Visualization - Intro (Lecture 2)

Peter Ganong and Maggie Shi

January 12, 2026

Citing our sources

- Schwabish: "Better Data Visualizations" (link to purchase)
- Healy: "Data Visualization" (link to full text)
- Heer, Moritz, VanderPlas, and Craft `altair` textbook: (link to full text)

Why visualize?

Why visualize?: roadmap

- Introduce **Anscombe’s Quartet** – a classic data visualization example
- Look at the data directly, calculate summary statistics, then plot
- Discuss what observations can be made at each step

Anscombe’s Quartet

- Introduced by statistician **Francis Anscombe** in **1973**
- Consists of **four small datasets** — each with **two variables:** x and y

The raw data

I		II		III		IV	
x	y	x	y	x	y	x	y
4	4.3	4	3.1	4	5.4	8	5.3
5	5.7	5	4.7	5	5.7	8	5.8
6	7.2	6	6.1	6	6.1	8	6.6
7	4.8	7	7.3	7	6.4	8	6.9
8	7.0	8	8.1	8	6.8	8	7.0
9	8.8	9	8.8	9	7.1	8	7.7
10	8.0	10	9.1	10	7.5	8	7.9
11	8.3	11	9.3	11	7.8	8	8.5
12	10.8	12	9.1	12	8.2	8	8.8
13	7.6	13	8.7	13	12.7	8	12.5
14	10.0	14	8.1	14	8.8	19	5.6

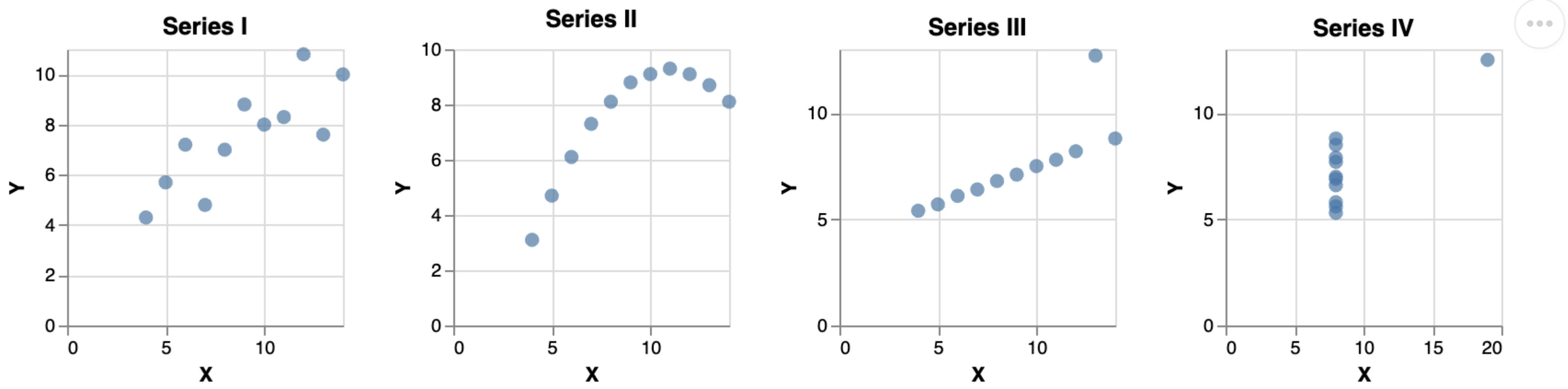
Summary statistics

Let’s also compute some summary statistics

	series	Mean X	Mean Y	Var(x)	Var(y)	Corr(x,y)	Regression
0	I	9.0	7.5	11.0	4.1	0.8	y = 0.50x + 3.0
1	II	9.0	7.5	11.0	4.1	0.8	y = 0.50x + 3.0
2	III	9.0	7.5	11.0	4.1	0.8	y = 0.50x + 3.0
3	IV	9.0	7.5	11.0	4.1	0.8	y = 0.50x + 3.0

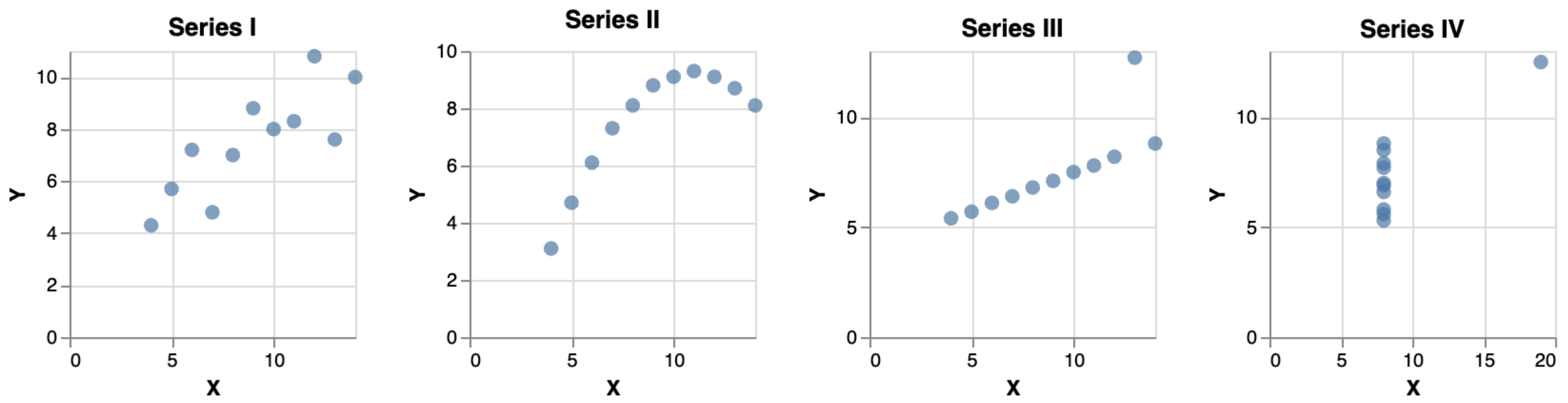
- All four datasets have basically **identical** summary statistics and regression coefficients!
- But are they really capturing the same relationships between x and y?

Anscombe's quartet, visualized



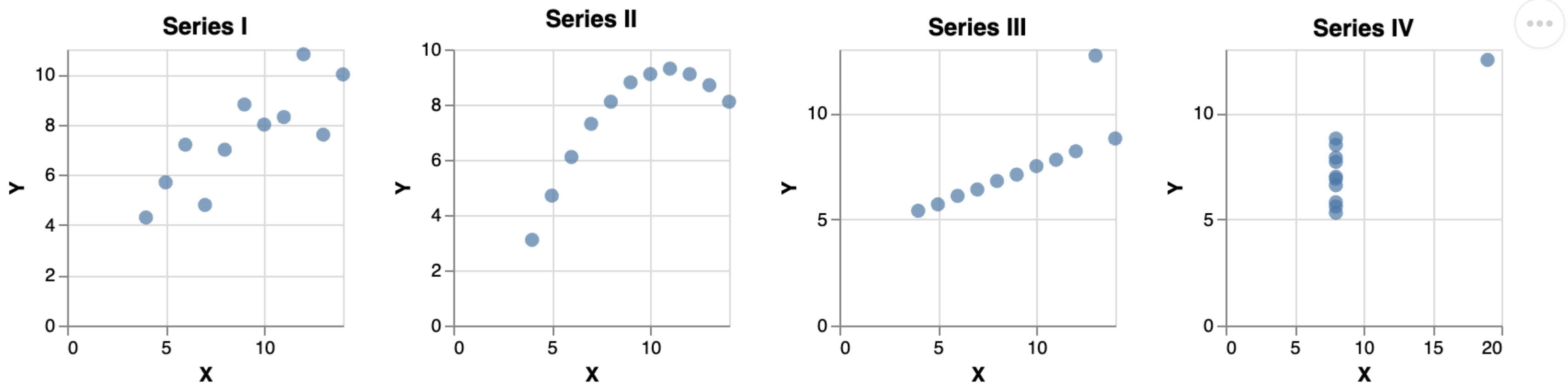
Discussion question: what observations can we now make about the similarities and differences across the series?

Observations from the visualization



- Series I, II, and III illustrate very different positive relationships between x and y
  - I: noisy and approximately linear
  - II: quadratic and fully deterministic
  - III: very linear, with exception of an outlier

Observations from the visualization



- The two outliers in III and IV stand out much more quickly
- The positive relationship suggested by the regression in series IV is just an artifact of the outlier

Why visualize?: summary

- Anscombe’s Quartet illustrates that while summary statistics are useful, we can’t rely on them alone
- The best way to detect patterns is to visualize your data

Introduction to Vega-Lite and `altair`

Roadmap

- What’s different from `matplotlib`?
- What is Vega-Lite?
- What is `altair`?
- First plot – image and then grammar

`matplotlib` is imperative

```python
import matplotlib.pyplot as plt

plt.ylabel("Y axis")
plt.figure(figsize=(6, 4))
plt.title("A Line Plot")
plt.plot([1, 2, 3], [4, 1, 6], color='red', marker='o')
plt.xlabel("X axis")

plt.grid(True)
plt.show()
```

- **Imperative**: you tell computer directly how to draw graph
- Each graphic element is layered on one-by-one, but not organized
- Not clear what type of plot this is
- Not clear how data is being plotted (which axis is which?)

Declarative approaches to visualization

- Good graphics packages are **declarative**: you provide a high-level specification of *what* you want in the visualization.
- Importantly, you do so in an **organized** way.
- Declarative visualization approaches have three inputs:
  - **Data**
  - **Graphical marks**: the "type" of plot – bar, scatter, etc.
  - **Encoding channels**: x-axis, y-axis, colors, etc

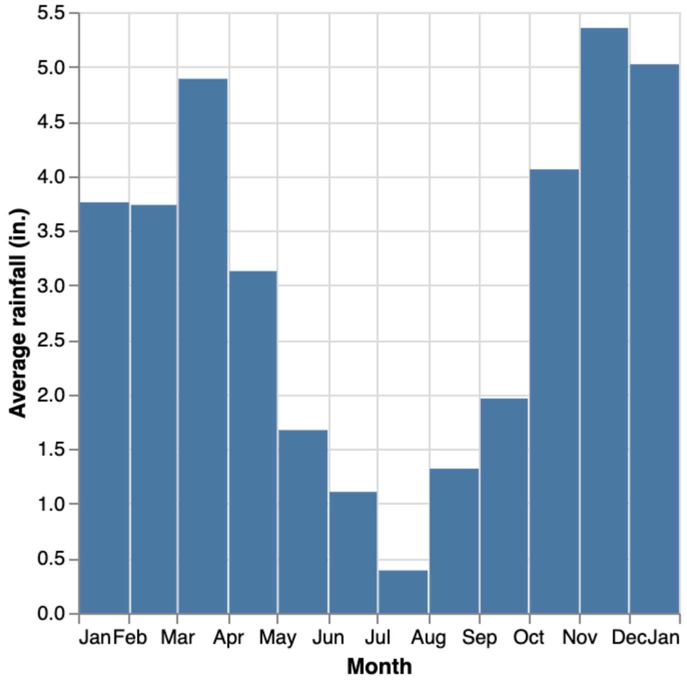
We are going to use *Vega-Lite* + `altair` in this class.

What is Vega-Lite and Altair?

- Vega-Lite is a “grammar” of interactive graphics
  - Under the hood: JSON (JavaScript Object Notation) is used to record Vega-Lite specifications
  - *Note: you will not have to write Vega-Lite manually*
- `altair` is a Python package allows Python to write Vega-Lite

Example: Graph vs. Grammar

Plot:



Source.

Vega-lite (JSON) underlying the plot:

```
{
  "data": {"url": "data/seattle-weather.csv"},
  "mark": "bar",
  "encoding": {
    "x": {"timeUnit": "month", "field": "date"
    "y": {"aggregate": "mean", "field": "preci
  }
}
```

- This is the code that is “under the hood” of the graphic – *not* the Python code you write.
- Discussion question: can you tell what each line of text means?

Why the emphasis on grammar?

- Packages for making graphics and coding languages change over time
- We chose to teach a package with an underlying grammar because we are trying to foreground the *conceptual* aspects of data visualization
- Insights are portable, even as the particular package/language you use changes over time

Summary

- Use a declarative approach
- Grammar: Vega-lite
- Python package to write Vega-lite: `altair`
- Gives a coherent conceptual representation underlying a plot

Introduction to `altair`

What is `altair`? + roadmap

`altair` is a Python API (Application Programming Interface) that generates Vega-Lite specifications.

Roadmap:

- Load package
- Define dataset we’ll work with through rest of class

Imports and Renderer

```python
import pandas as pd
import altair as alt
```

Note: depending on your environment, you may also need to specify a renderer for `altair` (see Leja’s Ed post as one example)

- If you run into this, please read the documentation for Displaying Altair Charts.
- If that fails, post in Ed and bring your question to lab.

Tidy data: weather data

- Visualization in `altair` begins with “tidy” data frames
  - Each variable is a column
  - Each observation is a row
  - Each value is a cell
- A simple data frame (`df`) containing the average precipitation (`precip`) for a given `city` and `month` :

```
df = pd.DataFrame({
     'city': ['Seattle', 'Seattle', 'Seattle', 'Seattle', 'New York', 'New Y
     'month': [1, 4, 8, 12, 1, 4, 8, 12, 1, 4, 8, 12],
     'precip': [3.12, 2.68, 0.87, 5.31, 2.1, 3.94, 4.13, 3.58, 3.3, 3.62, 3.
})
```

Tidy data: weather data

```
df.head()
```

	city	month	precip
0	Seattle	1	3.12
1	Seattle	4	2.68
2	Seattle	8	0.87
3	Seattle	12	5.31
4	New York	1	2.10

Aside: in-class examples and tidy data

- Data you will work with is most often *not* as tidy as the in-class examples
- Much of the work you will do as an analyst is kind of this “unglamorous” work!
- Using `altair` to make the plot is often the last step
  - In lecture, we’ll skip the data cleaning to focus on ideas and skills about how to do visualization well
  - In problem sets + project, you will start with messy data that will require cleaning
  - Why don’t we cover data cleaning in lecture? Every dataset needs different steps for cleaning; best way to teach it is therefore through hands-on exercises in the problem sets.

Summary

- `altair` is an API that enables Python to “speak” in Vega-Lite’s grammar
- Input to `altair`: tidy data

Building a first chart

Building a first chart: roadmap

- Incrementally build our first chart in `altair`
- Then build our first aggregated chart

`altair` ingredients

Recall that we need three inputs to a *Vega-lite* chart:

- **Data**
- **Graphical marks**: the “type” of plot - bar, scatter, etc.
- **Encoding channels**: x-axis, y-axis, colors, etc

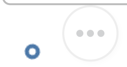
Data: the `Chart` object

```
mychart = alt.Chart(df)
```

- We have defined the `mychart` object and passed it the data
- But nothing has been plotted yet – still need mark and encoding

Mark: point

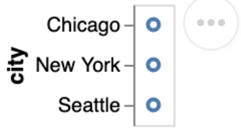
```python
mychart = alt.Chart(df).mark_point()

mychart
```



- Now, declare the mark: a `mark_point()`
- We haven’t declared what’s on the axes
- So actually this is *all the data points*, located in the same place

Encoding: one point per city on y-axis

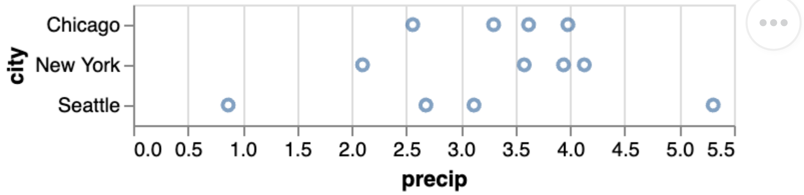
```python
mychart = alt.Chart(df).mark_point().encode(
  alt.Y('city')
)

mychart
```



- Using `alt.Y`, we’ve separated data by one attribute along the y-axis: city
- We haven’t encoded anything on the x-axis yet!
- So underneath each point, we have multiple points overlapping within each city

Encoding: xy coordinates

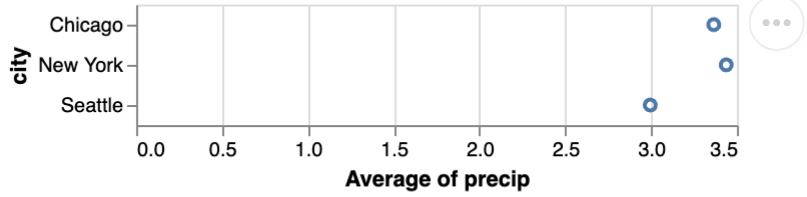
```
mychart = alt.Chart(df).mark_point().encode(
  alt.X('precip'),
  alt.Y('city')
)

mychart
```



- Process: Code is super-duper readable.
- Note that we’ve directly plotted all the data
- Substance: *Seattle exhibits both the least-rainiest and most-rainiest months!*

Quick Aggregation

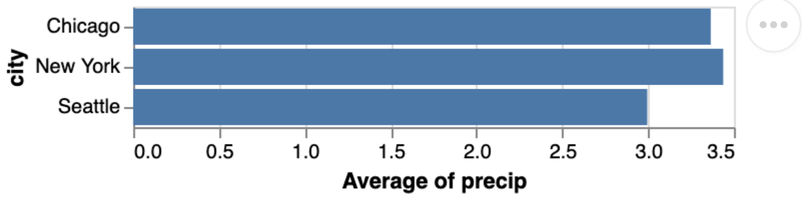
```
mychart = alt.Chart(df).mark_point().encode(
  alt.X('average(precip)'),
  alt.Y('city')
)

mychart
```



- Say instead of plotting all data points, we want to transform and aggregate it first
- `altair` has a few "shorthand" aggregation functions. like `average()`
- We will discuss more complicated aggregation in week 3

Changing Marks is Straightforward

```
mychart = alt.Chart(df).mark_bar().encode(
  alt.X('average(precip)'),
  alt.Y('city')
)

mychart
```



Syntax: Understanding Altair’s shorthands.

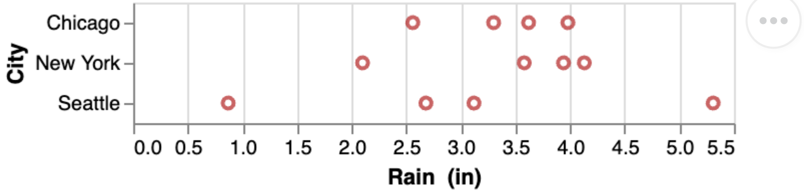
Three ways to say the same idea

```python
# what we will continue to use
alt.X('average(precip)')

# shorter
x = 'average(precip)'

# longer
alt.X(aggregate='average', field='precip', type='quantitative')
```

- Going forward, we will primarily use the first
- But you may see all three in problem sets, lab assessments, etc.

Customizing a plot: colors and labels

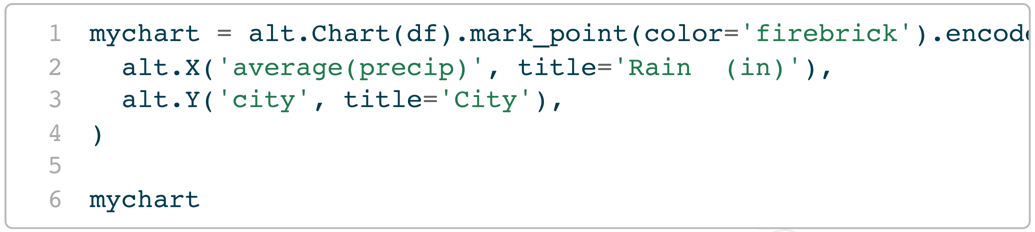
```
alt.Chart(df).mark_point(color='firebrick').encode(
  alt.X('precip', title='Rain  (in)'),
  alt.Y('city', title='City'),
)
```

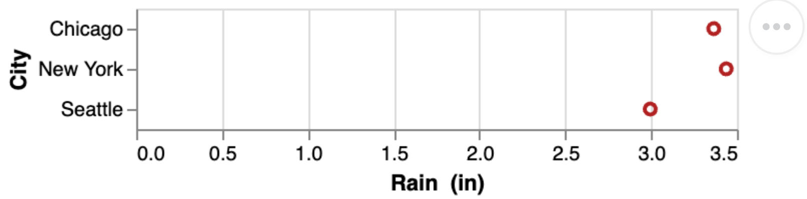


- Add customizations and labels where you declare the relevant mark or encoding
- This is what we mean by “grammar” – code is organized in a consistent, readable way

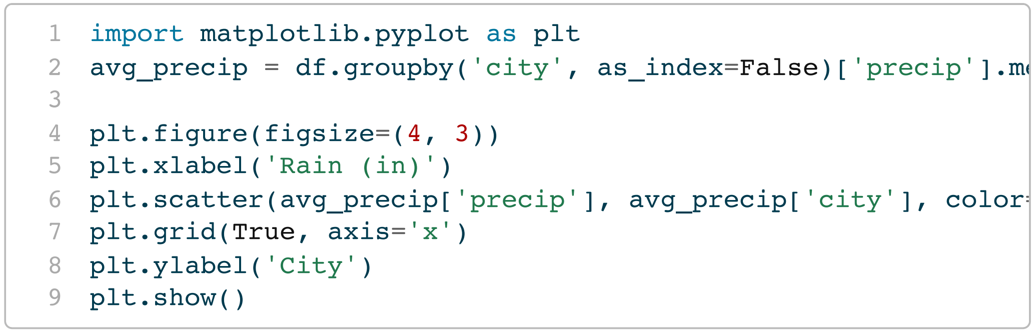
Comparison to `matplotlib`

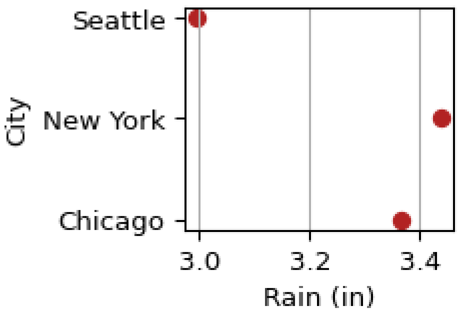
altair


```
mychart = alt.Chart(df).mark_point(color='firebrick').encod
  alt.X('average(precip)', title='Rain  (in)'),
  alt.Y('city', title='City'),
)

mychart
```



matplotlib


```
import matplotlib.pyplot as plt
avg_precip = df.groupby('city', as_index=False)['precip'].me

plt.figure(figsize=(4, 3))
plt.xlabel('Rain (in)')
plt.scatter(avg_precip['precip'], avg_precip['city'], color=
plt.grid(True, axis='x')
plt.ylabel('City')
plt.show()
```



- matplotlib: have to first make a separate, collapsed pandas dataframe
- x and y labels defined separately from code for plot

Visualization guidelines

- All axes and units are properly labeled and legible
- No words or data points are cut off in your final output
- Encodings should be sensible/appropriate – *more in this next lecture*

Building a first chart: summary

- Everything begins with a `Chart(data)`
- Every `Chart` needs a `mark`
- Every `Chart` needs guidance how to encode the data in terms of `mark`s
- Simple chart formatting: `mark_point(color='firebrick')`, `alt.X(title = '...')`

Do-pair-share

- *Do* – make a plot on your own
- *Pair* – compare your results with person next to you
- *Share* – discuss results as a class

1. Open `viz_1_intro/viz_1_dps.qmd` in VSCode
2. If you have the Quarto and Jupyter extensions installed + `dap` conda environment set up (link), you should be able to directly “Run Cell” within `.qmd` file

Data Transformation: Do-pair-share

- Make a bar plot showing the **lowest** rainfall for each city in the dataset.
- Starter code in `viz_1_intro/viz_1_dps.qmd` file in student repo:

```python
import pandas as pd
import altair as alt
df = pd.DataFrame({
    'city': ['Seattle', 'Seattle', 'Seattle', 'Seattle', 'New York', 'New York', 'New York', 'N
    'month': [1, 4, 8, 12, 1, 4, 8, 12, 1, 4, 8, 12],
    'precip': [3.12, 2.68, 0.87, 5.31, 2.1, 3.94, 4.13, 3.58, 3.3, 3.62, 3.98, 2.56]
})
```

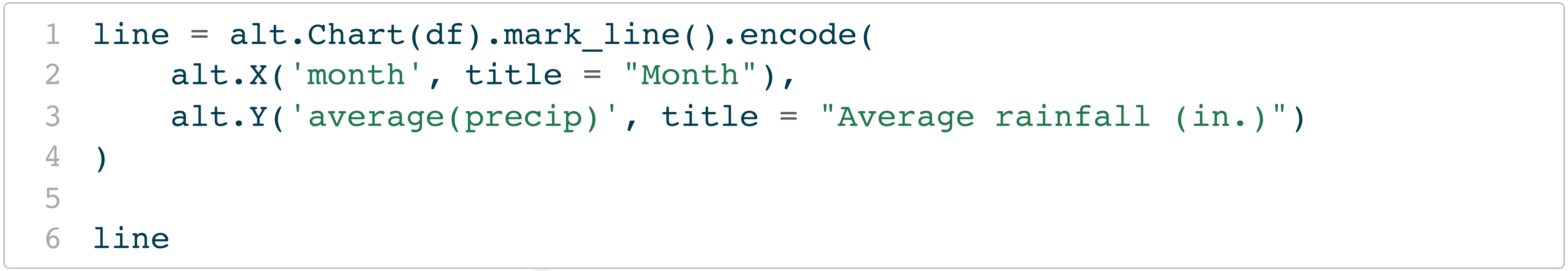
- Hint: Altair’s aggregation methods are here

Multiple Views

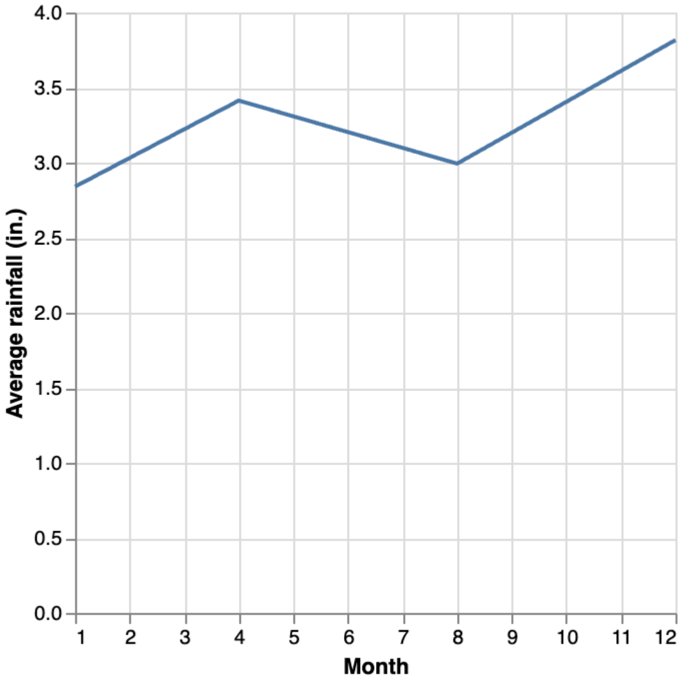
Multiple Views: roadmap

- Introduce `mark_line()`
- Multiple marks
- Multiple panels

mark_line()



```python
line = alt.Chart(df).mark_line().encode(
    alt.X('month', title = "Month"),
    alt.Y('average(precip)', title = "Average rainfall (in.)")
)

line
```

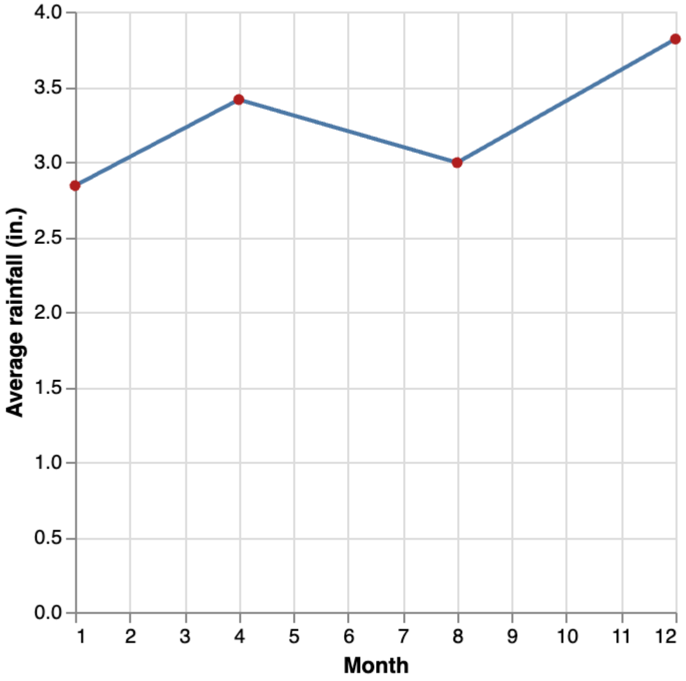


Multiple Marks

- Now say that we want to add layer scatter points on top of our line to highlight the unit of observation
- `altair` grammar allows you to “layer” one set of marks on another with intuitive syntax: `+`

Multiple Marks

```
line = alt.Chart(df).mark_line().encode(
    alt.X('month', title = "Month"),
    alt.Y('average(precip)', title = "Average rainfall (in.)")
)
point = alt.Chart(df).mark_circle(color='firebrick').encode(
    alt.X('month', title = "Month"),
    alt.Y('average(precip)', title = "Average rainfall (in.)")
)
line + point
```

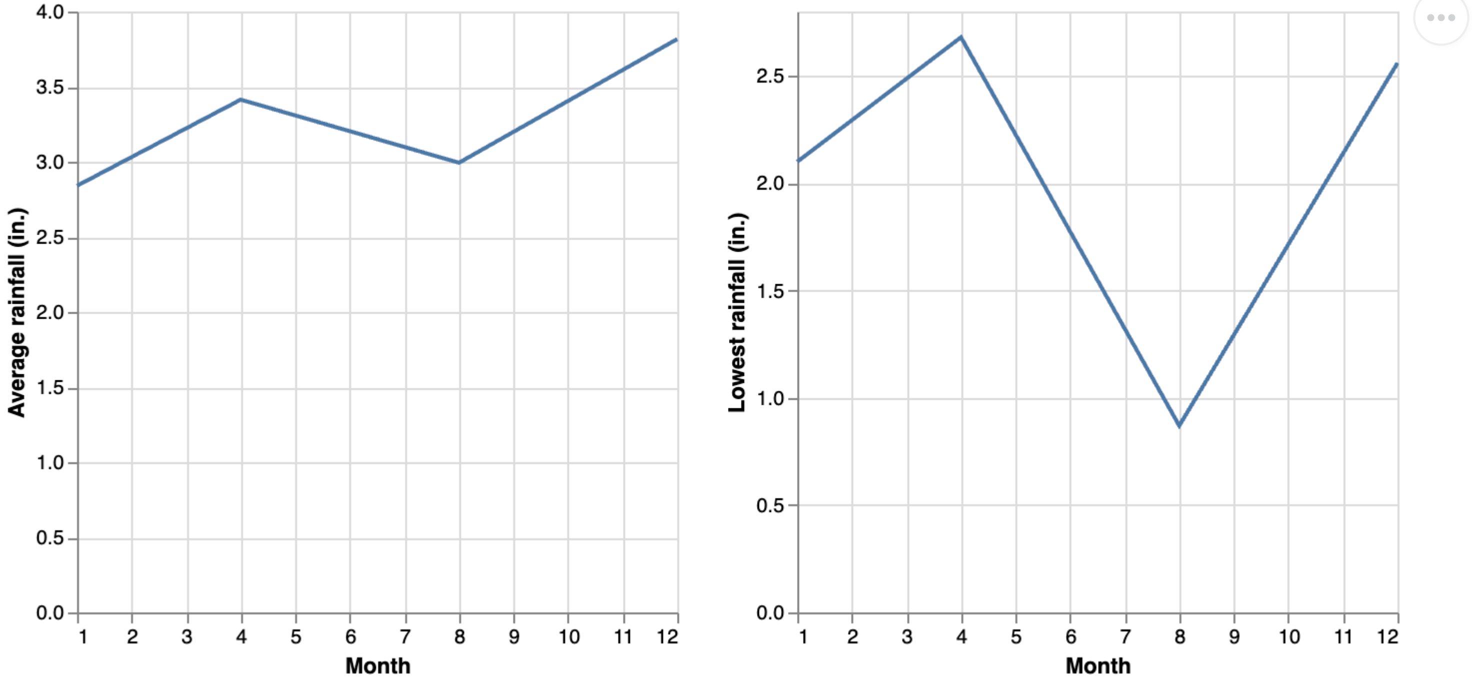


Multiple Panels

- Now say we want to place another plot *next to* our original plot
- Again, `altair` grammar allows for fairly intuitive syntax: `|`

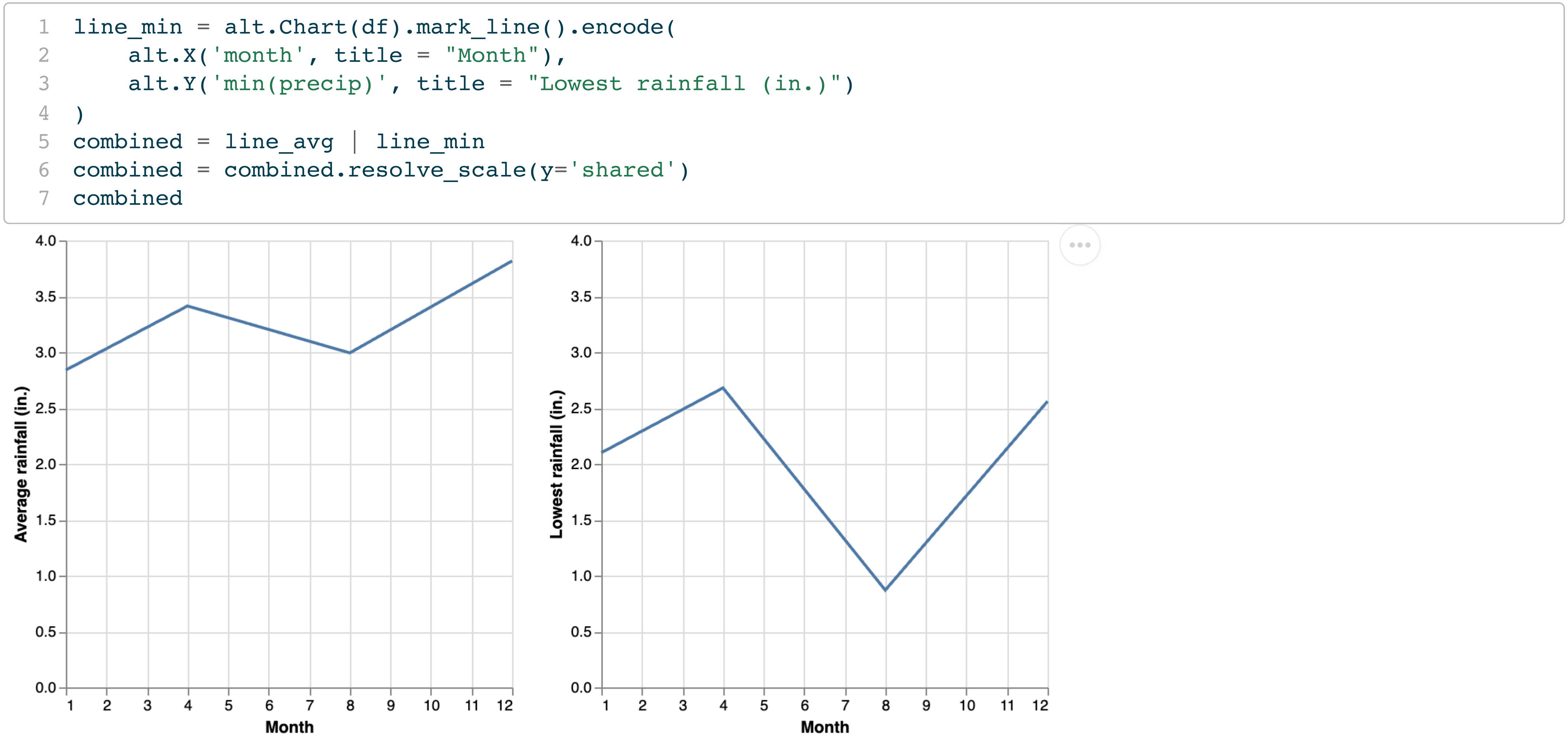
Multiple Panels

```
line_min = alt.Chart(df).mark_line().encode(
    alt.X('month', title = "Month"),
    alt.Y('min(precip)', title = "Lowest rainfall (in.)")
)

line_avg | line_min
```



Discussion: while technically correct, there's something not quite right about these charts…

Multiple Panels, take 2



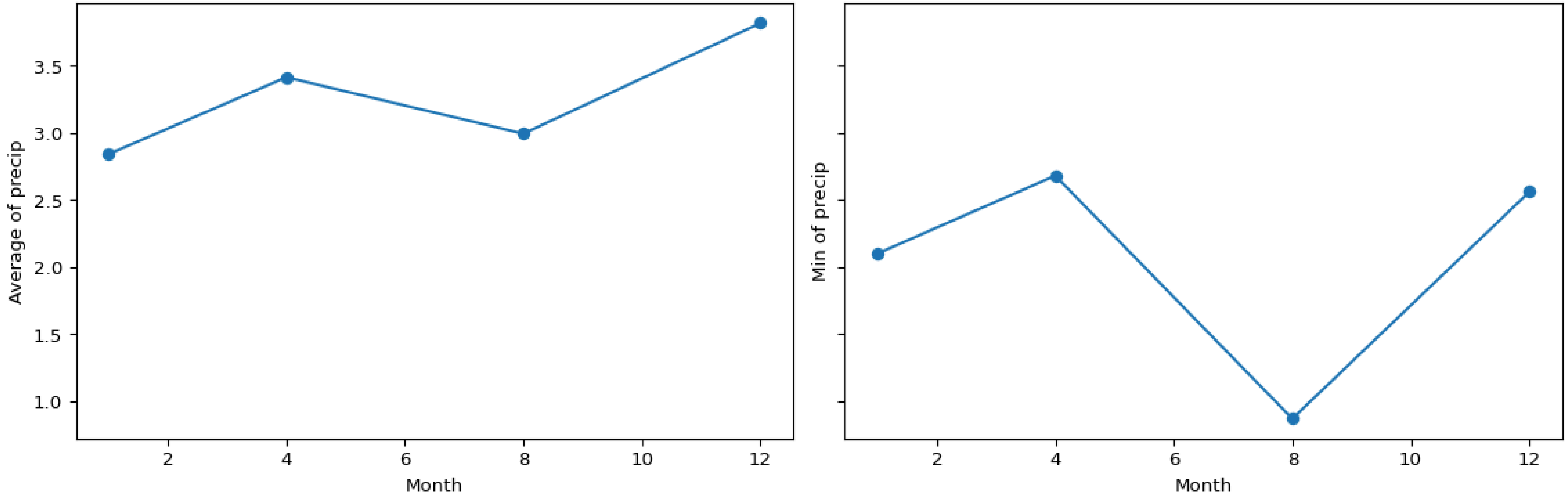
```python
line_min = alt.Chart(df).mark_line().encode(
    alt.X('month', title = "Month"),
    alt.Y('min(precip)', title = "Lowest rainfall (in.)")
)
combined = line_avg | line_min
combined = combined.resolve_scale(y='shared')
combined
```

`resolve_scale()` is applied to `combined`, not `line_avg` or `line_min`

Organized and easy to understand!

Equivalent graph in `matplotlib`

Of course, you can make a very similar plot in `matplotlib`



Equivalent code in `matplotlib`

```python
avg_by_month = df.groupby('month', as_index=False)['precip'].mean()
fig, axs = plt.subplots(1, 2, figsize=(12, 4), sharey=True)
axs[1].set_ylabel('Min of precip')
axs[0].plot(avg_by_month['month'], avg_by_month['precip'])
min_by_month = df.groupby('month', as_index=False)['precip'].min()
axs[1].scatter(min_by_month['month'], min_by_month['precip'])
axs[0].scatter(avg_by_month['month'], avg_by_month['precip'])
axs[0].set_ylabel('Average of precip')
axs[0].set_xlabel('Month')
axs[1].plot(min_by_month['month'], min_by_month['precip'])
axs[1].set_xlabel('Month')

plt.tight_layout()
plt.show()
```

Multiple Views – summary

- New syntax:
  - `plot1 + plot2` for multiple marks on same panel
  - `plot1 | plot2` for multiple panels
- `altair` allows you to combine multiple plots in an organized and intuitive way
- Usefulness of grammar and delcarative approach becomes more apparent when extending beyond a single plot