Data 550: Data Visualization I

Lecture 1: Altair (Python)

Dr. Irene Vrbik

University of British Columbia Okanagan

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

Why Altair?

- Altair is a visualization library for Python that leverages on the most common (and powerful) visualization technologies
- Besides being a declarative tool for producing static data visualizations, it allows us to add interactive components[1]
- It has data aggregation operations (alternatively these can be done with pandas library).
- Installations instructions for Altair and vega_datasets)

The following has been restructured and modified from the University of Washington's Altair course (Copyright (c) 2019, University of Washington)

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

Data

- Data in Altair and ggplot is built around “tidy” (or Pandas dataframe) dataframes
- These consists of a set of named columns corresponding to different data features (aka data fields/variables)
- An *observation* corresponds to a row
- We will start by discussing Altair for which we will often use datasets from the vega_datasets repository.

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

Cars dataset

```
1 import altair as alt
2 from vega_datasets import data
3
4 cars = data.cars()
5 data.cars.description
```

```
'Acceleration, horsepower, fuel efficiency, weight, and other characteristics
of different makes and models of cars. This dataset was originally published
by Donoho et al (1982) [1]_, and was made public at
http://lib.stat.cmu.edu/datasets/'
```

```
1 cars.head(3)
```

	Name	Miles_per_Gallon	Cylinders	Displacement
0	chevrolet chevelle malibu	18.0	8	307.0

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

	Name	Miles_per_Gallon	Cylinders	Displacement
1	buick skylark 320	15.0	8	350.0
2	plymouth satellite	18.0	8	318.0

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

```
1 cars.tail()
```

	Name	Miles_per_Gallon	Cylinders	Displaceme
401	ford mustang gl	27.0	4	140.0
402	vw pickup	44.0	4	97.0
403	dodge rampage	32.0	4	135.0
404	ford ranger	28.0	4	120.0

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

	Name	Miles_per_Gallon	Cylinders	Displaceme
405	chevy s-10	31.0	4	119.0

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

Chart

Create a canvas/chart

MPG (Q)	Origin (N)	Rank (N)	Year (T)
27.0	USA	1	1982-01-01
44.0	Japan	10	1972-01-01
32.0	Japan	221	1970-01-01
28.0	Europe	4	2023-01-01
31.0	USA	53	1980-01-01

```
ggplot(data)
alt.Chart(data)
```

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

6

Chart

- The fundamental object that we'll always be creating in altair is the *chart*
- It will take a data frame as a single argument:

```
1 alt.Chart(cars)
```

- If you run the above, nothing will happen–all we have is a blank canvas
- To start creating plots, we'll need to layer on some more components …

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

Marks

Create a canvas/chart

Add geometric marks

MPG (Q)	Origin (N)	Rank (N)	Year (T)
27.0	USA	1	1982-01-01
44.0	Japan	10	1972-01-01
32.0	Japan	221	1970-01-01
28.0	Europe	4	2023-01-01
31.0	USA	53	1980-01-01

* points
* lines
* Box-plot
* bar

```
ggplot(data, aes(x,y)) + geom_points()
alt.Chart(data).mark_points().encode(x,y)
```

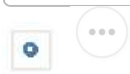
https://github.com/ubco-mds-2022/Data-550

Marks

Now we need to specify how the data should be visualized.

- what kind of graphical *mark* (geometric shape) do you want to represent the data? (set by `Chart.mark_*` methods).

```
1 alt.Chart(cars).mark_point() # create a point for every observation
```



- Here the rendering consists of one point per observation, all plotted on top of each other, since we have not yet specified positions for these points.

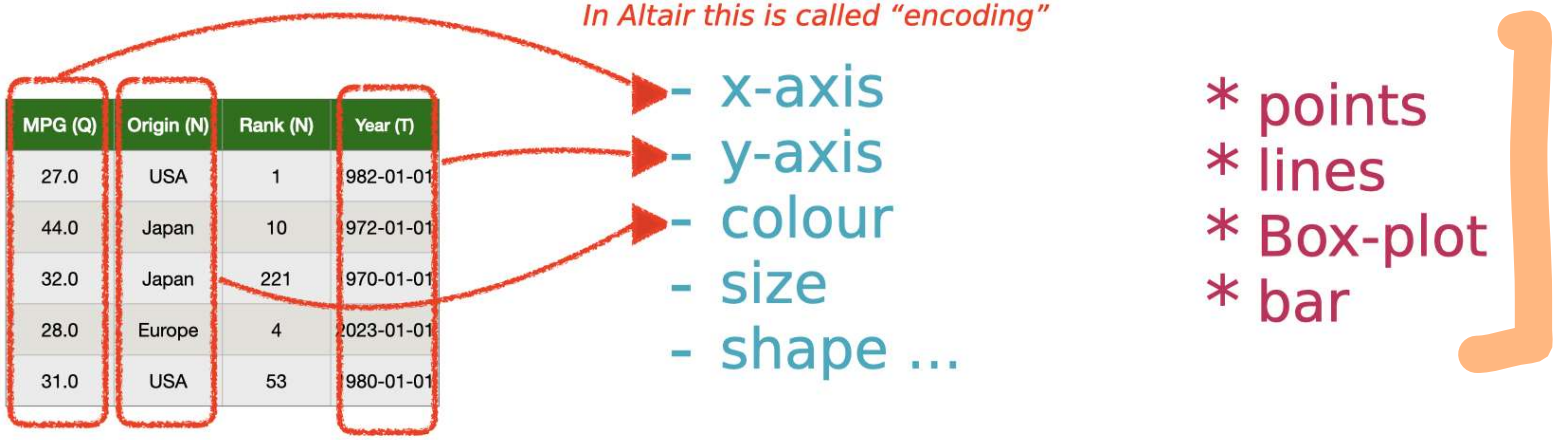
https://github.com/ubco-mds-2022/Data-550

Encodings

Create a canvas/chart

Encode visual aesthetic

Add geometric marks




```
ggplot(data,aes(x,y)) + geom_*()
alt.Chart(data) .mark_*().encode(x,y)
```

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

Encodings

- To visually separate the points, we need to map *properties of the data* to *visual properties* in our plot.
- In Altair, this mapping of visual properties to data columns is referred to as an *encoding*
- The visual properties are referred to as *encoding channels*, or *channels* for short.
- We will go through an example that uses several common channels.

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

Encoding Channels

Position Channels:

- `x`: Horizontal (x-axis) position of the mark.
- `y`: Vertical (y-axis) position of the mark.

Mark Property Channels:

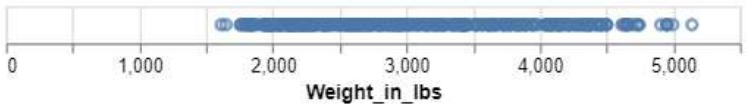
- `size`: Size of the mark.
- `color`: Mark color, specified as a legal CSS color.
- `shape`: Plotting symbol shape for point marks.

see Altair User Guide (Encoding Section) for complete list

encode()

- The `encode()` method builds a key-value mapping between encoding channels (such as x, y, color, shape, size, etc.) to fields in the dataset, accessed by field name.
- For example, to encode the field `Weight_in_lbs` using the x channel, which represents the x-axis position of the points, we use:

```
alt.Chart(cars).mark_point().encode(
  x = 'Weight_in_lbs' # x channel representing the x-axis
)
```



Dot plot or 1-D scatter plot

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

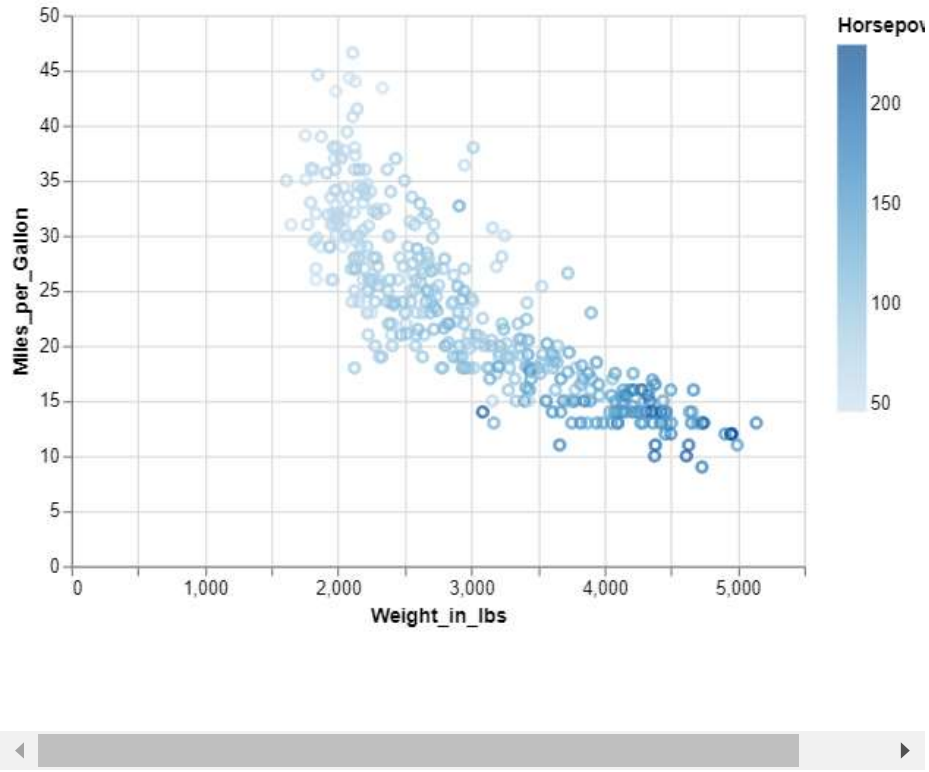
Multiple channels

- Though we've separated the data by one feature, we still have multiple points overlapping.
- Let's further separate these points by adding an y encoding channel for `Miles_per_Gallon`:

Colour channels (continuous feature)

To enrich this display of information further we can ask Altair to colour the points according to the `Horsepower`

```
1 alt.Chart(cars).mark_point().encod
2     x='Weight_in_lbs',
3     y='Miles_per_Gallon',
4     color='Horsepower')
```



Altair will automatically select the colourscale and legend.

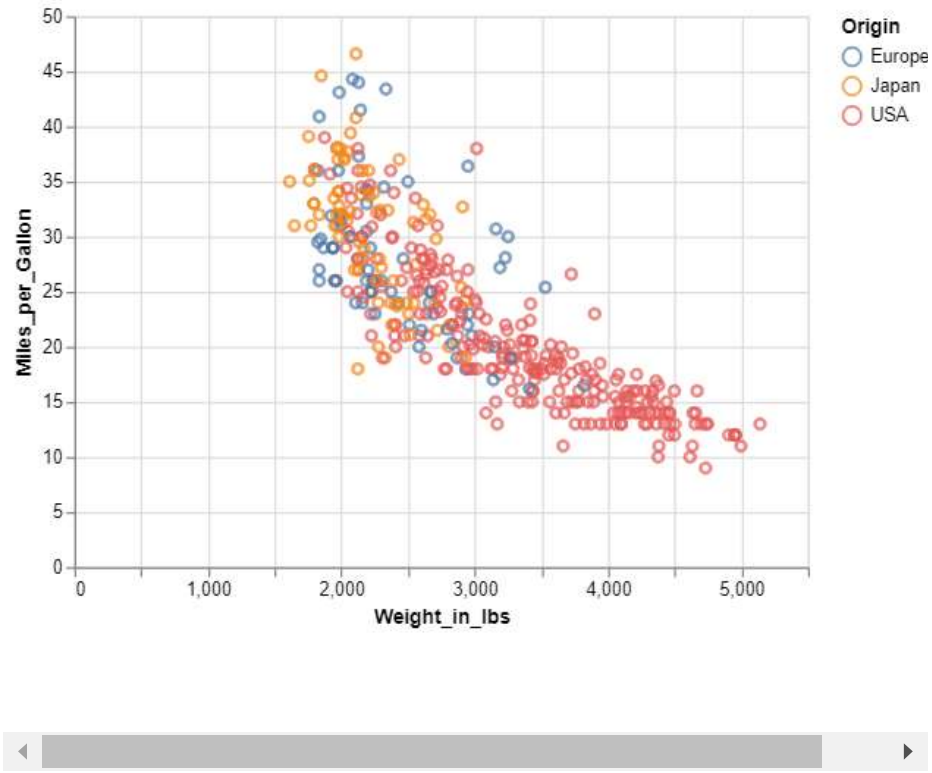
https://github.com/ubco-mds-2022/Data-550

15

Colour channels (discrete feature)

If we instead coloured the points according to a *discrete* feature (as opposed to a *continuous* feature), Altair will adjust the default colouring accordingly. For example,

```
1 alt.Chart(cars).mark_point().encod
2     x='Weight_in_lbs',
3     y='Miles_per_Gallon',
4     color='Origin')
```



jump to Aggregations demo

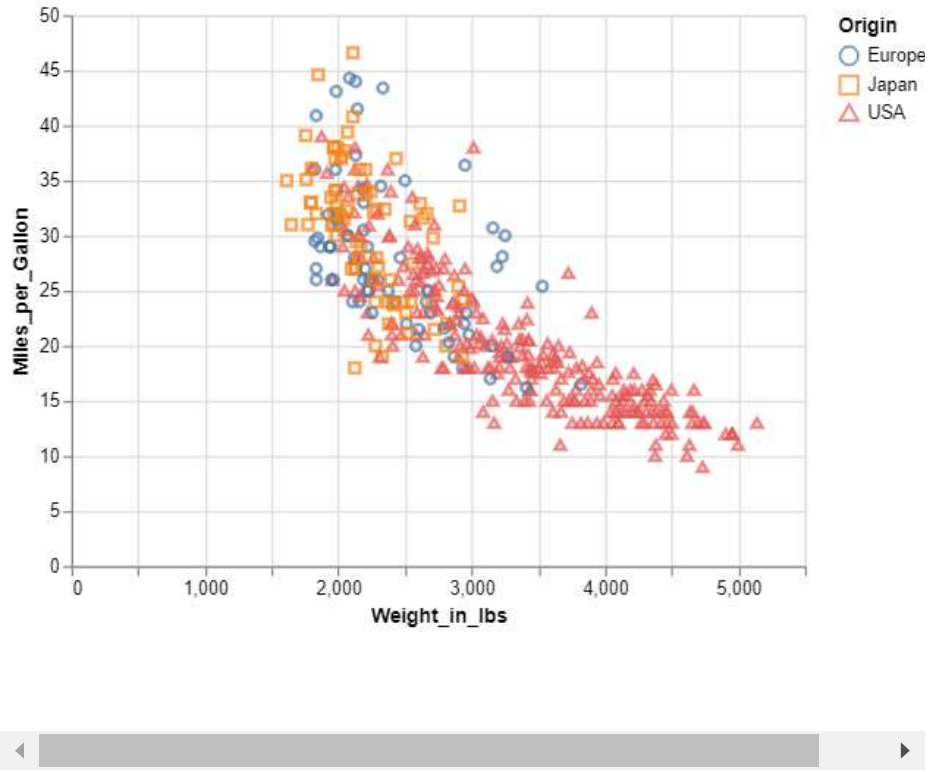
https://github.com/ubco-mds-2022/Data-550

16

Multiple channels

We could have also encoded `Origin` to the shape channel (the syntax is similar)

```
alt.Chart(cars).mark_point().encod
    x='Weight_in_lbs' ,
    y='Miles_per_Gallon',
    color='Origin',
    shape='Origin')

# this may seem like a
# bit of an overkill but
# it could be helpful for
# b&w print, for example
```



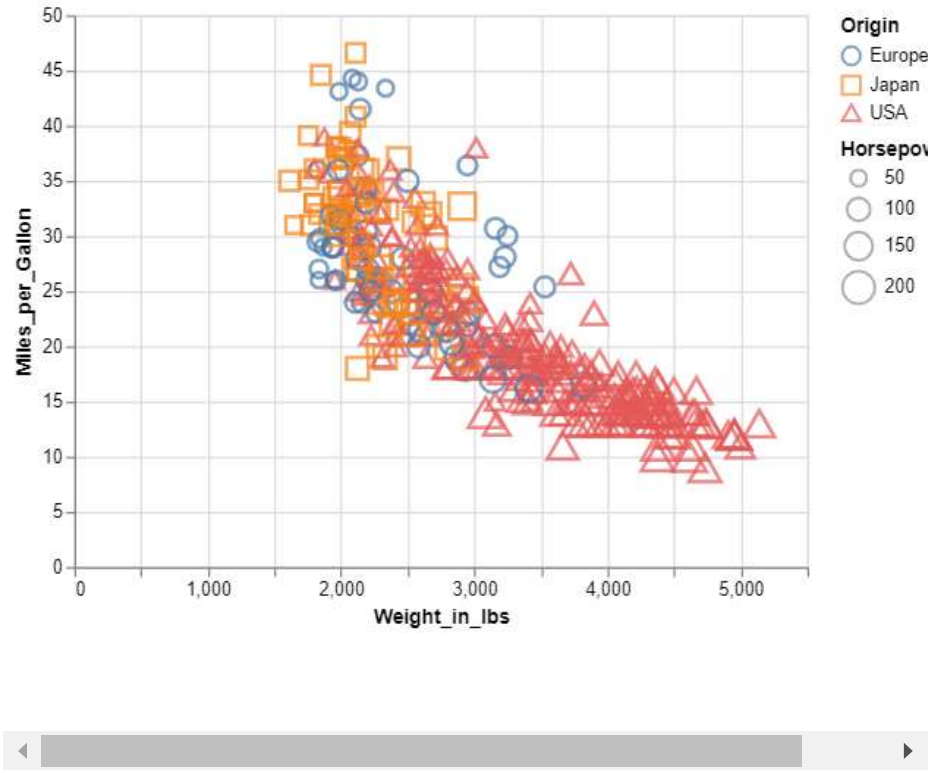
https://github.com/ubco-mds-2022/Data-550

17

Size channel

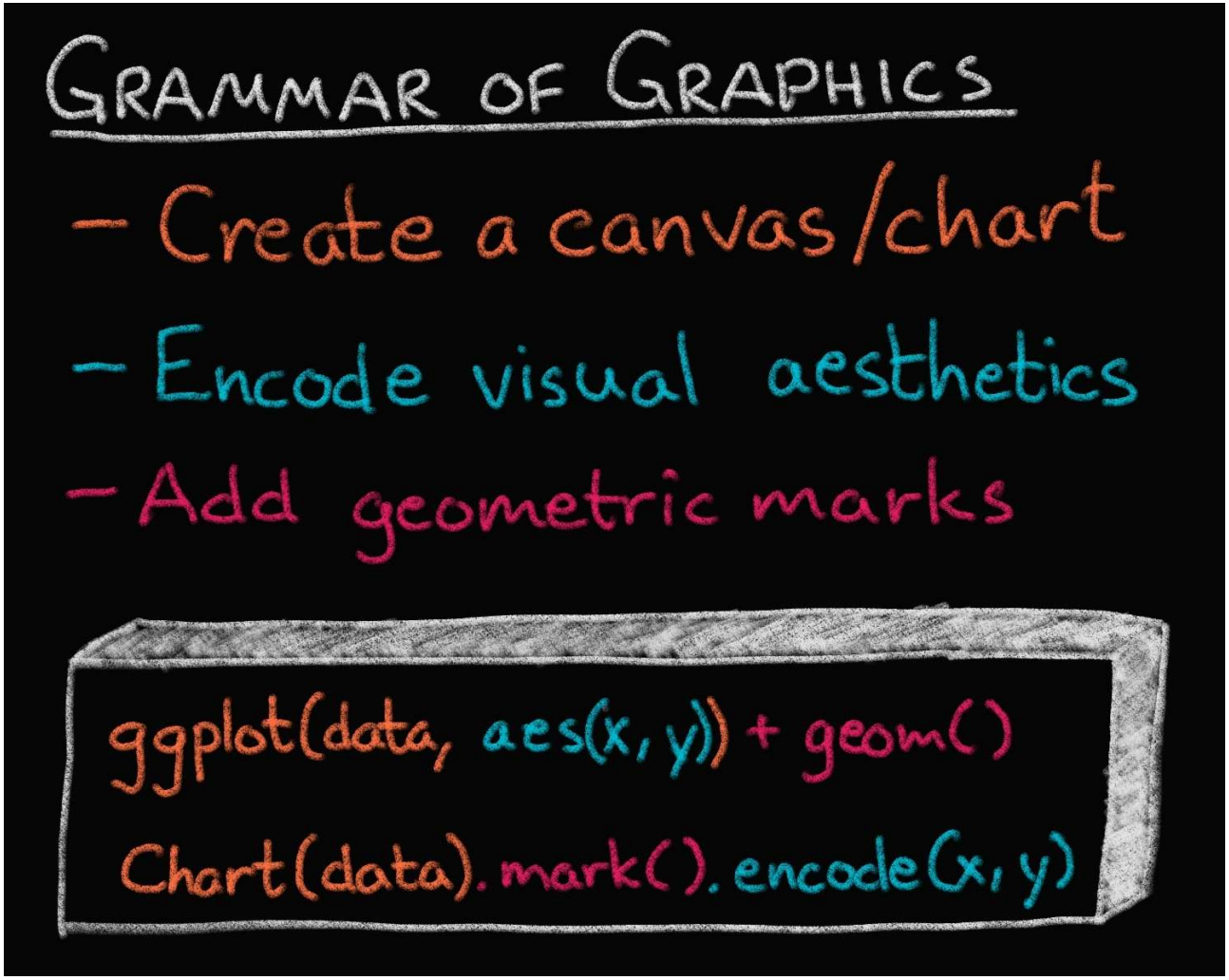
Another common encoding aestheic is `size`.

```
alt.Chart(cars).mark_point().encoc
    x='Weight_in_lbs' ,
    y='Miles_per_Gallon',
    color='Origin',
    shape='Origin',
    size='Horsepower')

# you can do "too much"
# as demonstrated here ...
```



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

Summarize



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

Aggregations, lines, and layers

Using the `cars` data, we'll see how we can:

- aggregate data directly within the Altair plotting functions
- create line plots via the `mark_line` method
- how to combine (i.e. layer) the line and point plots together

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

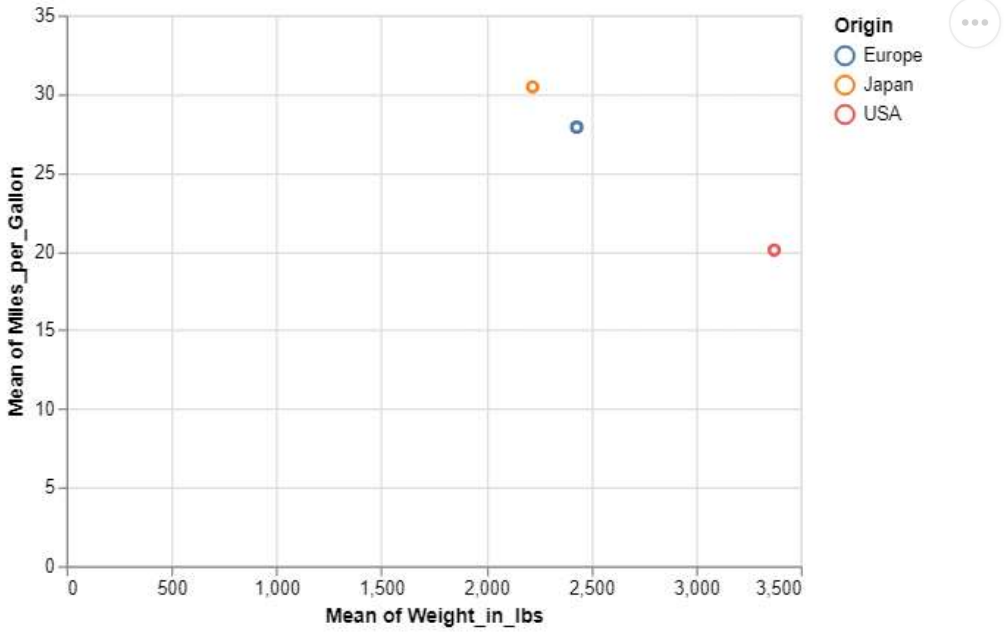
Aggregations

- In addition to simple channel encodings, Altair facilates aggregation functions applied to data features.
- This simplifies verbose code which might otherwise call to `pandas` to first calculate these statistics.
- For example, returning to this example we might want to plot the mean weight (`Weight_in_lbs`) and fuel efficiency (`Miles_per_Gallon`) for each country of car `Origin`.

To see a table with all available aggregations visit the Altair documentation (Binning and
https://github.com/ubco-mds-2022/Data-550

Aggregations demo

```
alt.Chart(cars).mark_point().encode(
    x='mean(Weight_in_lbs)',
    y='mean(Miles_per_Gallon)',
    color='Origin')
```



- As we saw previously, American cars indeed have higher weight and lower fuel efficiency on average.
- We can now detect small differences between the means of the Japanese and Europeans cars

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

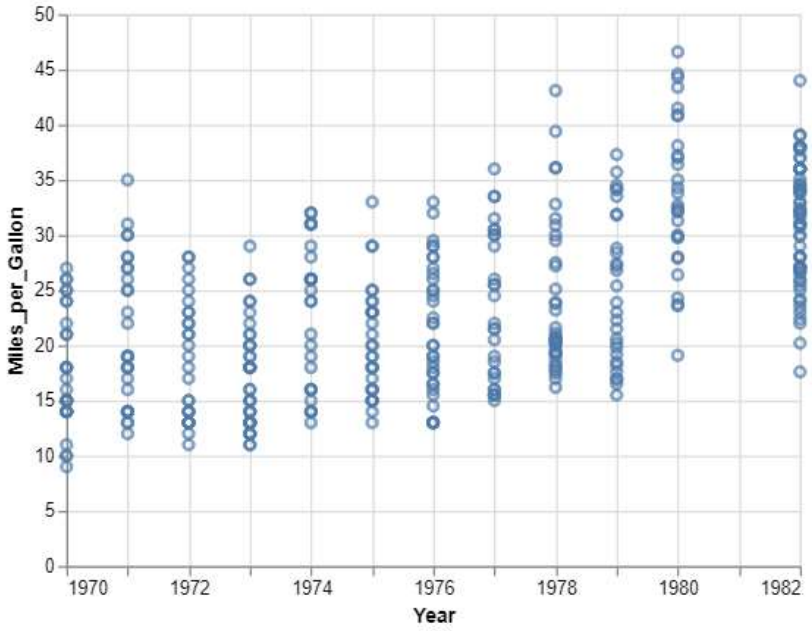
Longitudinal Data

- Aggregations are often helpful when comparing trends over time (especially between multiple groups in your data)
- We can use the point chart to visualize trends over time, such as how the miles per gallon has changes over the years.
- In this case, we might be interested in knowing whether newer cars are more fuel-efficient than older ones.
- Is this answer dependend on where the car was made?

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

Plotting trends over time

```
alt.Chart(cars).mark_point().encode(
    x='Year',
    y='Miles_per_Gallon')
```

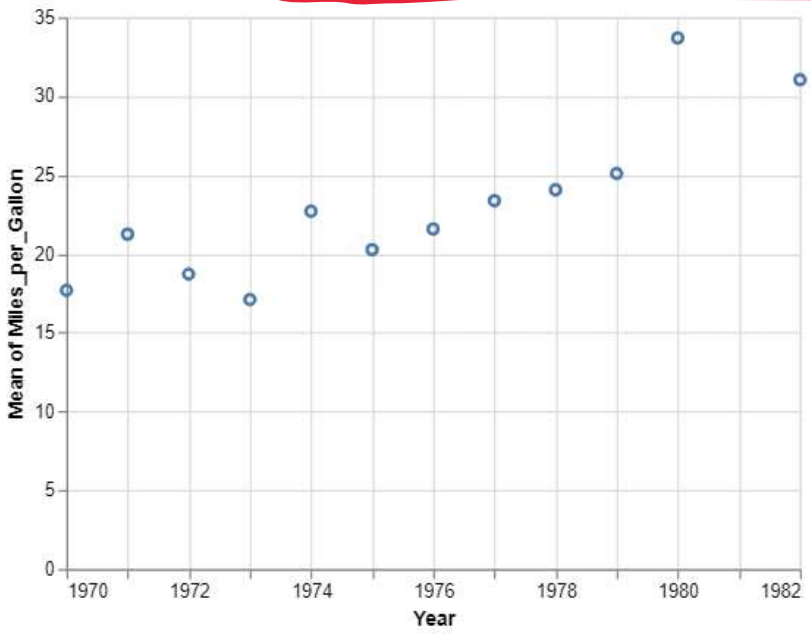


It's difficult to tease out the general trend when all the data are plotted

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

Aggregations over time

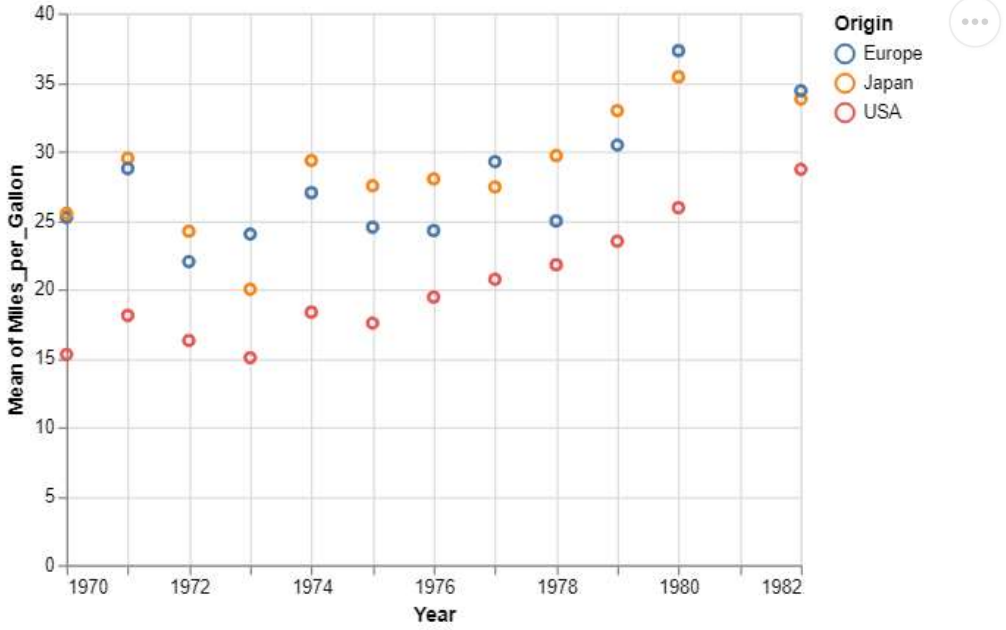
```
alt.Chart(cars).mark_point().encode(
    x='Year',
    y='mean(Miles_per_Gallon)')
```



- We can perform an aggregation on the data which plots the *mean* mileage (over all cars) for each year.
- The general trend is that the fuel efficiency is improving

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

```
alt.Chart(cars).mark_point().encode(
    x='Year',
    y='mean(Miles_per_Gallon)',
    color='Origin')
```

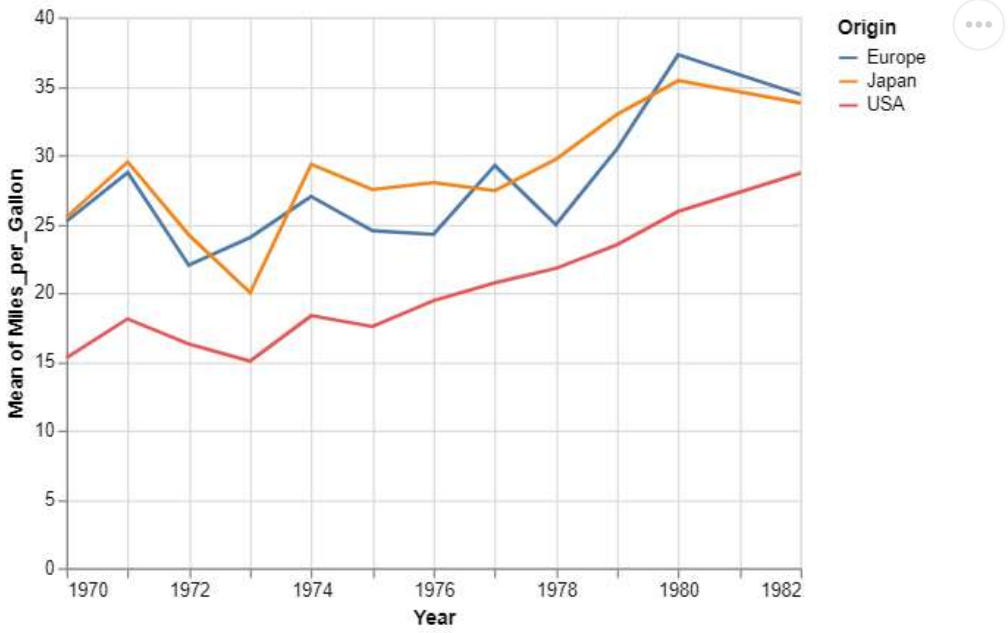


If we try to explore the mileage over time while grouping the cars according to their origin, it is a bit difficult to immediately recognize which points belong to which group.

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

Line plots

```
alt.Chart(cars).mark_line().encode(
    x='Year',
    y='mean(Miles_per_Gallon)',
    color='Origin')
```



One key advantage of line plots is that they connect all the observations that belong to the same group presenting them as one unified graphical object (one line)

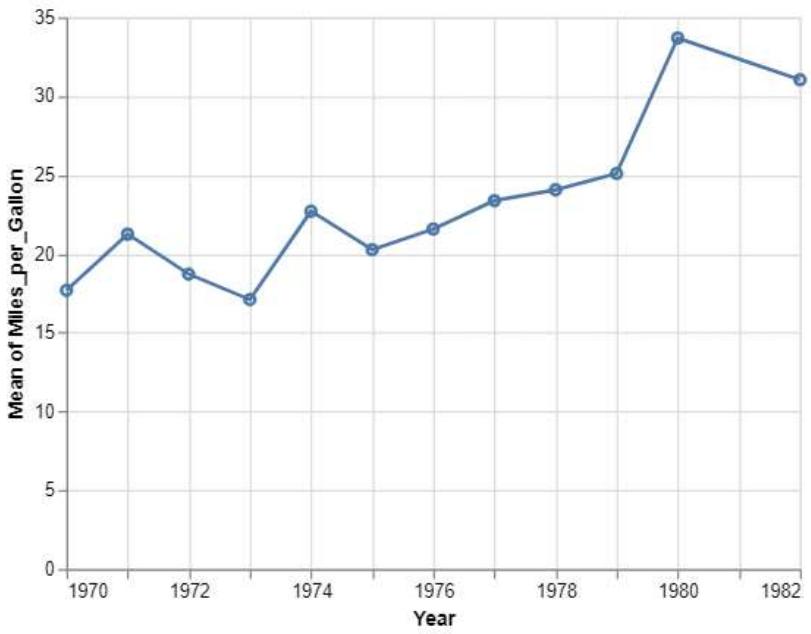
https://github.com/ubco-mds-2022/Data-550

Layering lines with points

Tt is sometimes helpful to add point marks for each data point along the line, to emphasize where the observations fall.

```
line = alt.Chart(cars).mark_line().encode(          # create a line plot
    x='Year',
    y='mean(Miles_per_Gallon)')

point = alt.Chart(cars).mark_point().encode(        # create a point plot
    x='Year',
    y='mean(Miles_per_Gallon)')

line + point                          # combine the two into a layered chart
```

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



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

28

Building on previous plots

We can also create a layered plot by reusing a previous chart definition.

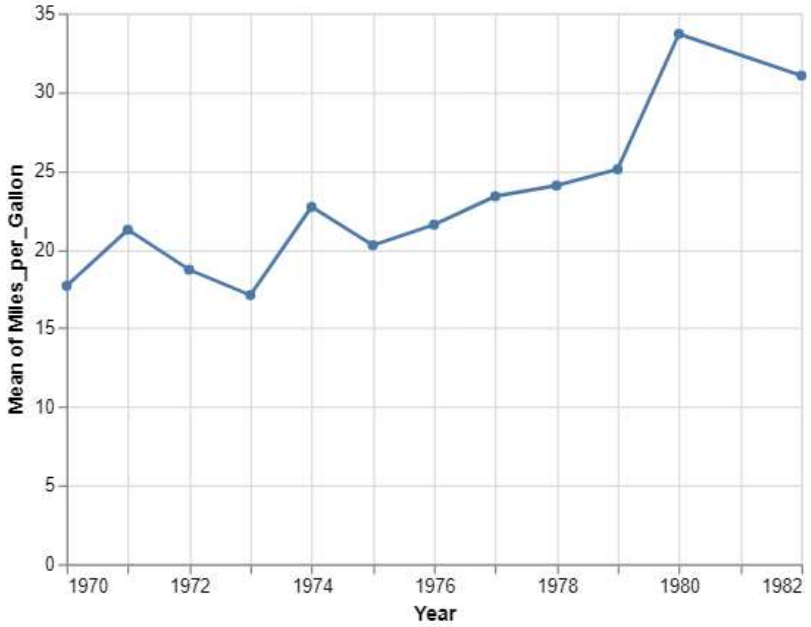
```
line = alt.Chart(cars).mark_line().encode(
    x='Year',
    y='mean(Miles_per_Gallon)')

line + line.mark_point()
```

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

Handy shortcut

Since points and line plots are such a common marking, there is actually a handy shortcut:

```
alt.Chart(cars).mark_line(point=True).encode(
    x='Year',
    y='mean(Miles_per_Gallon)')
```



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

This is just a special case, however, and we will have much more use with the plus operator (+) for layering.

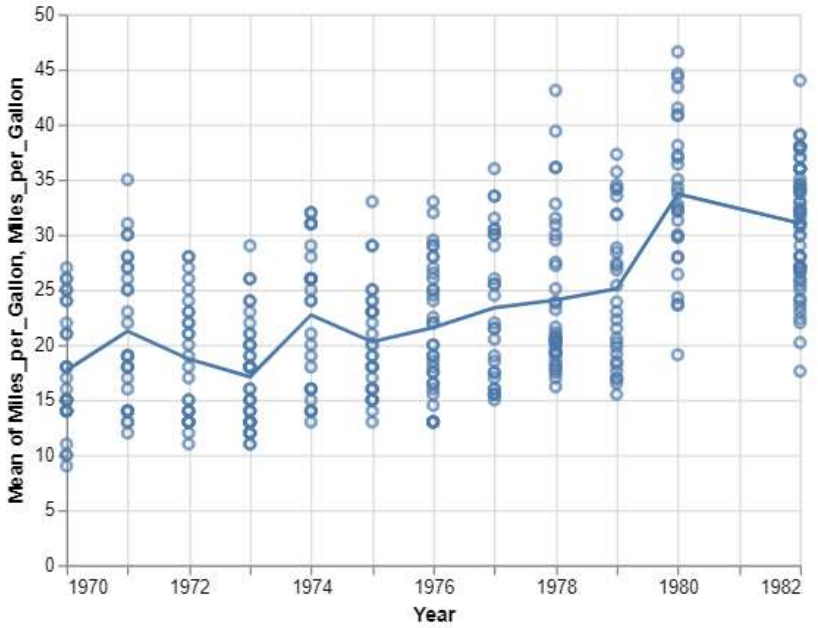
https://github.com/ubco-mds-2022/Data-550

30

Raw values with mean

Showing raw values together with the mean is often helpful

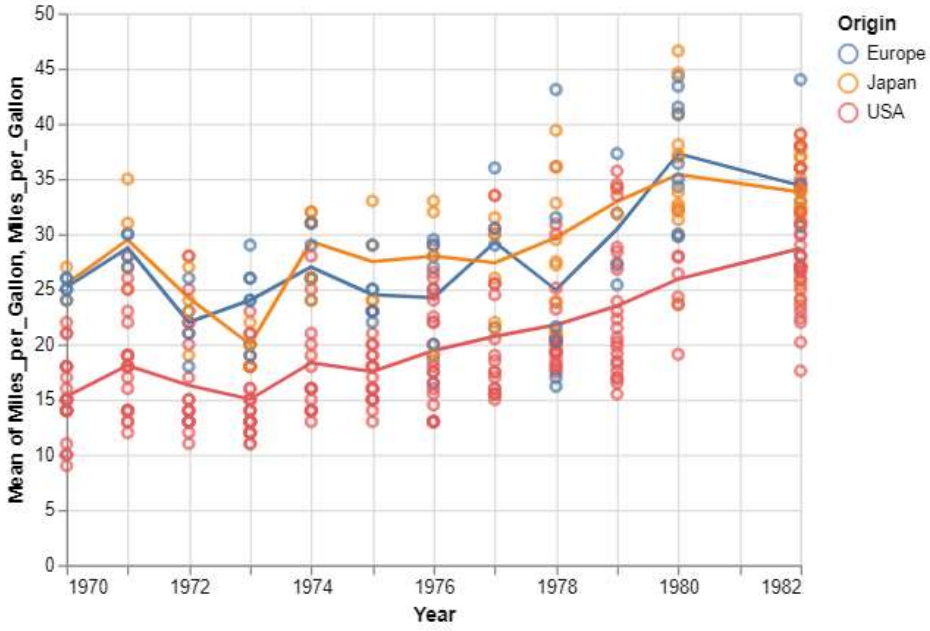
```
line = alt.Chart(cars).mark_line().encode(
    x='Year',
    y='mean(Miles_per_Gallon)')

line + line.mark_point().encode(y='Miles_per_Gallon')
```



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

All encodings of the base chart are propagated unless they are overwritten

```
line = alt.Chart(cars).mark_line().encode(
    x='Year',
    y='mean(Miles_per_Gallon)',
    color='Origin')

line + line.mark_point().encode(y='Miles_per_Gallon')
```



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

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