$584
Dec 02	$622
Dec 03	$627
Dec 04	$668
Dec 05	$709
Dec 06	$100
Dec 07	$560
Dec 08	$1160
Dec 09	$1397
Dec 10	$1338

date	7-day avg.
Dec 01	$553
Dec 02	$635
Dec 03	$746
Dec 04	$847
Dec 05	$928




```
Plot.line(data, Plot.windowY({ reduce: "mean", k: 7, anchor: "start" }, { x: "date", y: "sales" }))
```

Show a moving average by day

Stack data into layers

`Plot.stack, Plot.stackX, Plot.stackX1, Plot.stackX2, Plot.stackY, Plot.stackY1 Plot.stackY2`

```
Plot.barY(data, { x: "date", y: "sales", fill: "category"})
```

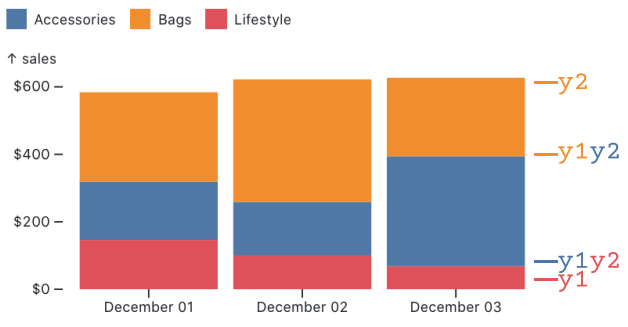
Values are stacked by default

starts on top of previous

date	category	sales	y1	y2
Dec 01	Lifestyle	$146	$0	$146
Dec 01	Accessories	$173	$146	$319
Dec 01	Bags	$265	$319	$584
Dec 02	Lifestyle	$100	$0	$100
Dec 02	Accessories	$159	$100	$259
Dec 02	Bags	$363	$259	$622
Dec 03	Lifestyle	$68	$0	$68
Dec 03	Accessories	$326	$68	$394
Dec 03	Bags	$233	$394	$627

y2 = y1 + sales

stack y values for each x




```
Plot.text(data, Plot.stackY({ x: "date", y: "sales", text: "category" }))
```

Explicitly stack marks like text to align with bars

Normalize relative to a reference

`Plot.normalize, Plot.normalizeX, Plot.normalizeY`

Divide each value by a common denominator

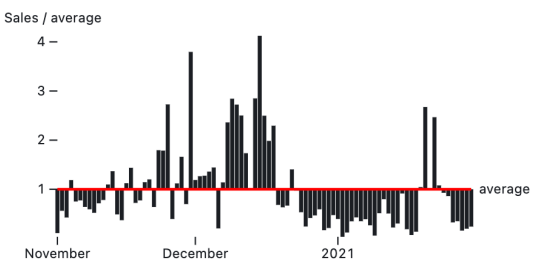
```
Plot.normalizeY({ basis: "mean", x: "date", y: "sales" })
```

Demoninator for each y value

date	sales
Dec 01	$584
Dec 02	$622
Dec 03	$627
Dec 04	$668
Dec 05	$709
Dec 06	$100
Dec 07	$560
Dec 08	$1160
Dec 09	$1397
Dec 10	$1338

Divide each value by the mean

date	sales / avg
Dec 01	1.2x
Dec 02	1.3x
Dec 03	1.3x
Dec 04	1.4x
Dec 05	1.4x
Dec 06	0.2x
Dec 07	1.1x
Dec 08	2.4x
Dec 09	2.8x
Dec 10	2.7x




```
Plot.ruleX(data, Plot.normalizeY({ basis: "mean", x: "date", y1: 1, y2: "sales"}))
```

Show normalized values above / below average

Select values to plot

`Plot.selectFirst, Plot.selectLast, Plot.selectMaxX, Plot.selectMaxY, Plot.selectMinX, Plot.selectMinY`

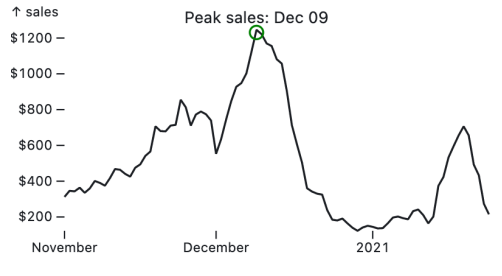
Choose values to plot

```
Plot.selectMaxY({x: "date", y: "sales"})
```

Select observation with highest y value

date	sales
Dec 01	$584
Dec 02	$622
Dec 03	$627
Dec 04	$668
Dec 05	$709
Dec 06	$100
Dec 07	$560
Dec 08	$1160
Dec 09	$1397
Dec 10	$1338

date	sales
Dec 09	$1397




```
Plot.dot(data, Plot.selectMaxY({ x: "date", y: "sales" }))
```

Highlight highest point in the dataset

Layouts Cheatsheet

Adjust the sizing and spacing of your plot

Observable

PLOT

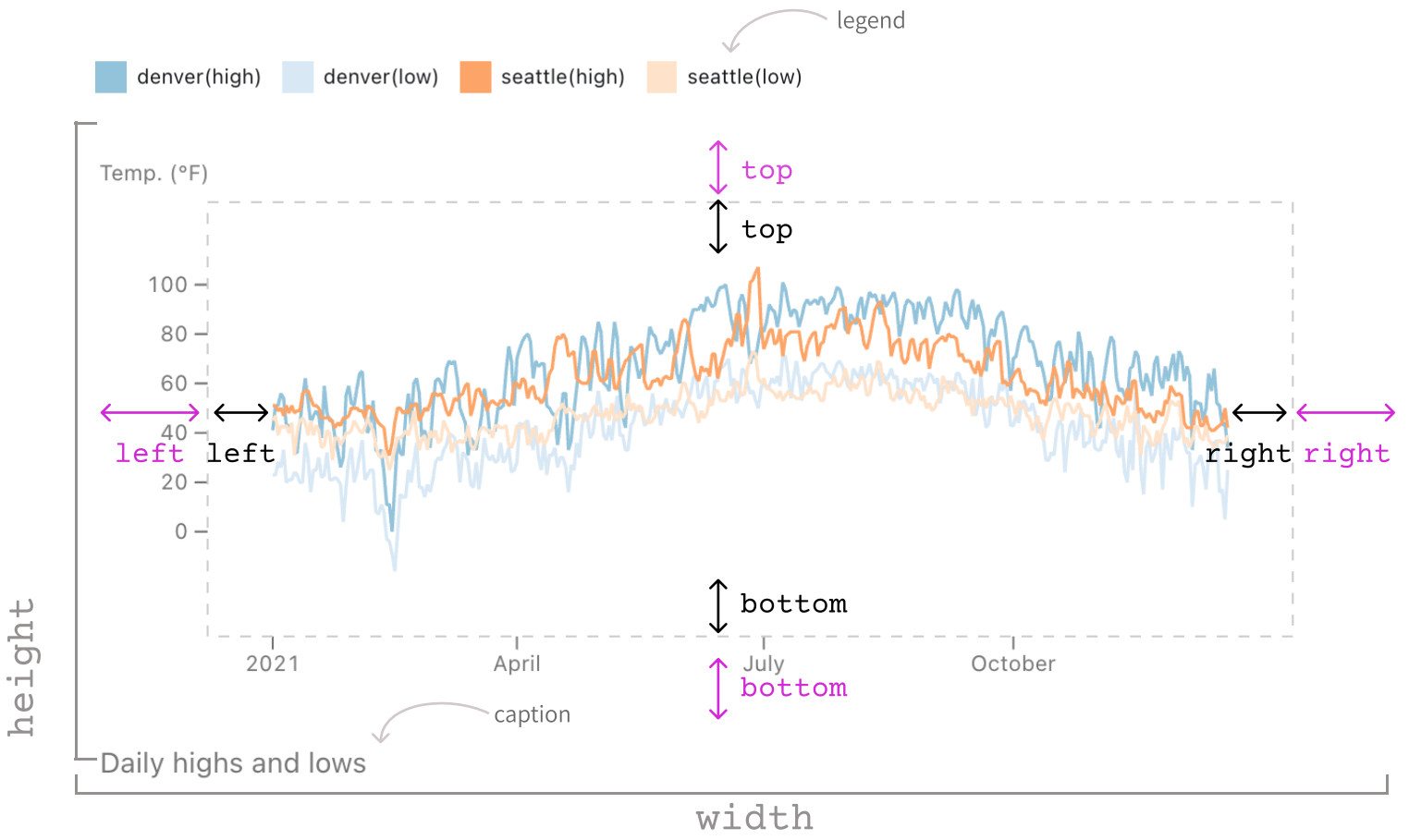
Sizing and Spacing

Adjust plot layout

```
Plot.plot({
  margin: 50,     // space around (all sides)
  insetTop: 30,   // space within (top only)
  width: 640,     // width of plot
  height: 400,    // height of plot
  caption: "Daily highs and lows",
  color: {
    legend: true // include a legend
  },
  marks: []
})
```

margin: space around chart

inset: space between marks and axes



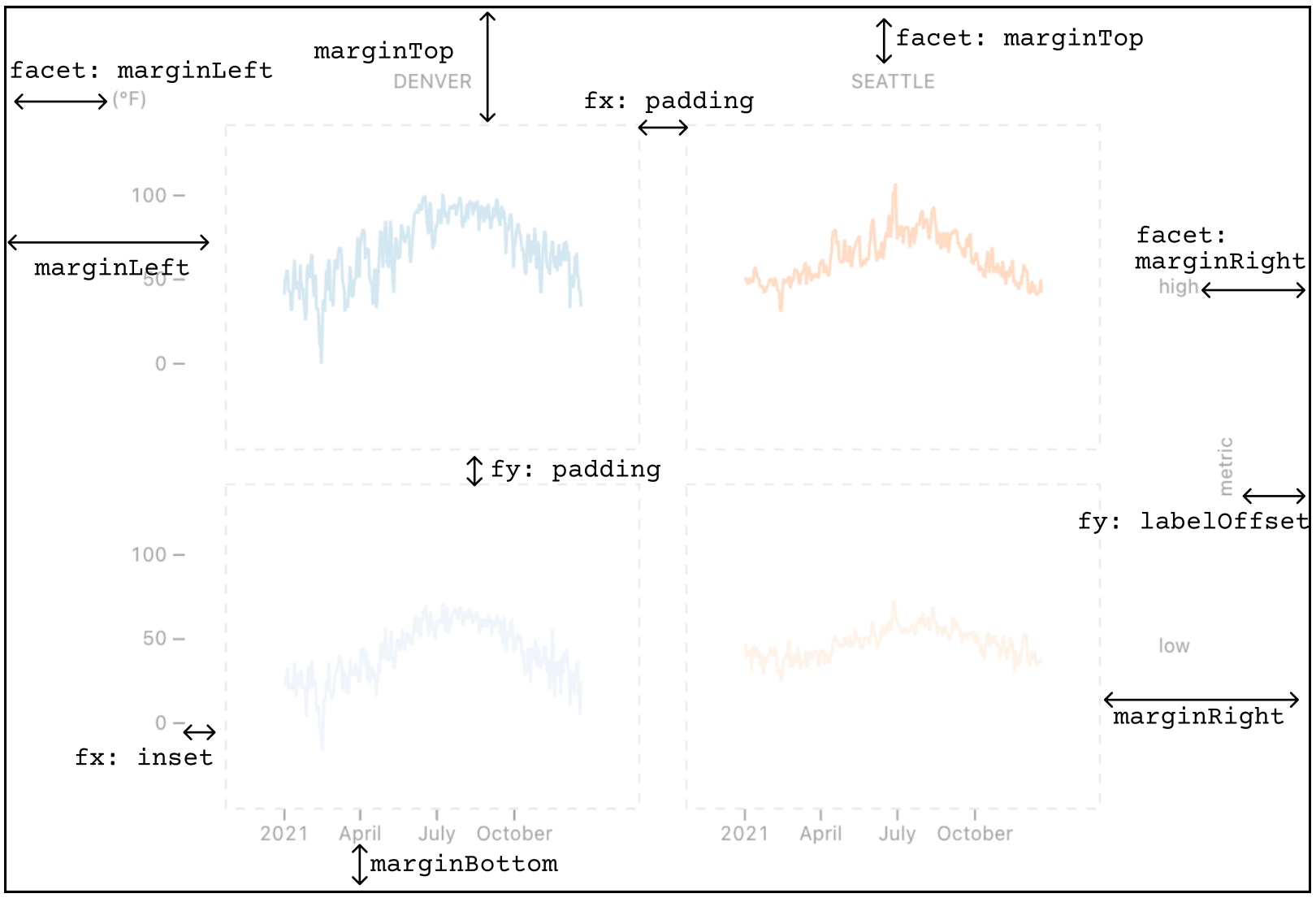
Faceting

Break a plot into small multiples

```
Plot.plot({
  facet: {
    data: data,    // pass data for faceting
    x: "location", // by `location` in the x direction
    y: "metric"    // by `metric` in the y direction
  },
  // Customize the x facet layout and scale
  fx: {
    tickFormat: (d) => d.toUpperCase(),
    padding: 0.1,
    insetRight: 50,
    label: null
  },
  // Customize the y facet layout and scale
  fy: {
    insetTop: 40,
    label: "Daily value"
  }
})
```

padding between charts

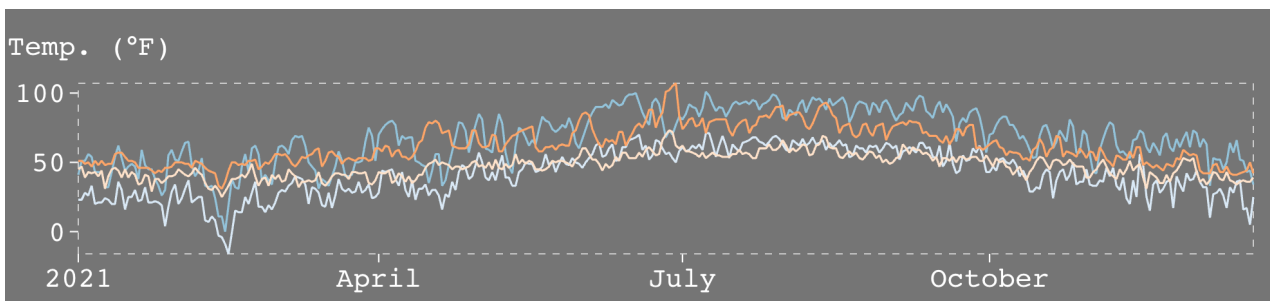
inset from edges



Styles

Customize plot styles

```
Plot.plot({
  style: {
    background: "grey",
    fontSize: 20,
    fontFamily: "monospace",
    color: "white",
    padding: "5px"
  }
})
```



Colors Cheatsheet

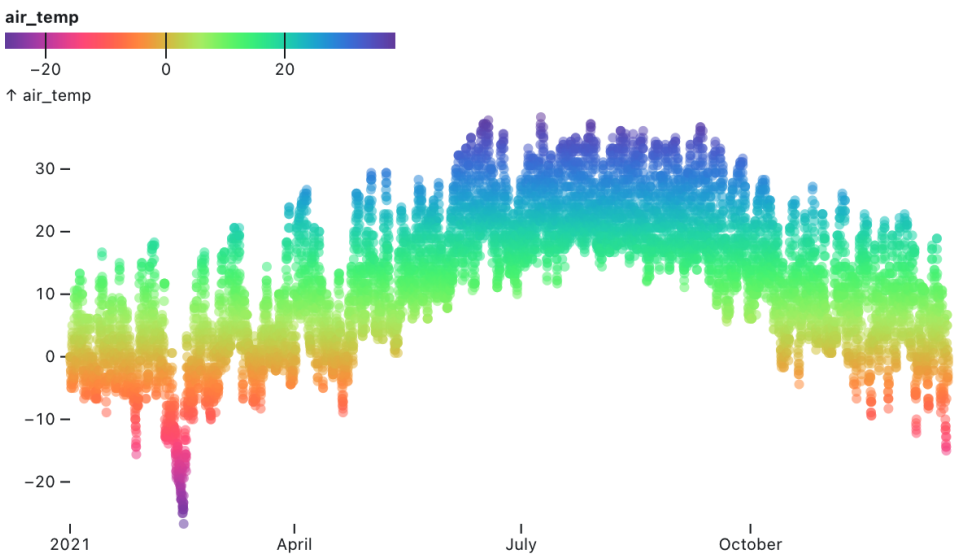
Set the colors of your marks

Observable

Color scales map from **data values** to an **output range** of colors

Set colors by choosing a scheme (below) or declaring a range of color

```
Plot.plot({
  marks: [
    Plot.dot(data, {
      x: "date",
      y: "air_temp",
      fill: "air_temp",
      fillOpacity: 0.5
    })
  ],
  color: {
    scheme: "rainbow",
  }
})
```



Multi-hue

turbo

viridis

magma

inferno

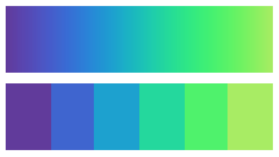
plasma

cividis

cubehelix

warm

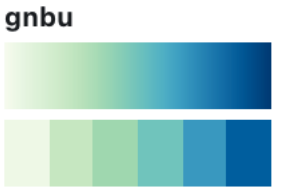
cool



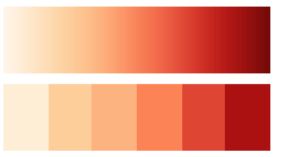
bugn

bupu

gnbu



orrd



pubu

pubugn

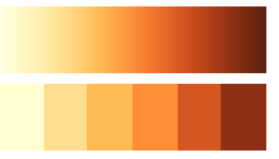
purd

rdpu

ylgn

ylgnbu

ylorbr



ylorrd

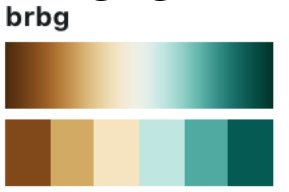
Cyclical

rainbow

sinebow

Diverging

brbg



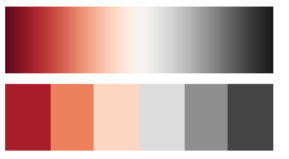
prgn

piyg

puor

rdbu

rdgy



rdylbu

rdylgn

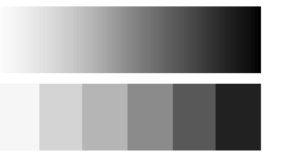
spectral

Single hue

blues

greens

greys



oranges

purples

reds

Categorical

accent

category10

dark2

paired

pastel1

pastel2

set1

set2

set3

tableau10

burd

buylrd