Data 550: Data Visualization I

Lecture 3: Multi-View Composition

Dr. Irene Vrbik

University of British Columbia Okanagan

https://github.com/ubco-mds-2022/Data-550

Introduction

- So far we've been mapping or encoding features to various encoding channels (eg. x-axis, y-axis, colour, shape, …).
- It may be tempting to keep layering on channels when visualizing multiple data fields.
- However, as the number of encoding channels increases, a chart can rapidly become cluttered and difficult to read.
- As a alternative to "over-loading" a single chart we compose *multiple charts* in a way that facilitates rapid comparisons.

https://github.com/ubco-mds-2022/Data-550

Outline

Some more advanced plotting options

- Time Units, Area marks for ranges, Data Transformations

Multi-view compositions operations:

- layer: place compatible charts directly on top of each other,
- stacking partition data into multiple charts
- facet: partition data into multiple charts
- concatenate: position arbitrary charts within a shared layout

https://github.com/ubco-mds-2022/Data-550

Data

https://github.com/ubco-mds-2022/Data-550

Weather Data

Today we will be visualizing weather statistics for the U.S. cities of Seattle and New York.

```
1 import pandas as pd
2 import altair as alt
3
4 weather = 'https://cdn.jsdelivr.net/npm/vega-datasets@1/data/weather.csv'
5 df = pd.read_csv(weather, parse_dates=['date'])
6 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2922 entries, 0 to 2921
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   location       2922 non-null   object
 1   date           2922 non-null   datetime64[ns]
 2   precipitation  2922 non-null   float64
 3   temp_max       2922 non-null   float64
 4   temp_min       2922 non-null   float64
 5   wind           2922 non-null   float64
 6   weather        2922 non-null   object
```

https://github.com/ubco-mds-2022/Data-550

Weather data (first 10)

```
1 df.head(10)
```

	location	date	precipitation	temp_max	temp_min	wind	weather
0	Seattle	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	Seattle	2012-01-02	10.9	10.6	2.8	4.5	rain
2	Seattle	2012-01-03	0.8	11.7	7.2	2.3	rain
3	Seattle	2012-01-04	20.3	12.2	5.6	4.7	rain
4	Seattle	2012-01-05	1.3	8.9	2.8	6.1	rain
5	Seattle	2012-01-06	2.5	4.4	2.2	2.2	rain
6	Seattle	2012-01-07	0.0	7.2	2.8	2.3	rain
7	Seattle	2012-01-08	0.0	10.0	2.8	2.0	sun
8	Seattle	2012-01-09	4.3	9.4	5.0	3.4	rain
9	Seattle	2012-01-10	1.0	6.1	0.6	3.4	rain

https://github.com/ubco-mds-2022/Data-550

Weather data (last 10)

```
1 df.tail(10)
```

	location	date	precipitation	temp_max	temp_min	wind	weather
2912	New York	2015-12-22	4.8	15.6	11.1	3.8	fog
2913	New York	2015-12-23	29.5	17.2	8.9	4.5	fog
2914	New York	2015-12-24	0.5	20.6	13.9	4.9	fog
2915	New York	2015-12-25	2.5	17.8	11.1	0.9	fog
2916	New York	2015-12-26	0.3	15.6	9.4	4.8	drizzle
2917	New York	2015-12-27	2.0	17.2	8.9	5.5	fog
2918	New York	2015-12-28	1.3	8.9	1.7	6.3	snow
2919	New York	2015-12-29	16.8	9.4	1.1	5.3	fog
2920	New York	2015-12-30	9.4	10.6	5.0	3.0	fog
2921	New York	2015-12-31	1.5	11.1	6.1	5.5	fog

https://github.com/ubco-mds-2022/Data-550

Advanced Plotting Options

https://github.com/ubco-mds-2022/Data-550

Precipitation Comparison

Seattle has a reputation as a rainy city. Is that deserved?

```
precip = alt.Chart(df).mark_line(
).encode(
  x = 'date',
  y = 'precipitation',
  color = 'location')
```

https://github.com/ubco-mds-2022/Data-550

Time Units

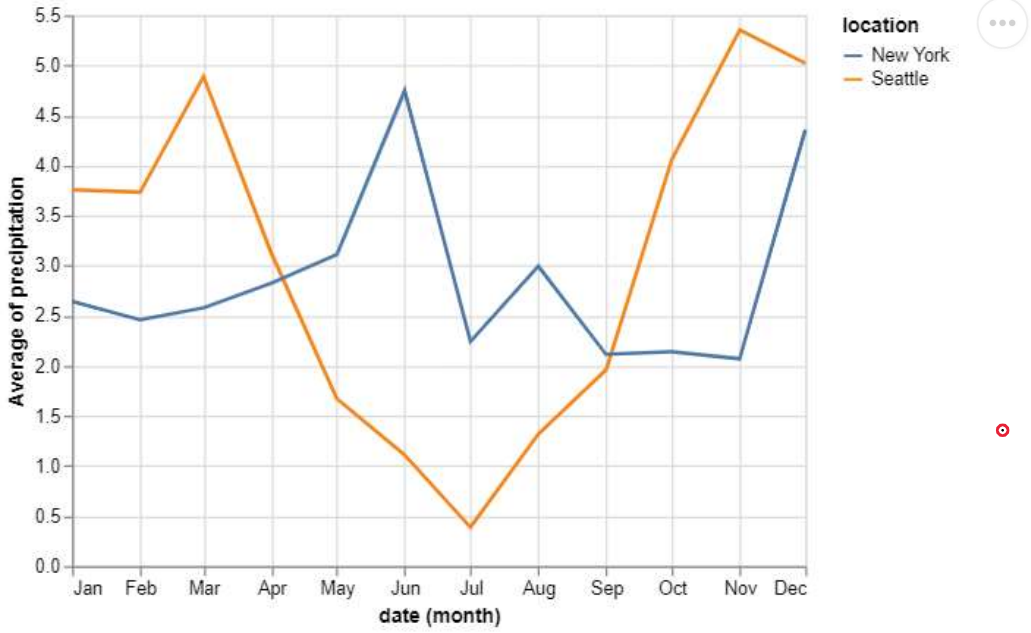
- To make this chart, we make use of the timeUnit property defined in the `date` field.
- Similar to binning, we use a timeUnit transform to map record dates to the month of the year.
- Valid time units include: year, quarter, date (numeric day in month), day (day of the week), hours, … and more
- For example, the chart on the next slide shows average temperature in Seattle and New York aggregated by month.

https://github.com/ubco-mds-2022/Data-550

Precipitation by Month

```
precip = alt.Chart(df).mark_line(
).encode(
  x = 'month(date)',              # will bin obs by month
  y = 'average(precipitation)', # will avg prec within in month
  color = 'location')
```

```
precip
```



We will build towards creating multi-view displays to examine precipitation *and* weather within and across the cities.

https://github.com/ubco-mds-2022/Data-550

Area marks for Ranges

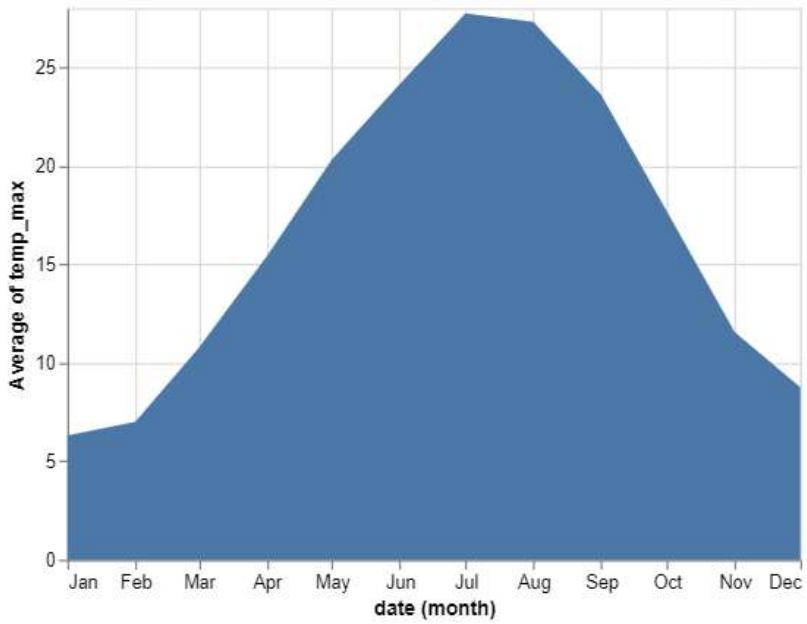
- For temperature, let's seek to visualize the *range* of minimum and maximum temperatures per month
- Last lecture we create how to area plots via `mark_area`
- The `area` mark type (as well as the `bar` mark type) allows us to use the `y` and `y2` channels (or `x` and `x2`) to provide end points for the area mark.
- So, rather than using a zero-baseline and filling the region from 0 to `y`, we will instead fill the region from `y2` to `y`.

https://github.com/ubco-mds-2022/Data-550

Area plots with zero-baseline

```
base = alt.Chart(df)

base.mark_area().encode(
    x = alt.X('month(date)'),
    y = alt.Y('average(temp_max)'),
)
```

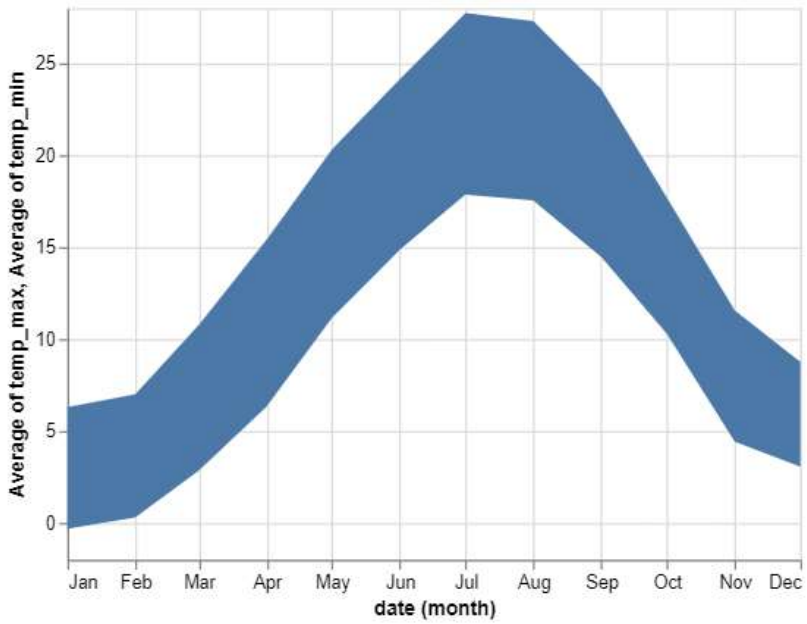


https://github.com/ubco-mds-2022/Data-550

Area plots with non-zero-baseline

```
base = alt.Chart(df)

base.mark_area().encode(
    x = alt.X('month(date)'),
    y = alt.Y('average(temp_max)'),
    y2= alt.Y2('average(temp_min)')
)
```



This is pretty misleading as it aggregates the measurements for both Seattle and New York, …

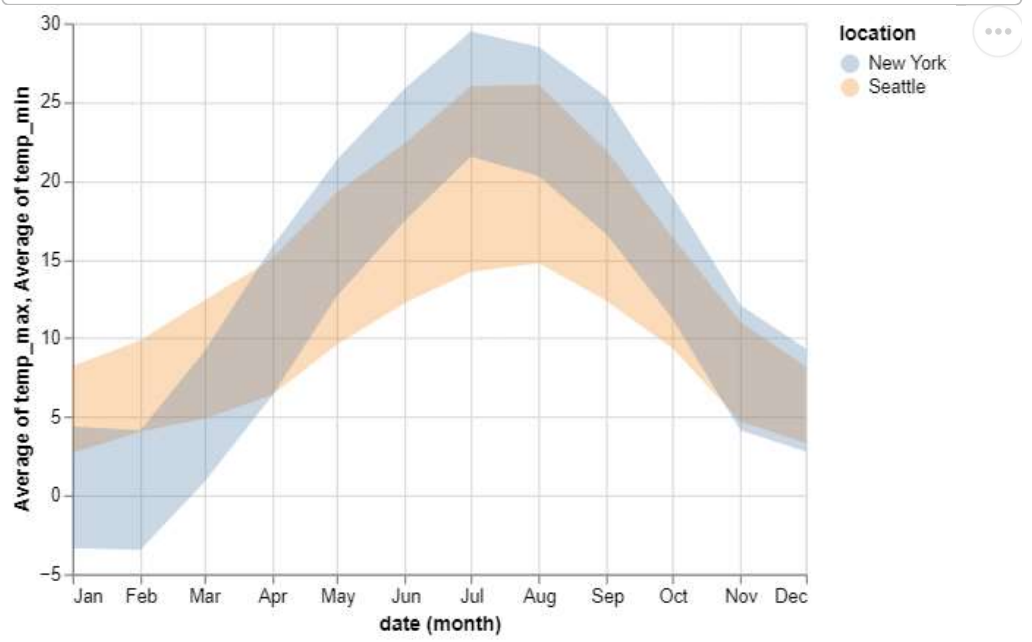
https://github.com/ubco-mds-2022/Data-550

Shared axes

To subdivide the data by location we can use the color encoding and adjust the mark opacity to accommodate overlapping areas:

```
temprange = base.mark_area(
  opacity=0.3
).encode(
  x = 'month(date)',
  y = 'average(temp_max)',
  y2 = 'average(temp_min)',
  color = 'location'
)
```

```
temprange
```



Let's add the average midpoint temperature …

https://github.com/ubco-mds-2022/Data-550

Data Transformations

- Since the midpoint temperature is not an existing feature in the data, we'll have to create it using *data transformations*.
- Altair allows us to specify data transformations within the chart specification itself via `transform_*` methods.
- We'll use a `calculate` transform to compute the midpoints between the min/max daily temperatures
- We will need to use the `datum.` prefix to access a field value on the input record.

A complete list of data transformation methods can be found here. See also Chapter 3: https://github.com/ubco-mds-2022/Data-550

16

`transform_*` methods

Partial table from Altair documentation: Data Transformation

Transform	Method	Description
Aggregate Transforms	`transform_aggregate()`	Create a new data column by aggregating an existing column.
Bin transforms	`transform_bin()`	Create a new data column by binning an existing column.
Calculate Transform	`transform_calculate()`	Create a new data column using an arithmetic calculation on an existing column.
Density Transform	`transform_density()`	Create a new data column with the kernel density estimate of the input.
Filter Transform	`transform_filter()`	Select a subset of data based on a condition.

...

Jump to filtering example

https://github.com/ubco-mds-2022/Data-550

Calculating midpoints

The midpoints between the minimum daily temperature and maximum daily temperature is calculated as:

$$\text{mid temp} = \frac{\text{min temp} + \text{max temp}}{2}$$

```
tempmid = alt.Chart(df).mark_line().transform_calculate(
  temp_mid='(datum.temp_min + datum.temp_max) / 2'
)
```

https://github.com/ubco-mds-2022/Data-550

Calculating midpoints

The midpoints between the minimum daily temperature and maximum daily temperature is calculated as:

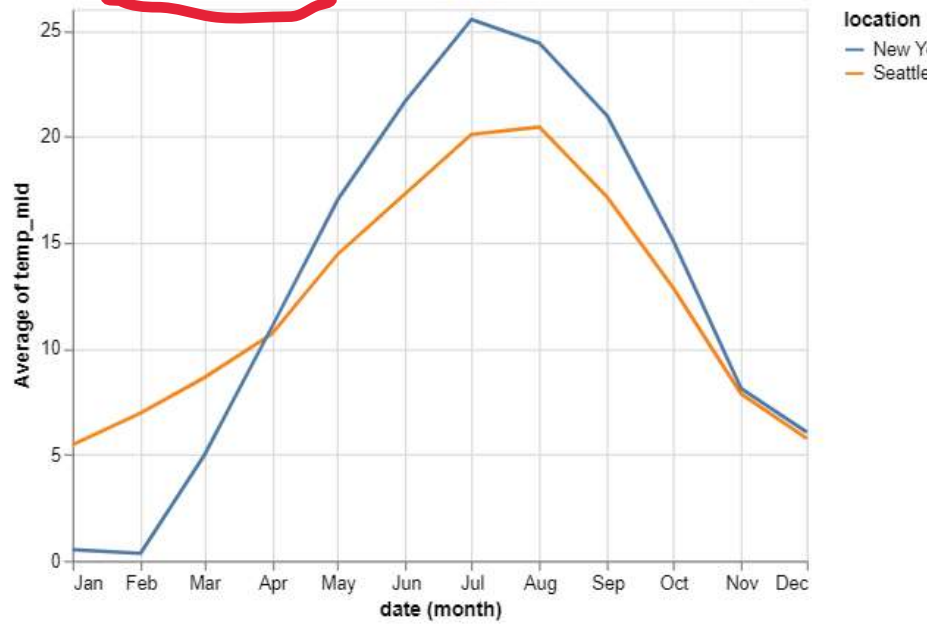
$$\text{mid temp} = \frac{\text{min temp} + \text{max temp}}{2}$$

```
tempmid = alt.Chart(df).mark_line().transform_calculate(
  temp_mid='(datum.temp_min + datum.temp_max) / 2'
).encode(
  alt.X('month(date)'),
  alt.Y('average(temp_mid):Q'), # recall :Q indicates quantitative data typ
  alt.Color('location')         # same as color = 'location'
)
```

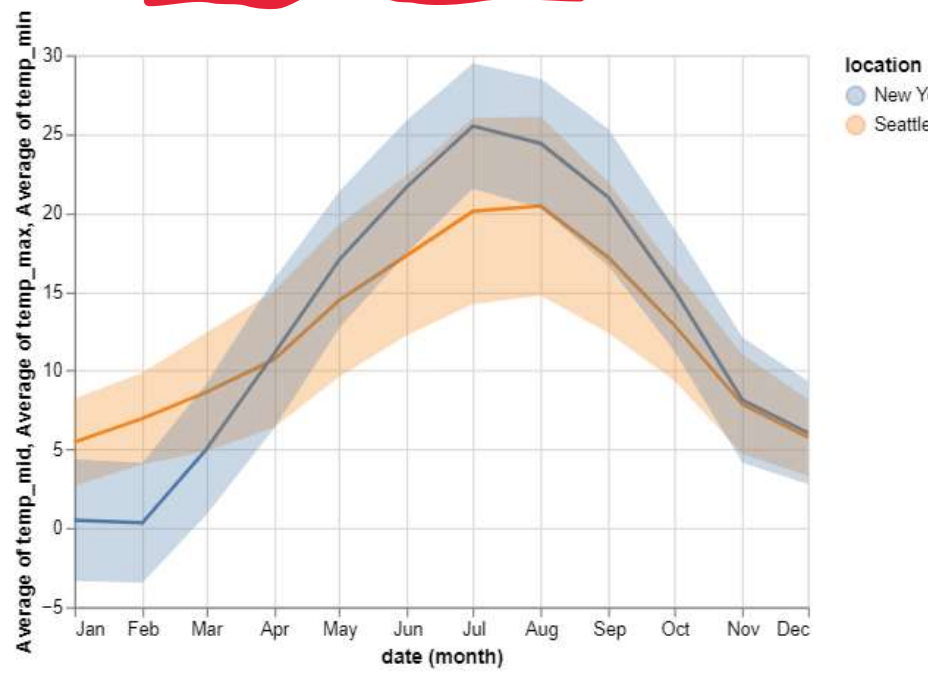
https://github.com/ubco-mds-2022/Data-550

Multi-layered plot

```
1 tempmid
```




```
1 tempmid + temprange
```



We now have a multi-layered plot

https://github.com/ubco-mds-2022/Data-550

Multi-View Composition

https://github.com/ubco-mds-2022/Data-550

Layers

- We've seen how to created layered plots by:
  - mapping nominal features to the `colour`/`size` encoding channel,
  - using the `+` operator
- Alternatively we could use the `alt.layer` function, which accepts as its arguments any number of charts.

https://github.com/ubco-mds-2022/Data-550

alt.layer

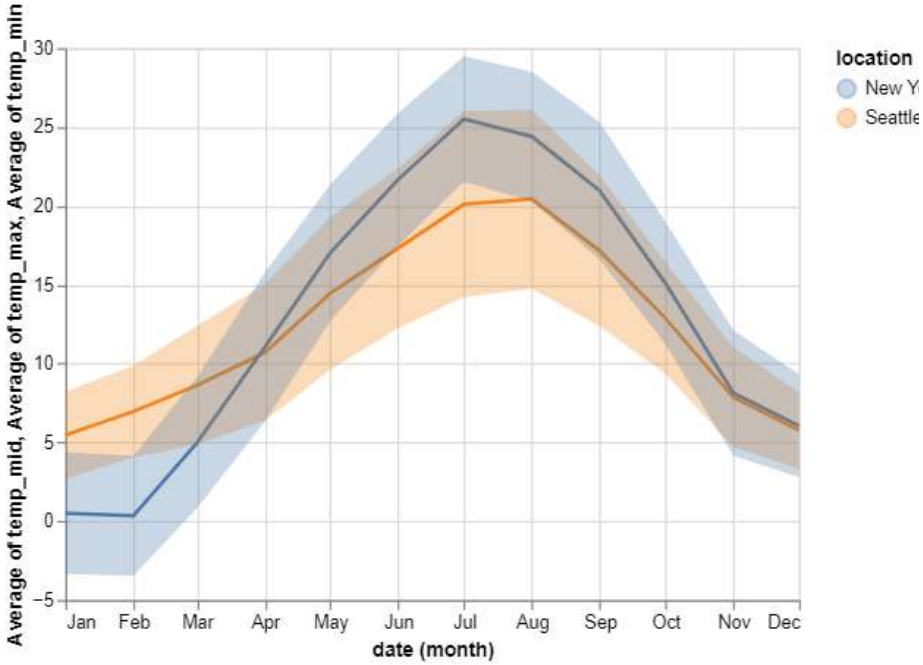
General syntax:

```
1 alt.layer(chart1, chart2, ...)
```

- `chart1` is the first layer
- `chart2` second layer drawn on top
- …

order of layers matter (read more here)

```
1 # same as tempmid + temprange
2 alt.layer(tempmid, temprange)
```



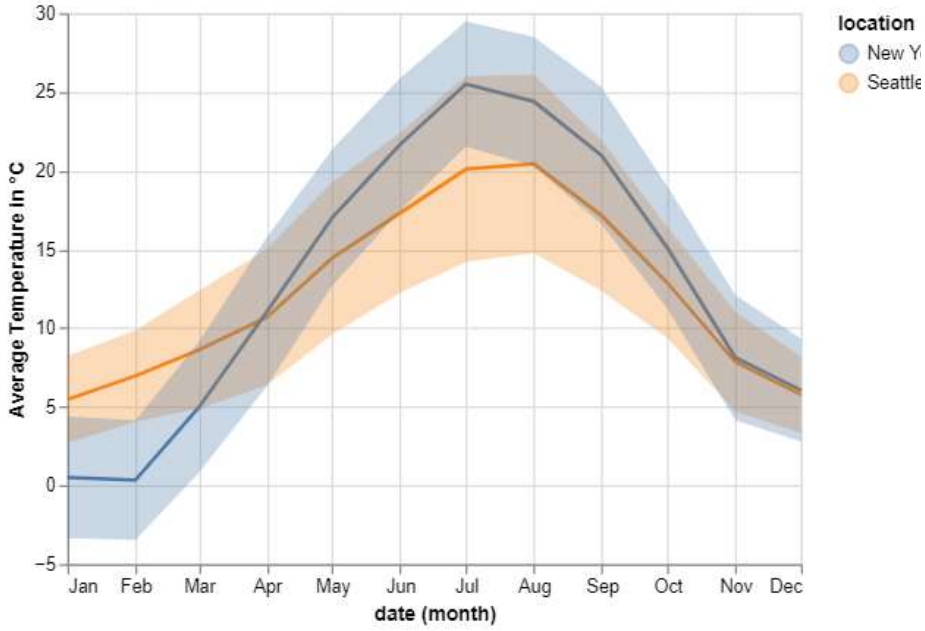
https://github.com/ubco-mds-2022/Data-550

Customize Axes

If we set a custom axis title within one of the layers, it will automatically be used as a shared axis title for all the layers:

```
temprange = base.mark_area(
  opacity=0.3
).encode(
  x = 'month(date)',
  y = alt.Y('average(temp_max)',
  title = "Average Temperature in
  y2 = 'average(temp_min)',
  color = 'location')
```

```
alt.layer(tempmid, temprange)
```



https://github.com/ubco-mds-2022/Data-550

Dual Axes plots

- In all of these layered plots we have been combining *compatible* charts directly on top of each other on the same set of axes.
- If either of the x or y encodings is not compatible, we might instead create a dual-axis chart, which overlays marks using separate scales and axes.
- For examples, let's seek to visualize both the temperature and precipitation on the same chart.
- For illustration, we'll just focus on Seattle.

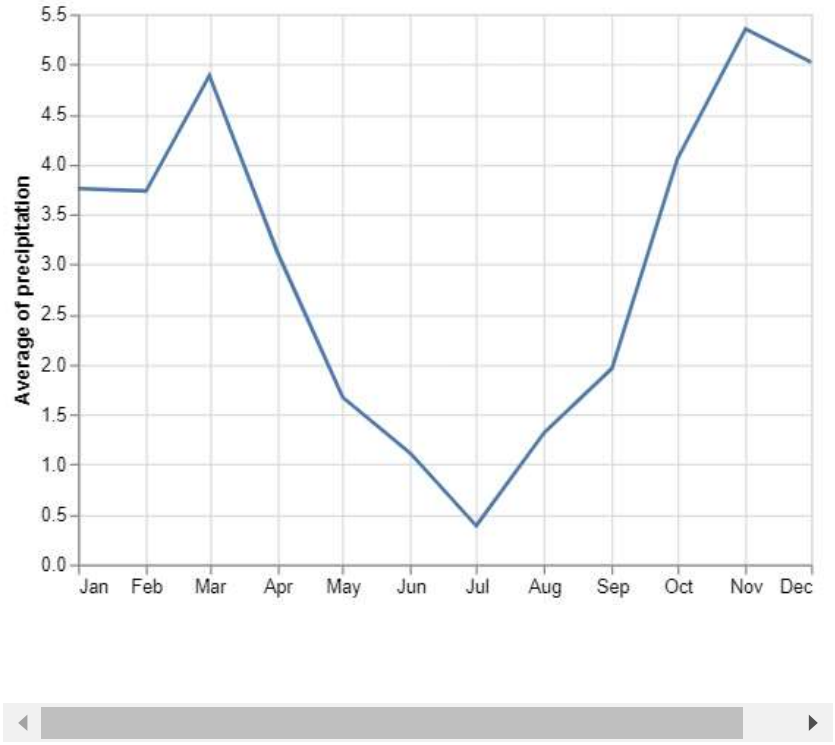
https://github.com/ubco-mds-2022/Data-550

Filter with pandas

Pandas `isin()` method is used to filter data frames.

```
ind =df['location'].isin(['Seattle'])

alt.Chart(df[ind]).mark_line(
).encode(
  alt.X('month(date)', title=None),
  alt.Y('average(precipitation):Q')
)
```

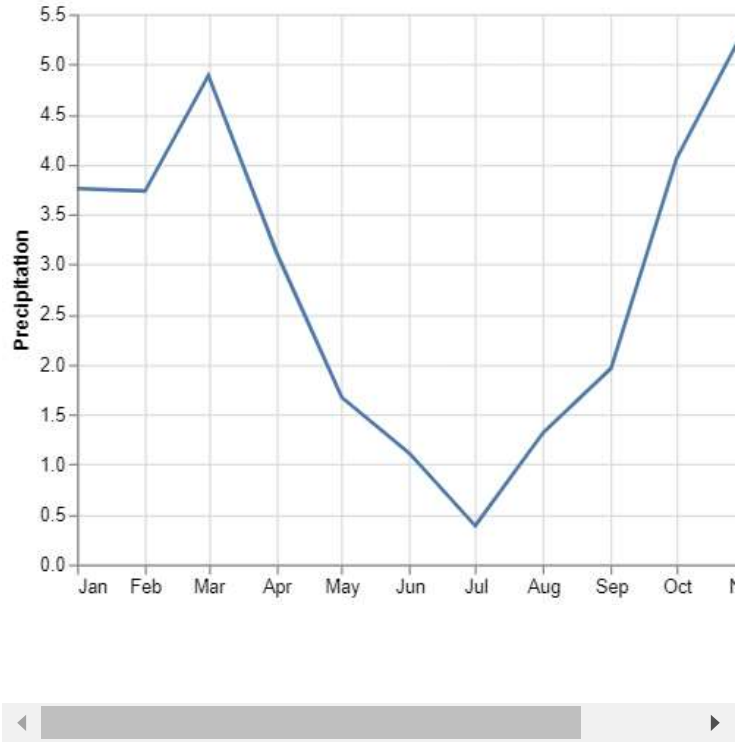


Alternatively, we can do our filtering using a `filter` transform (as mentioned in this slide)

https://github.com/ubco-mds-2022/Data-550

Filtering Example

```
sprecip = alt.Chart(df).transform_filter(
  'datum.location == "Seattle"'
).mark_line(
).encode(
  alt.X('month(date)', title=None),
  alt.Y('average(precipitation):Q',
  title='Precipitation'))
sprecip
```

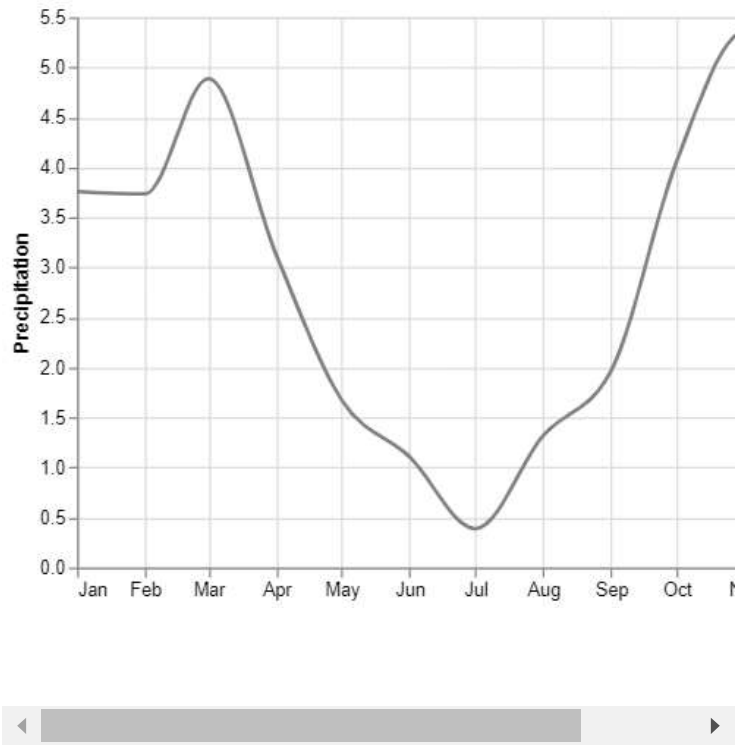


as before, the `datum.` prefix is used to access a field value on the input record.

https://github.com/ubco-mds-2022/Data-550

Seattle Precipitation

```
sprecip = alt.Chart(df).transform_filter(
  'datum.location == "Seattle"'
).mark_line(
  interpolate='monotone', # smooths
  stroke='grey'
).encode(
  alt.X('month(date)', title=None),
  alt.Y('average(precipitation):Q',
  title='Precipitation'))
sprecip
```

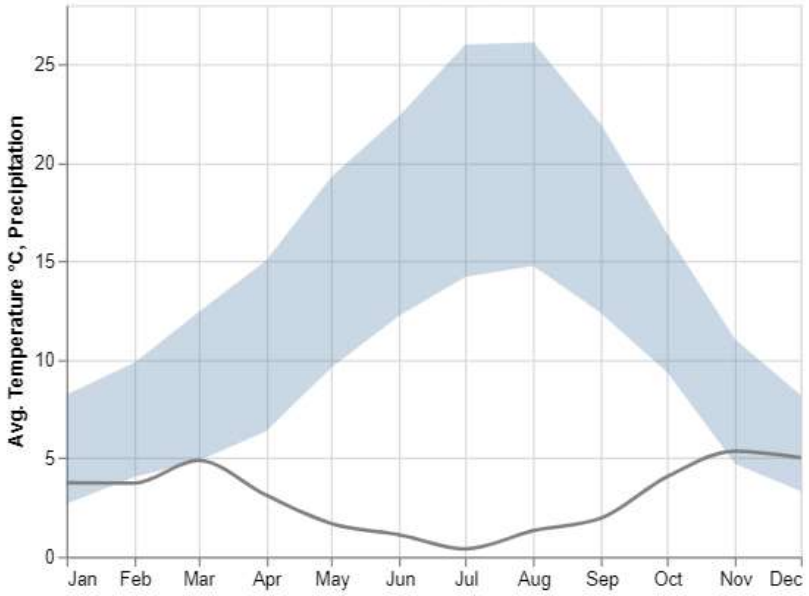


https://github.com/ubco-mds-2022/Data-550

Shared domain

```
# temperature range for only Seattle
srange = alt.Chart(df).transform_filter(
  'datum.location == "Seattle"'
).mark_area(opacity=0.3).encode(
  alt.X('month(date)'),
  alt.Y('average(temp_max)', title='Avg. Temperature °C'),
  alt.Y2('average(temp_min)'))
```

```
alt.layer(srange, sprecip)
```



Precipitation are on different units and have a range of y-values much smaller than that of temperatures.

https://github.com/ubco-mds-2022/Data-550

Layering Seattle Precipitation

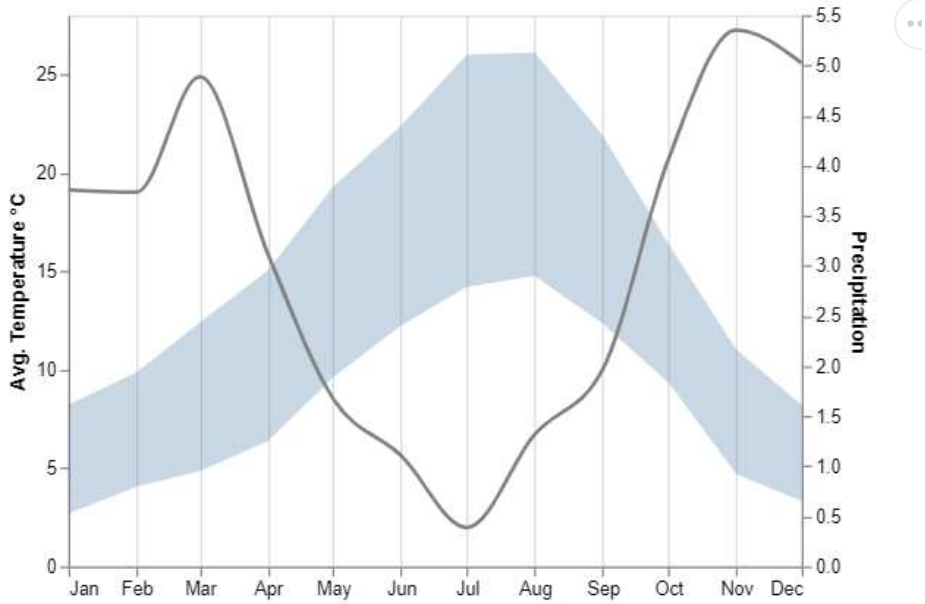
- By default layered charts use a shared domain: the y-values are combined across layers to determine a shared range.
- This default can be adjusted using the `Chart.resolve_scale()`[1] function
- If we want to use different y-axis scales, we need to specify how to *resolve* the data across layers.
- In this case, we want to resolve the y-axis scale domains to be `'independent'` rather than `'shared'`.

https://github.com/ubco-mds-2022/Data-550

1. also known as trellis plots or small multiples

Seattle Dual-axis plot

```
alt.layer(srange, sprecip).resolve_scale(y='independent')
```



Warning dual-axis charts are often prone to misinterpretation

https://github.com/ubco-mds-2022/Data-550

Compound plots

Altair provides a number of compound plot types that can be used to create stacked, layered, faceted, and repeated charts.

Table source altair documentation

method/functional form	operator form	reference
`alt.layer(chart1, chart2)`	`chart1 + chart2`	layered charts
`alt.hconcat(chart1, chart2)`	`chart1 | chart2`	Horizontal Concatenation
`alt.vconcat(chart1, chart2)`	`chart1 & chart2`	Vertical Concatenation
`chart.repeat(row, column)`		Repeated Charts
`chart.facet(facet, row, column)`		Faceted Charts

https://github.com/ubco-mds-2022/Data-550

What are facets

- *Facet plots*[1] are figures made up of multiple subplots
- Faceting produces horizontal and/or vertical suplots by partitioning data into discrete sets and repeating the plot for each set
- These subplots will have the same set of axes, which helps facilitate rapid comparison.
- The term was popularized by Edward Tufte.

https://github.com/ubco-mds-2022/Data-550

1. also known as trellis plots or small multiples

According to Tufte

> At the heart of quantitative reasoning is a single question: *Compared to what?* Small multiple designs, multivariate and data bountiful, answer directly by visually enforcing comparisons of changes, of the differences among objects, of the scope of alternatives. For a wide range of problems in data presentation, small multiples are the best design solution.

Tufte, Edward (1990). *Envisioning Information*. Graphics Press. p. 67. ISBN 978-0961392116.

Global Development Data

- As a motivating example let's return to our global health and population data from last lecture; (gm_url here)

```
import pandas as pd
import altair as alt

gm = pd.read_csv(gm_url, parse_dates=['year'])
gm2018 = gm[gm['year'] == '2018']
```

- Notice we have filtered again by the year 2018 for feasibility

https://github.com/ubco-mds-2022/Data-550

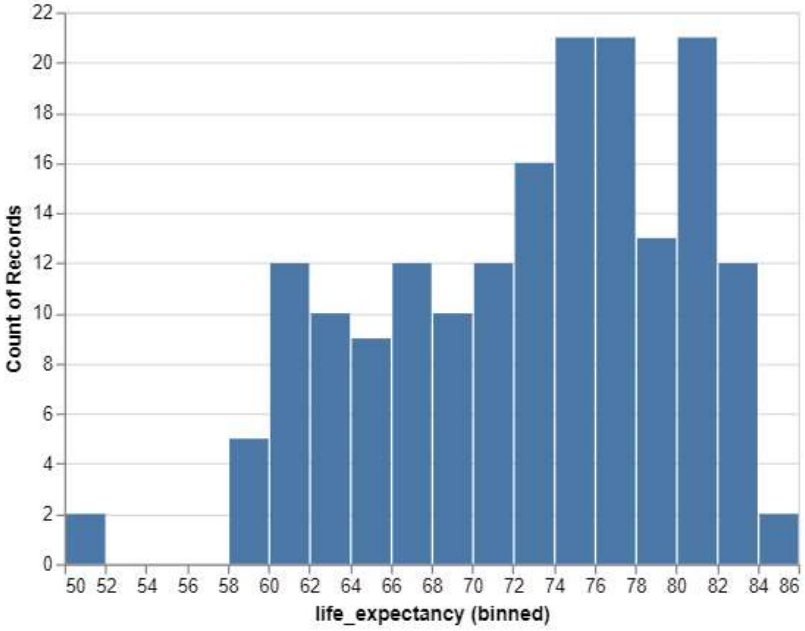
Column	Description
country	Country name
year	Year of observation
population	Population in the country at each year
region	Continent the country belongs to
sub_region	Sub-region the country belongs to
income_group	Income group as specified by the world bank in 2018
life_expectancy	The mean number of years a newborn would live if mortality patterns remained constant
income	GDP per capita (in USD) adjusted for differences in purchasing power
children_per_woman	Average number of children born per woman
child_mortality	Deaths of children under 5 years of age per 1000 live births
pop_density	Average number of people per km2
co2_per_capita	CO2 emissions from fossil fuels (tonnes per capita)
years_in_school_men	Mean number of years in primary, secondary, and tertiary school for 25-36 years old men
years_in_school_women	Mean number of years in primary, secondary, and tertiary school for 25-36 years old women

https://github.com/ubco-mds-2022/Data-550

Histogram

What is the distribution of life expectancy within our data set?

```
alt.Chart(gm2018).mark_bar().encode(
    alt.X('life_expectancy', bin=alt.Bin(maxbins=30),
    alt.Y('count()'))
```



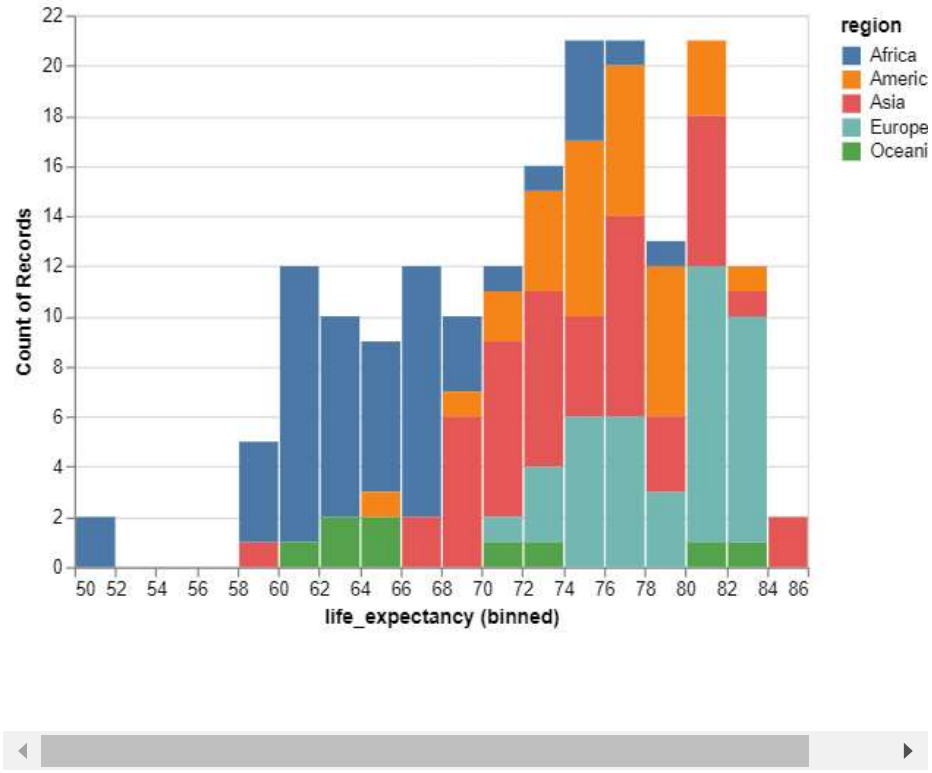
This class: *How does life expectancy differ between regions?*

https://github.com/ubco-mds-2022/Data-550

Stacked Histogram

If we map `region` to color, Altair creates a *stacked bar chart*.

```
chart = alt.Chart(gm2018)
chart.mark_bar().encode(
    alt.X('life_expectancy',
    bin=alt.Bin(maxbins=30)),
    alt.Y('count()'),
    color = 'region')
```



https://github.com/ubco-mds-2022/Data-550

Comments about stacking

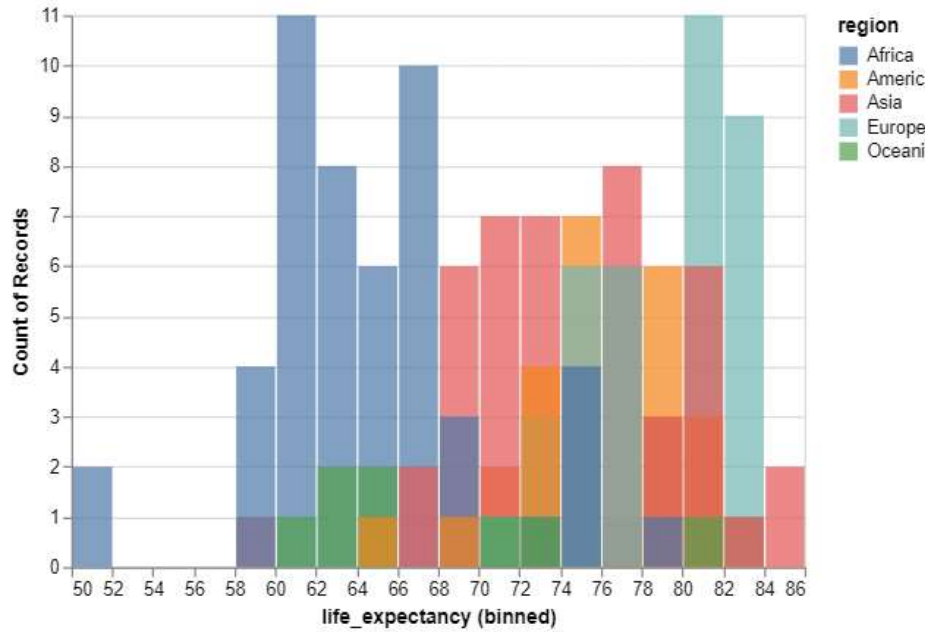
- Just like with the stacked area chart, stacked bar charts are good when the *total* height of each bar is the most important
- Since they do not necessarily share the same baseline, comparison between coloured segments (both within a bar and between bars) are difficult to do.
- Stacked histograms are there not an effective visualization, when the main focus of our visualization is to compare the coloured groups against each other.

https://github.com/ubco-mds-2022/Data-550

Layered Histograms

If we tell Altair not to stack along the y-axis, it will instead layer the histograms behind each other. To be able to see all groups, we need to add some opacity to the bar mark.

```
chart.mark_bar(opacity=0.7).encode
    alt.X('life_expectancy',
    bin=alt.Bin(maxbins=30)),
    alt.Y('count()',stack=False),
    alt.Color('region'))
    # ^^ same as
    # color = 'region'
```



https://github.com/ubco-mds-2022/Data-550

Comments about layering

- Although the bars share the same baseline here, they are still difficult to compare against each other, because there is so much overlap with different colours.
- Layered histograms and bar charts can be effective when there are few groups and clear separation between them
- As that is not the case here, this plot is even harder to interpret than the previous one.

https://github.com/ubco-mds-2022/Data-550

Facets

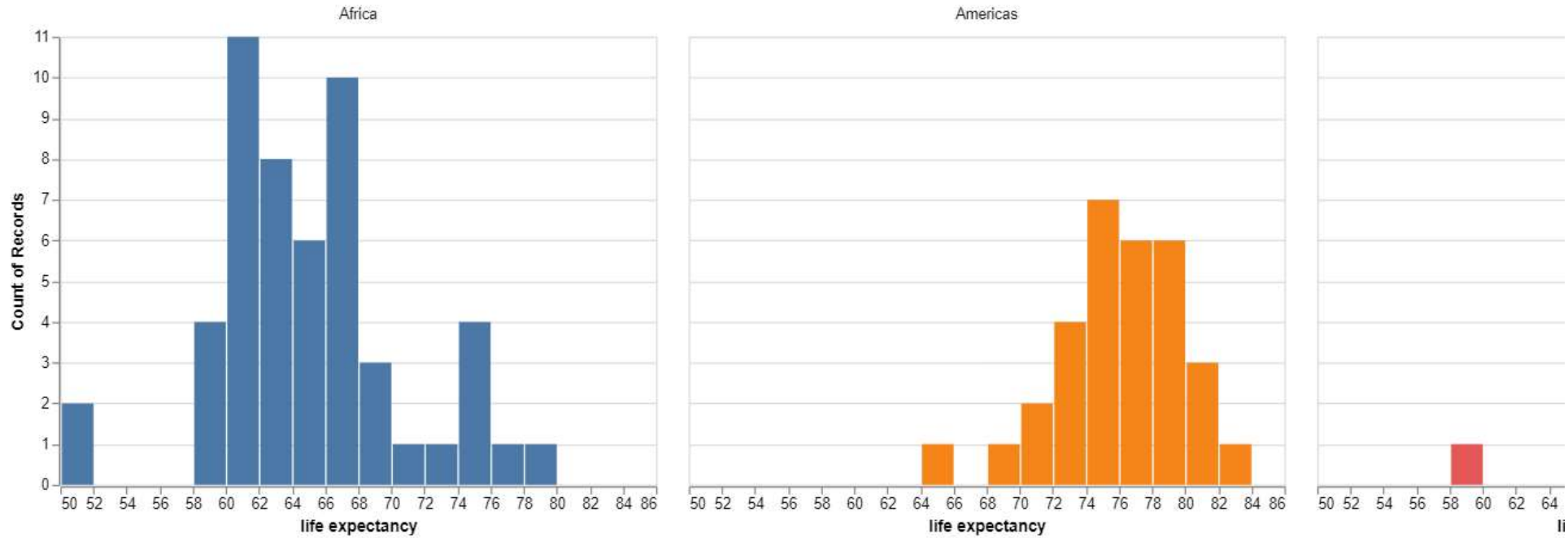
- So rather than stacking or layering the information for each region on a single plot, we will instead create a separate sub-plot[1] for each region
- This is accomplished using the `.facet()` method.
- Faceting creates one facet/subplot for each group in the specified dataframe column, i.e. feature.
- Let's facet the data by `region` ...

https://github.com/ubco-mds-2022/Data-550

1. aka small multiple or trellis plot

Horizontal layout

```
1  chart.mark_bar().encode(
2      alt.X('life_expectancy', bin=alt.Bin(maxbins=30), title = "life expecta
3      alt.Y('count()'),
4      alt.Color('region')
5  ).facet('region')
```

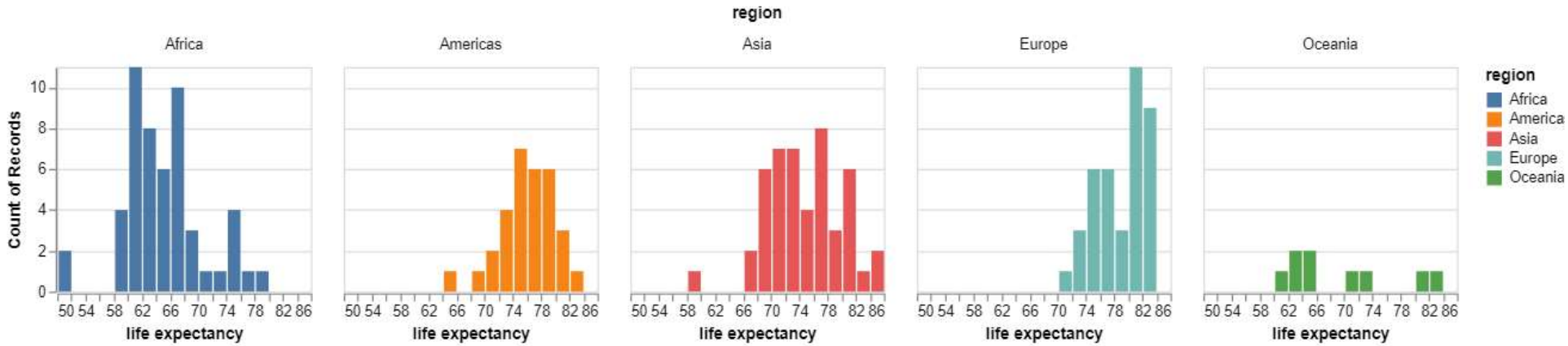


https://github.com/ubco-mds-2022/Data-550

Properties

We can configure facet headers[1] including the font, color, size, and position of the title and labels. Here is an example:

```
chart.mark_bar().encode(
    alt.X('life_expectancy', bin=alt.Bin(maxbins=30), title="life expectanc
    alt.Y('count()'),
    alt.Color('region')
).properties(width=170, height=150
).facet('region')
```



https://github.com/ubco-mds-2022/Data-550

1. we used this methods for chart configuration of main titles last lecture

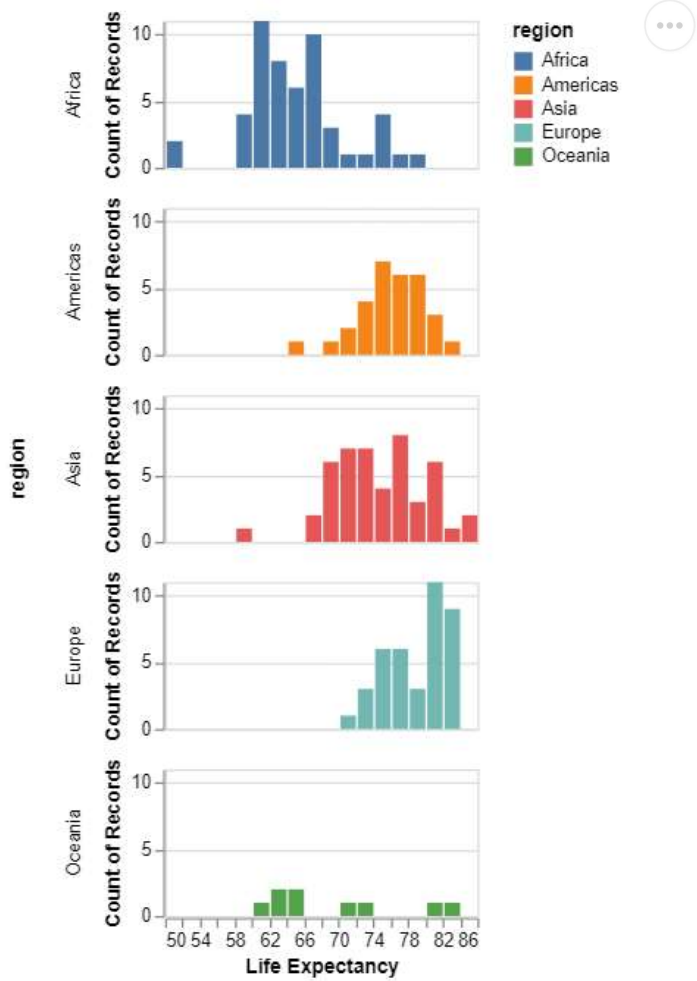
Facet rows and columns

- The `.facet()` method accepts both `row` and `column` arguments; see complete documentation here
- By default, facet will use the `column` encoding channel to facet the data by feature (in this case, `region`). That is to say, the plots will be layed horizontally.
- To make it easier to compare along on the x-axis, we can lay out the facets vertically in a single *column*
- The two can be used together to create a 2D grid of faceted plots.

https://github.com/ubco-mds-2022/Data-550

Vertical layout

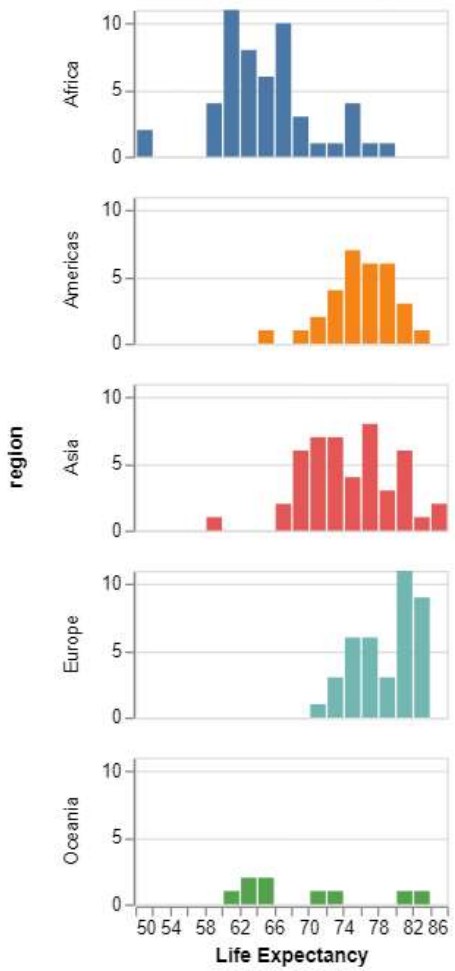
```
chart.mark_bar().encode(
    alt.X('life_expectancy',
    bin=alt.Bin(maxbins=30),
    title = "Life Expectancy"),
    alt.Y('count()'),
    alt.Color('region')
).properties(width=170, height=80
).facet(row = 'region')
```



https://github.com/ubco-mds-2022/Data-550

Vertical layout (cleaned up)

```
chart.mark_bar().encode(
    alt.X('life_expectancy',
    bin=alt.Bin(maxbins=30),
    title = "Life Expectancy"),
    alt.Y('count()', title = None),
    alt.Color('region', legend=None)
).properties(width=170, height=80
).facet(row = 'region')
```



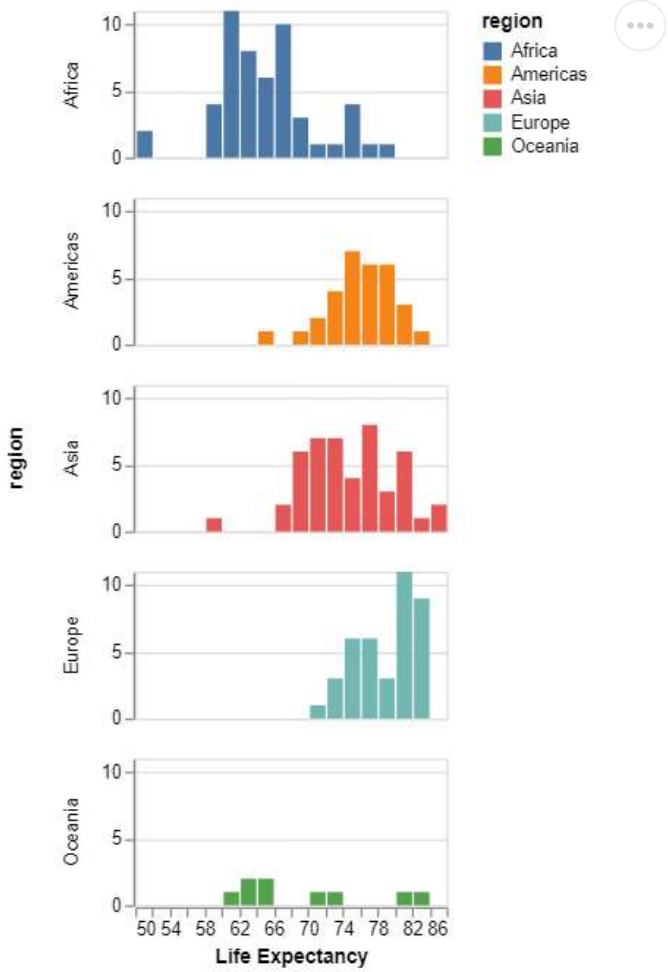
https://github.com/ubco-mds-2022/Data-550

Equivalencies

Facets could alternatively by plotted using `column` (or `row`) encoding channels. For example:

https://github.com/ubco-mds-2022/Data-550

```
chart.mark_bar().encode(
    alt.X('life_expectancy',
    bin=alt.Bin(maxbins=30),
    title = "Life Expectancy"),
    alt.Y('count()', title=None),
    alt.Color('region'),
    row = alt.Row('region')
).properties(width=170, height=80
)
```

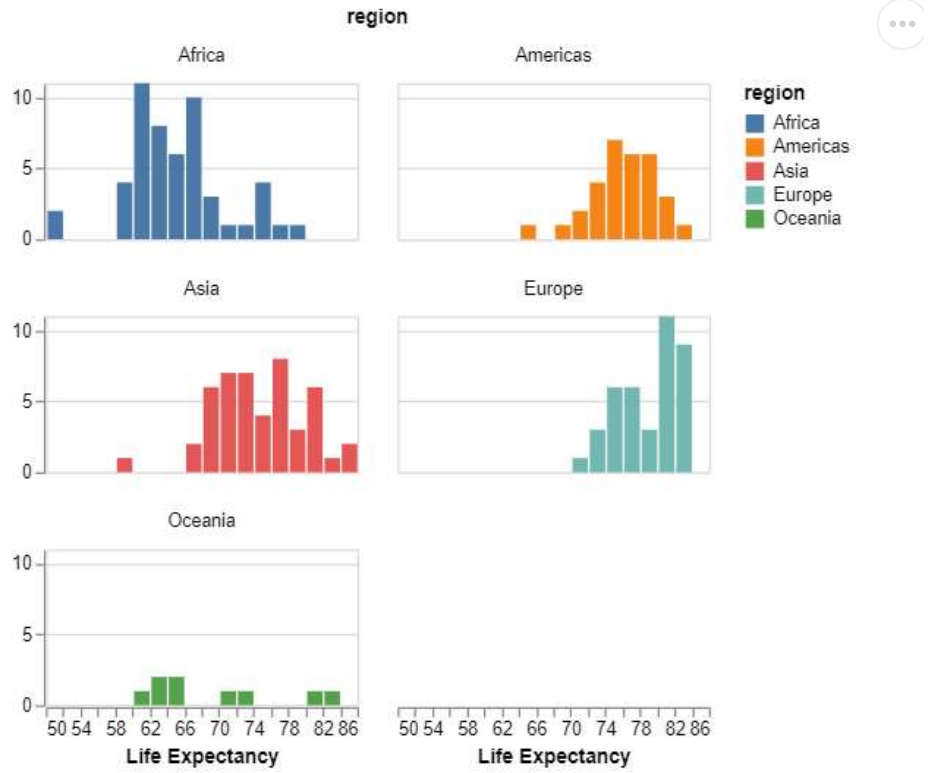


https://github.com/ubco-mds-2022/Data-550

Variations

Alternatively, we could specify how many rows/column for a grid that we then populate with our subpolts

```
chart.mark_bar().encode(
    alt.X('life_expectancy',
    bin=alt.Bin(maxbins=30),
    title = "Life Expectancy"),
    alt.Y('count()', title=None),
    alt.Color('region')
).properties(width=170, height=85
).facet('region', columns=2)
# note the 's' on columns
```



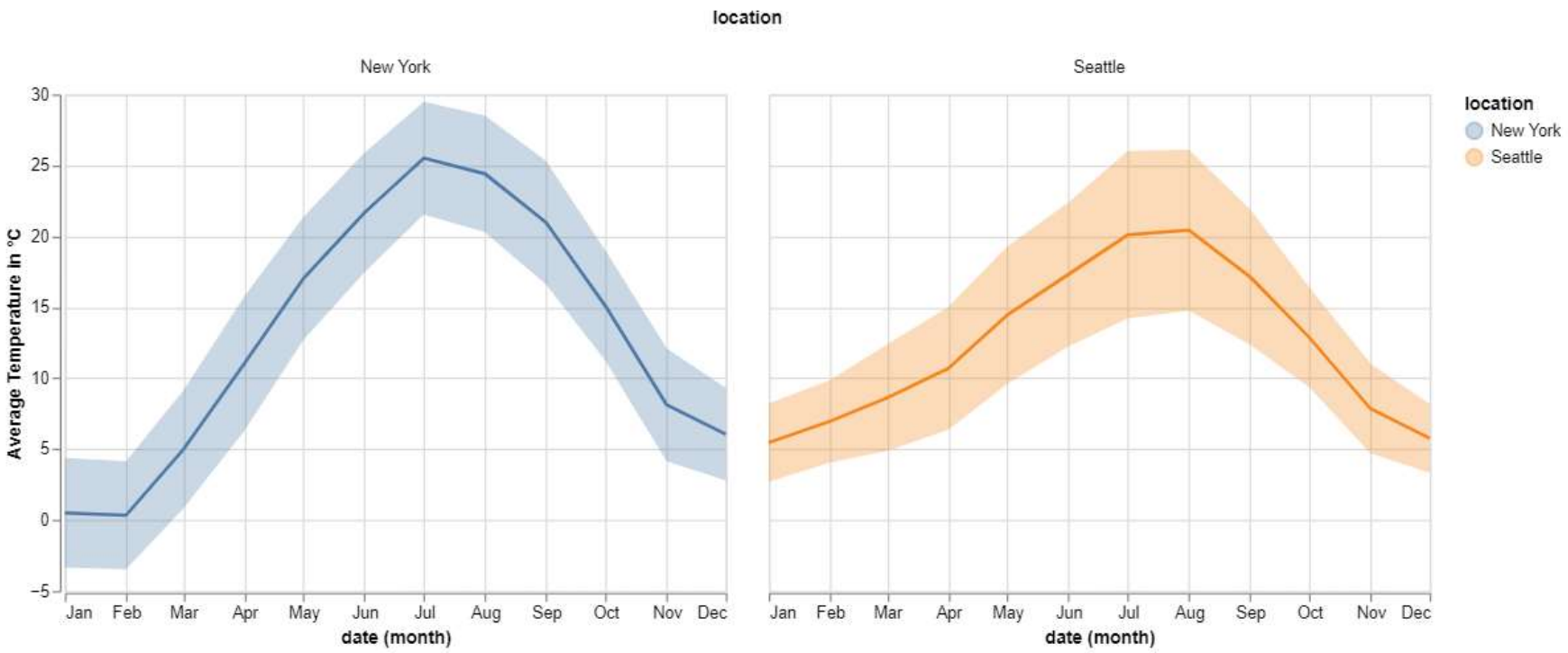
https://github.com/ubco-mds-2022/Data-550

Faceting layered charts

Returning to this composed multi-layered plot, let's see how we can facet on location

```
# same as tempmid + temprange
alt.layer(tempmid, temprange).facet(
  column='location:N'
)
```

https://github.com/ubco-mds-2022/Data-550



https://github.com/ubco-mds-2022/Data-550

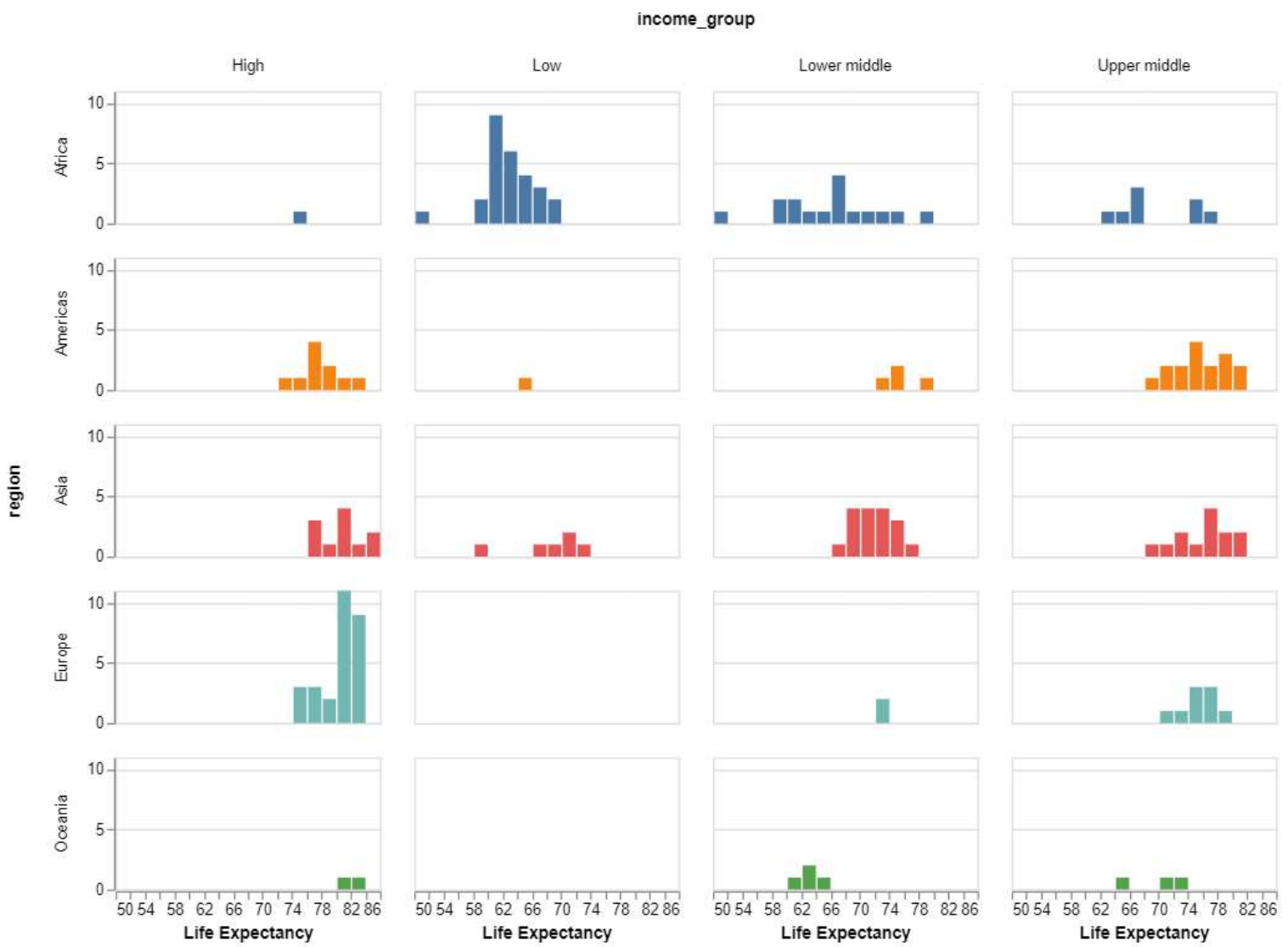
2D grid

```
chart.mark_bar().encode(
    alt.X('life_expectancy', bin=alt.Bin(maxbins=30), title = "Life Expecta
    alt.Y('count()', title=None),
    alt.Color('region', legend=None)
).properties(width=170, height=85
).facet(row = 'region', column = 'income_group')
```

Add this line in .facet() after column argument

sort=['Low', 'Lower middle', 'Upper Middle', 'High']

https://github.com/ubco-mds-2022/Data-550



https://github.com/ubco-mds-2022/Data-550

Concatenate

- Faceting creates a series of similar graphs using the same scale and axes, to show different *partitions* of a dataset.
- However, we might wish to create a multi-view display with different views of the *same* dataset (not subsets).
- Altair provides *concatenation* operators to combine arbitrary charts into a composed chart.
  - `hconcat` (shorthand `|`) performs horizontal concatenation,
  - `vconcat` operator (shorthand `&`) performs vertical concatenation.

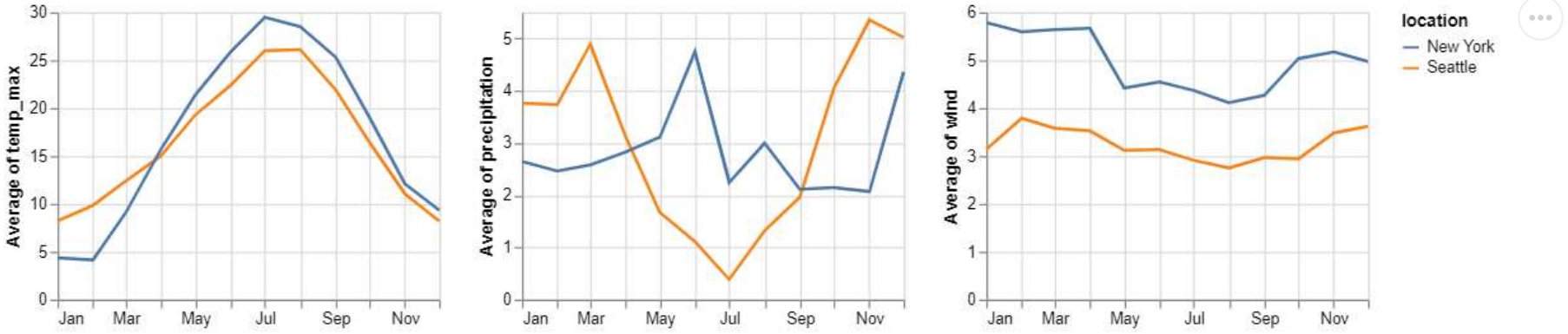
https://github.com/ubco-mds-2022/Data-550

Horizontal concatenate

```
base = alt.Chart(df).mark_line().encode(
  alt.X('month(date):T', title=None),
  color='location:N'
).properties(
  width=240,
  height=180)

temp = base.encode(alt.Y('average(temp_max):Q'))
precip = base.encode(alt.Y('average(precipitation):Q'))
wind = base.encode(alt.Y('average(wind):Q'))

temp | precip | wind ## sames as alt.hconcat(temp, precip, wind)
```



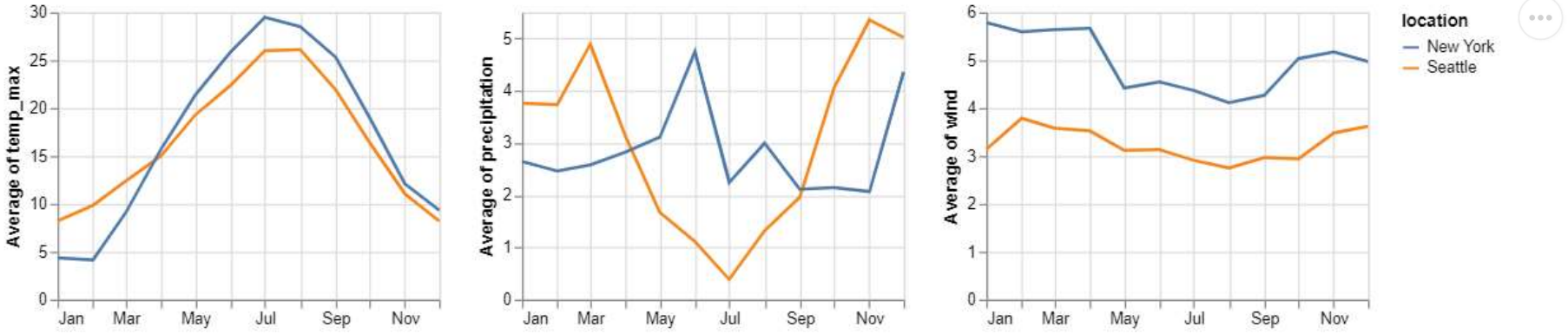
https://github.com/ubco-mds-2022/Data-550

Horizontal concatenate

```
base = alt.Chart(df).mark_line().encode(
  alt.X('month(date):T', title=None),
  color='location:N'
).properties(
  width=240,
  height=180) # base is common over all plots

temp = base.encode(alt.Y('average(temp_max):Q'))
precip = base.encode(alt.Y('average(precipitation):Q'))
wind = base.encode(alt.Y('average(wind):Q'))

temp | precip | wind ## sames as alt.hconcat(temp, precip, wind)
```



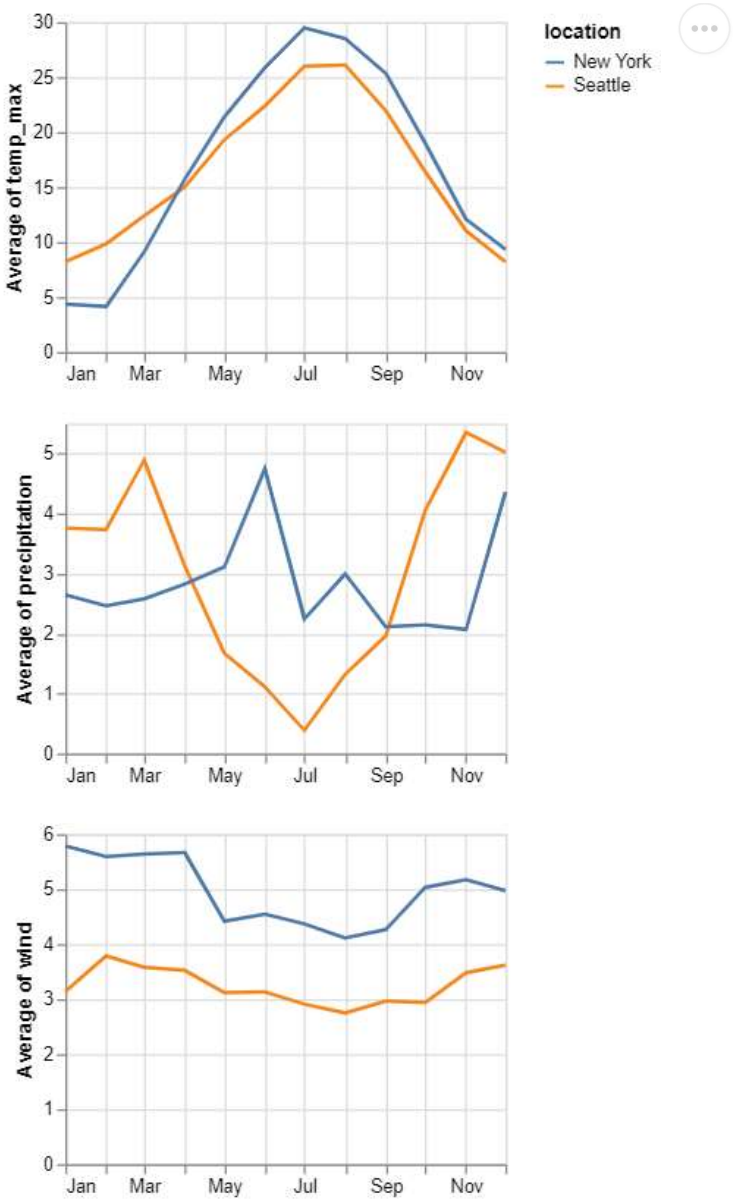
https://github.com/ubco-mds-2022/Data-550

Vertical concatenate

https://github.com/ubco-mds-2022/Data-550

```
temp & precip & wind
## sames as:
## alt.vconcat(temp, precip, wind)
```

- Note: We could use `repeat` to remove some of the repetitiveness in this code
- *for the sake of time I will not be covering this in lecture but you can read about it in Visualization Curriculum*



https://github.com/ubco-mds-2022/Data-550

https://github.com/ubco-mds-2022/Data-550